Trident 24.10文档

Trident

NetApp
March 04, 2026

目录

Trident 24.10文档

发行说明

新增功能

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

> `tridentctl`安装程序zip文件中提供的Linux二进制文件是经过测试且受支持的版本。请注意、`macos`此zip文件中提供的二进制 `/extras`文件未经过测试或不受支持。

24.10中的新增功能

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。
- 为LUKS LUN增加了18MiB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

    "详细了解 Trident Protect"。
- 为安装程序添加了一个新标志 `--k8s_api_qps`、用于设置Trident与Kubbernetes 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。请参阅 "光纤通道支持"。

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中增加了对默认情况下使用REST而不是ZAPI for ONTAP 9.15.1及更高版本的支持。
- 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20MiB的卷大小"问题[#883."()。
- 修复了Trident、以防止在对Flex-SAN驱动程序执行调整大小操作期间缩减ONTAP。
- 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()的功能"问题描述#864"。
- 在ONTAP驱动程序中添加了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

- 固定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ONTAP－NAS和ONTAP－NAS－FlexGroup存储驱动程序时仅显示卷的可用大小"问题编号：第请"()。
- 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中的变化

修复

Kubernetes

- 修复了Trident升级以忽略处于终止状态()的旧Pod的问题"问题描述#740"。
- 在“瞬时Trident版本POD”定义中增加了容差"问题编号"()。

Trident

- 修复了ONTAP 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()中的错误处理"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。

- 将发布信息字段从还原 `rawDevicePath`到 `devicePath`；添加了用于填充和恢复(在某些情况下)字段的逻辑 `devicePath`。

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#出在#原因"()。

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(、"#543.")，则更新后端状态"问题801"。
- 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中的变化

> 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Trident安装程序标志启用强制断开 `--enable-force-detach`。

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。
- 固定的Trident操作员集群角色权限与捆绑包权限同步"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Trident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"问题描述#781"问题()。
- 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Trident确定在不影响活动工作负载的情况下可以安全地切换到新的igrop方案时，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"()。
- 通过首先停用API服务器，改进了Trident关闭过程"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。
- 固定的Trident操作员集群角色权限与捆绑包权限保持同步"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。

23.01中的变化

现在、在Trident中支持Kubnetes 1.27。请先升级Trident、然后再升级Kubernetes。

修复

- Kubelnetes：添加了用于排除Pod安全策略创建的选项，以通过Helm修复Trident安装"问题783、794"()。

增强功能

Kubernetes

- 增加了对Kubnetes 1.26的支持。
- 提高了整体Trident RBAC资源利用率"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ONONTAP -NAS存储驱动程序中增加了对使用Amazon FSX for ONTAP 的SMB卷的支持。
- 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。
- 在管理多个FlexVol时提高了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。
- 在配置或 `ontap-san-economy`驱动程序时、不应再指定数据 `ontap-san`文件。

22.10中的变化

*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在Multipath.conf文件中使用 `find_multipaths: yes`或 `find_multipaths: smart`值将导致挂载失败。自21.07版本以来、Trident已建议使用 `find_multipaths: no`。

修复

- 修复了特定于使用字段无法联机升级22.07.0 ("问题描述#759")创建的ONTAP后端的问题 `credentials`。
- **Docker**：修复了导致Docker卷插件在某些环境(和"问题描述#760")中无法启动的问题"问题描述#548"。
- 修复了ONTAP SAN后端专用的SLM问题描述 、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"。
- 现在、所有Trident存储驱动程序均 `ontap-*`可与ONTAP REST API结合使用。
- 添加了(`bundle_post_1_25.yaml`不带A的 `PodSecurityPolicy`新运算符YAML以支持Kubnetes 1.25。
- 添加了"支持LUKS加密卷" `ontap-san`和 `ontap-san-economy`存储驱动程序。
- 增加了对Windows Server 2019节点的支持。
- 已通过 `azure-netapp-files`存储驱动程序添加"支持Windows节点上的SMB卷"。
- 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。
- 删除了为iSCSI配置工作节点多路径时对的支持 `yes`和 `smart`选项 `find_multipaths`。

22.07中的变化

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

- **Kubenetes:**已将支持的最小Kubenetes更新为1.19。
- 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中的变化

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"。

如果要从任何早期Trident版本升级并使用Azure NetApp Files、则`location`config参数现在是一个必需的单个字段。

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问题编号"
- 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#691"
- 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标志时提取自动支持映像的问题 `imageRegistry`。()"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中的变更

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#691"
- 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中的变更

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。
- 如果使用括号指定 ONTAP SAN IPv6 数据 LIF ，则此 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Kubnetes 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中的变更

> 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中的变更

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：
  - 使用和 `ontap-nas-flexgroup`驱动程序创建快照时、返回卷的已用空间作为最小可还原大小 `ontap-nas`(GitHub问题645)。
  - 修复了在调整卷大小后记录错误的问题 `Failed to expand filesystem`(GitHub问题560)。
  - 修复了POD可能停留在状态的问题 `Terminating`(GitHub问题572)。
  - 修复了FlexVol可能已满Snapshot LUN的问题 `ontap-san-economy`(GitHub问题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

- 增加了对GCP-CVS性能卷选项的支持 `unixPermissions`。
- 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Kubnetes 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 ）。
  - 在映像命令中添加了操作员 `tridentctl`映像(GitHub问题570)。

实验增强功能

- 在驱动程序中增加了对卷复制的支持 `ontap-san`。
- 为、 `ontap-san`和 `ontap-nas-economy`驱动程序增加了*tech preview* REST支持 `ontap-nas-flexgroup`。

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。这对于RERS/RedHat CoreTM OS实例是如此、它需要 `discard` "挂载选项"；请确保您的包含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。

- 如果将驱动程序与OpenShift 4.5结合使用 `solidfire-san`、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 24.10，则可从获取以前版本的文档"Trident支持生命周期"。

- "Trident 24.06"
- "Trident 24.02"
- "Trident 23.10"
- "Trident 23.07"
- "Trident 23.04"
- "Trident 23.01"
- "Trident 22.10"
- "Trident 22.07"
- "Trident 22.04"

开始使用

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支持在公有云或内部环境中的所有常见NetApp存储平台上使用和管理存储资源、包括ONTAP (AFF、FAS、Select、云、Amazon FSx for NetApp ONTAP)、Element软件(NetApp HCI、SolidFire)、Azure NetApp Files服务和Google Cloud上的Cloud Volumes Service。

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"。

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"是由NetApp提供支持的企业级Azure文件共享服务。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"存储管理员可以通过保障性能并简化存储占用空间来整合工作负载。

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 系统采用全闪存，混合或全 HDD 配置，并提供多种不同的部署模式，包括专门设计的硬件（ FAS 和 AFF ），白盒（ ONTAP Select ）和纯云（ 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。

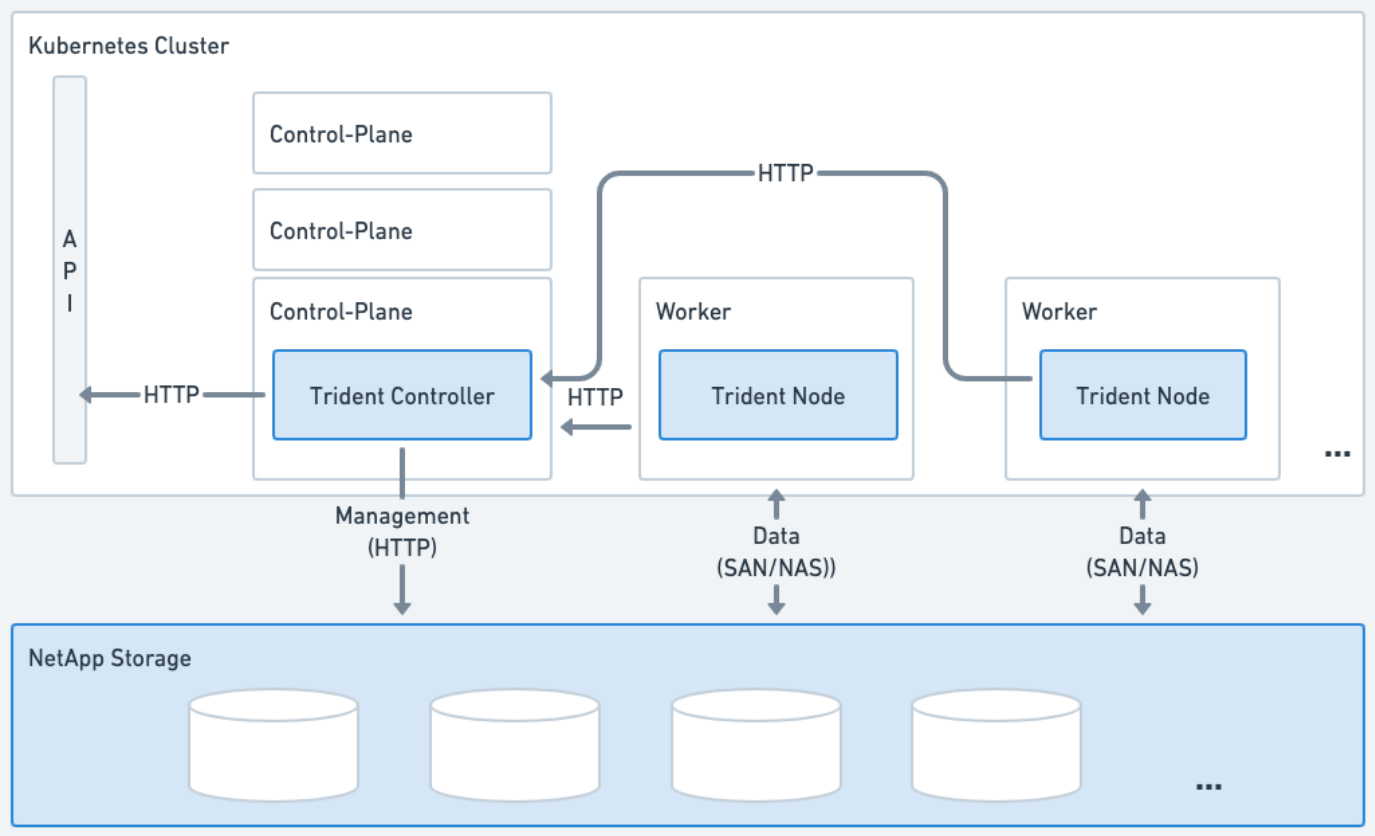


图 **1. Trident**部署在**Kubbernetes**集群上

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。

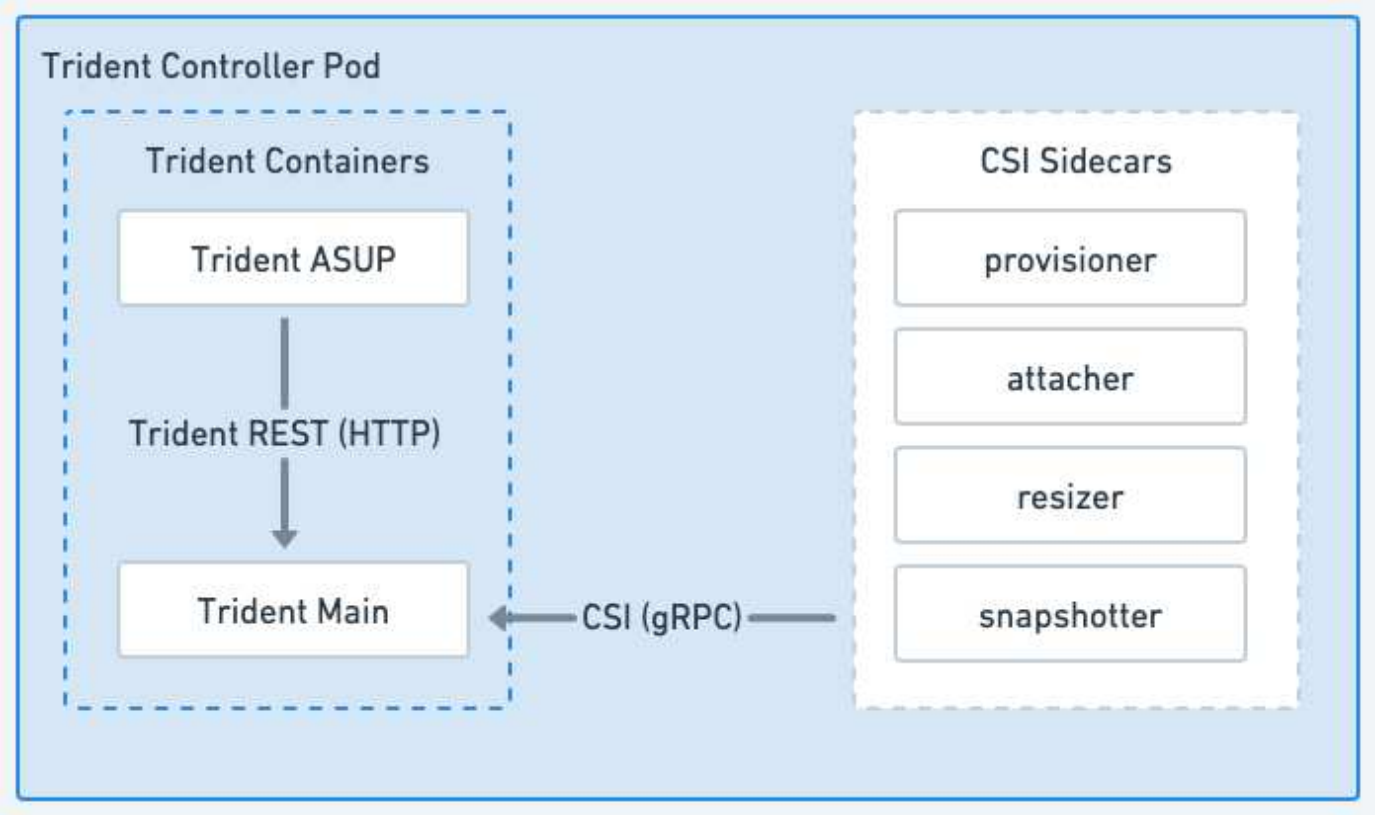


图 **2. TRIdent**控制器**Pod**示意图

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

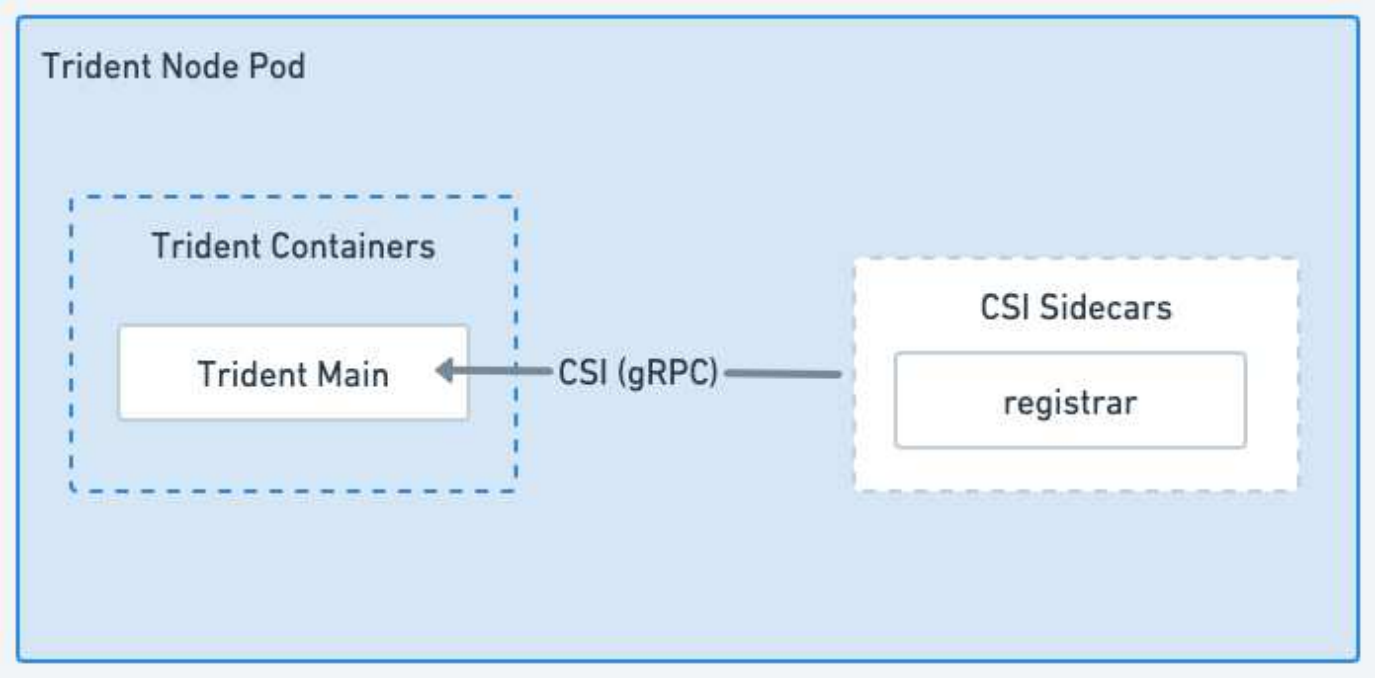


图 **3. TRIdent**节点**Pod**图

支持的 **Kubernetes** 集群架构

以下Kubenetes架构支持Trident：

Kubernetes 集群架构	支持	默认安装
单个主节点，计算节点	是	是
多主机，计算	是	是
主、 `etcd`、计算	是	是
主机，基础架构，计算	是	是

概念

配置

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。

创建卷

然后、Trident使用存储类和存储池之间的关联来确定在何处配置卷。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。

卷快照

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

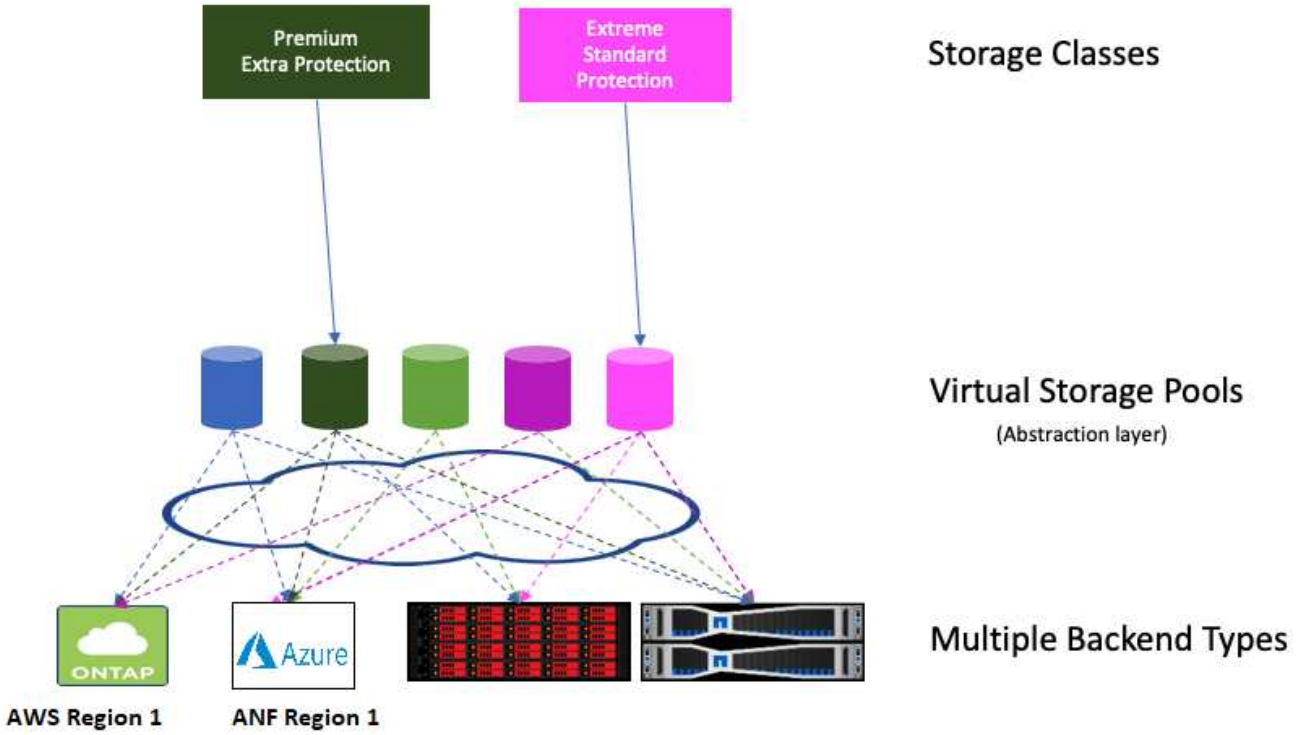
- 对于 `ontap-nas`、、 `ontap-san` `gcp-cvs`和 `azure-netapp-files`驱动程序，每个永久性卷(PV)都会映射到FlexVol。因此，卷快照会创建为 NetApp 快照。与竞争对手的快照技术相比、NetApp快照技术可提供更高的稳定性、可扩展性、可恢复性和性能。无论是在创建 Snapshot 副本所需的时间还是在存储空间中，这些 Snapshot 副本都极为高效。
- 对于 `ontap-nas-flexgroup`驱动程序、每个永久性卷(PV)都会映射到一个FlexGroup。因此，卷快照会创建为 NetApp FlexGroup 快照。与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会映射到在共享FlexVol上创建的LUN。可以通过对关联 LUN 执行 FlexClones 来实现 PV 的卷快照。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/LUN集群上创建的NetApp HCI 。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。

通过Trident、您可以使用卷快照创建新的PV.通过对支持的 ONTAP 和 CVS 后端使用 FlexClone 技术，可以从这些快照创建 PV 。从快照创建PV时、备份卷是快照父卷的FlexClone。此 `solidfire-san`驱动程序使用Element软件卷克隆从快照创建PV.此时，它将从 Element 快照创建一个克隆。

虚拟池

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。

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，Aspect值不会在后端驱动程序之外公开，也不能在中进行匹配 `StorageClasses`。相反，管理员为每个虚拟池定义了一个或多个标签。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。

`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
in	位置（东部，西部）	位于一组值中

运算符	示例	池的标签值必须：
`notin`	性能注释（银牌，铜牌）	不在值集内
`<key>`	保护	存在任何值
`!<key>`	！保护	不存在

卷访问组

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"。

如果您要将配置从使用默认访问组的配置修改为也使用其他访问组的配置 `trident`、请在列表中包括访问组的ID `trident`。

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。

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

- 现在、在Trident中支持Kubnetes 1.32。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在Multipath.conf文件中使用 `find_multipaths: yes`或 `find_multipaths: smart`值将导致挂载失败。自21.07版本以来、Trident已建议使用 `find_multipaths: no`。

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
- Kubbernetes 1.25 - 1.32
- OpenShift 4.10 - 4.17
- R能 手Kubernetes Engine 2 (RKE2) v1.28.5+rke2r1

以下版本支持 Trident 操作符：

- Anthos On－Prem (VMware)和Anthos on Bare metal 1.16
- Kubbernetes 1.25 - 1.32
- OpenShift 4.10 - 4.17
- R能 手Kubernetes Engine 2 (RKE2) v1.28.5+rke2r1

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1.24升级到1.25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

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

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	1.25 - 1.32	否
卷快照	1.25 - 1.32	否
卷快照中的 PVC	1.25 - 1.32	否
iSCSI PV 调整大小	1.25 - 1.32	否
ONTAP 双向 CHAP	1.25 - 1.32	否
动态导出策略	1.25 - 1.32	否
Trident 运算符	1.25 - 1.32	否
CSI 拓扑	1.25 - 1.32	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(AMD64和ARM64)支持的RedHat CorEOS (RHCOS)版本
- RHEL 8+(AMD64和ARM64)

  NVMe/TCP需要RHEL 9或更高版本。

- Ubuntu 22.04或更高版本(AMD64和ARM64)
- Windows Server 2022

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。

该 `tridentctl`实用程序还可以在这些Linux分发版中的任何一个上运行。

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。

Kubernetes版本	容器映像
v1.25.0、v1.26.0、v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0、v1.32.0	• dDocker。io/NetApp/trdent：24.10.0 • docer.io/NetApp/trdent-autostsupport：24.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v5.1.0 • 注册表.k8s.io/sig-storage/CsI-Attacher：v4.7.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.12.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v8.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.12.0 • dDocker .io/NetApp/trdent操作员：24.10.0 (可选)

安装 Trident

使用Trident操作员安装

使用tridentctl进行安装

使用Trident

准备工作节点

Kubernetes集群中的所有工作节点都必须能够挂载为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RedHat CoreTM OS已安装这些工具。

NFS工具

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

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"用于 `ontap-san`基于TCP (NVMe/TCP)协议的非易失性内存标准(NVMe)。

对于NVMe/TCP、建议使用ONTAP 9.12或更高版本。

基于**FC**的**SCSI**工具

基于光纤通道的**SCSI (FC)**是**Trident 24.10**版本中的一项技术预览功能。

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

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

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卷

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 * 。
- 如果在驱动程序和Element OS 12.5或更早版本中使用RHCOS 4.5或更高版本或其他与RHEL兼容的Linux分发版 `solidfire-san`、请确保在中将CHAP身份验证算法设置为MD5 `/etc/iscsi/iscsid.conf` 。Element 12.7可使用安全FIPS兼容CHAP算法SHA1、SHA-256和SHA3-256。

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

- 如果在iSCSI PV中使用运行RHE/RedHat CoreTM OS的工作节点、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 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. 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`。

4. 确保 `iscsid`和 `multipathd`正在运行：

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

5. 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 `open-iscsi`和 `multipath-tools`已启用且正在运行：

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

> 对于Ubuntu 18.04、您必须先使用发现目标端口 `iscsiadm`、然后 `open-iscsi`才能启动iSCSI守护进程。您也可以将此服务修改 `iscsi`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、请在Helm安装或Helm更新期间传递 `iscsiSelfHealingInterval`和 `iscsiSelfHealingWaitTime`参数。

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

```
helm install trident trident-operator-100.2410.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

Tridentctl

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

以下示例将iSCSI自我修复间隔设置为3分钟、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。

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RHE/RedHat Core-OS的工作节点与FC PV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
```

光纤通道(FC)支持

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。

基于光纤通道的**SCSI (FC)**是**Trident 24.10**版本中的一项技术预览功能。

光纤通道因其高性能、可靠性和可扩展性而成为企业存储环境中广泛采用的协议。它为存储设备提供了一个强大而高效的通信通道、可实现快速而安全的数据传输。通过使用基于光纤通道的SCSI、您可以利用现有基于SCSI的存储基础架构、同时享受光纤通道的高性能和远距离功能。它可以整合存储资源、创建可扩展的高效存储区域网络(SAN)、从而以低延迟处理大量数据。

将FC功能与Trident结合使用、您可以执行以下操作：

- 使用部署规范动态配置PVC。
- 创建卷快照并从此快照创建新卷。
- 克隆现有FC-PVC。
- 调整已部署卷的大小。

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()"

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。要为FC准备工作节点、必须安装所需的工具。

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RHE/RedHat Core-OS的工作节点与FC PV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
```

创建后端配置

为驱动程序和 `fcp`创建一个Trident后端 `ontap-san`作为sanType。

请参阅：

- "准备使用ONTAP SAN驱动程序配置后端"
- "ONTAP SAN配置选项和示例"

使用**FC**的后端配置示例

```
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
  sanType: fcp
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

创建存储类。

有关详细信息、请参见：

- "存储配置选项"

存储类示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fcp-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  storagePools: "ontap-san-backend:.*"
  fsType: "ext4"
allowVolumeExpansion: True
```

配置和管理后端

配置后端

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"
- "配置适用于 Google 云平台的 Cloud Volumes Service 后端"
- "配置 NetApp HCI 或 SolidFire 后端"
- "使用ONTAP或Cloud Volumes ONTAP NAS驱动程序配置后端"
- "使用ONTAP或Cloud Volumes ONTAP SAN驱动程序配置后端"
- "将Trident与Amazon FSx for NetApp ONTAP结合使用"

Azure NetApp Files

配置 **Azure NetApp Files** 后端

您可以将Azure NetApp Files配置为Trident的后端。您可以使用Azure NetApp Files后端连接NFS和SMB卷。Trident还支持使用托管身份对Azure Kubnetes Services (AKS)集群进行凭据管理。

Azure NetApp Files驱动程序详细信息

Trident提供了以下Azure NetApp Files存储驱动程序来与集群进行通信。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
`azure-netapp-files`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	`nfs`、 `smb`

注意事项

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

AKS的受管身份

Trident支持"受管身份"Azure Kubnetes服务集群。要利用受管身份提供的简化凭据管理、您必须：

- 使用AKS部署的Kubenetes集群

- 在AKS Kubbernetes集群上配置的受管身份
- 安装了Trident，其中包括要指定 "Azure"`的 `cloudProvider。

Trident 运算符

要使用Trident运算符安装Trident，请编辑 tridentorchestrator_cr.yaml`以将设置 `cloudProvider`为 `"Azure"。例如：

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

掌舵

以下示例使用环境变量将Trident集安装 cloudProvider`到Azure `$CP：

```
helm install trident trident-operator-100.2410.0.tgz --create
-namespace --namespace <trident-namespace> --set cloudProvider=$CP
```

<code></code>

以下示例将安装Trident并将标志设置 cloudProvider`为 `Azure：

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

适用于**AKS**的云身份

通过云身份、Kubnetes Pod可以通过作为工作负载身份进行身份验证来访问Azure资源、而不是提供明确的Azure凭据。

要在Azure中利用云身份、您必须：

- 使用AKS部署的Kubenetes集群
- 在AKS Kubelnetes集群上配置的工作负载身份和oidc-Issuer
- 已安装Trident、其中包括 `cloudProvider`用于指定 `"Azure"`和 `cloudIdentity`指定工作负载标识的

Trident 运算符

要使用Trident运算符安装Trident，请编辑 `tridentorchestrator_cr.yaml`以将设置为，并将 `cloudIdentity`设置 `cloudProvider`为 `"Azure" azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

例如：

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

掌舵

使用以下环境变量设置*云提供程序(CP)*和*云身份(CI)*标志的值：

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

以下示例将安装Trident并使用环境变量将设置 `cloudProvider`为`Azure `$CP`、并使用环境变量 `$CI`设置 `cloudIdentity`：

```
helm install trident trident-operator-100.2410.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$CI"
```

<code></code>

使用以下环境变量设置*云提供程序*和*云身份*标志的值：

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

以下示例将安装Trident并将标志设置 `cloud-provider`为 `$CP`、和 `cloud-identity` `$CI`：

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

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

NFS和SMB卷的前提条件

如果您是首次使用Azure NetApp Files 或在新位置使用、则需要进行一些初始配置来设置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、您需要满足以下条件：

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

- 一个容量池。请参阅 "Microsoft：为Azure NetApp Files 创建容量池"。
- 委派给Azure NetApp Files 的子网。请参阅 "Microsoft：将子网委派给Azure NetApp Files"。
- `subscriptionID`通过启用了Azure NetApp Files的Azure订阅。
- `tenantID` `clientID`和 `clientSecret` "应用程序注册"中具有足够权限的Azure NetApp Files服务。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/
read",

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

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

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

"Microsoft.Network/virtualNetworks/subnets/read",

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

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

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

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

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

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

- 至少包含一个 "委派子网"的Azure `location`。自Trident 22.01起、 `location`参数是后端配置文件顶层的必填字段。在虚拟池中指定的位置值将被忽略。
- 要使用 `Cloud Identity`，请从 "用户分配的托管身份"获取 `client ID`并在中指定该`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。请参阅 "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。要配置 `csi-proxy`，请参阅"GitHub：CSI代理"或了解在Windows上运行的Kubornetes"GitHub：适用于Windows的CSI代理"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可使用您的后端配置(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。

Trident不支持手动QoS容量池。

Azure NetApp Files后端提供了这些配置选项。

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	"Azure-netapp-files"
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + 随机字符
`subscriptionID`	在AKS集群上启用托管标识时、Azure订阅中的订阅ID为可选。	
`tenantID`	在AKS集群上使用托管身份或云身份时、应用程序注册中的租户ID为可选。	
`clientID`	在AKS集群上使用托管身份或云身份时、应用程序注册中的客户端ID为可选。	
`clientSecret`	在AKS集群上使用托管身份或云身份时、应用程序注册中的客户端密钥可选。	
`serviceLevel`	`Premium`或 `Ultra`之一 `Standard`	"" （随机）
`location`	要在其中创建新卷的Azure位置的名称在AKS集群上启用受管标识时为可选。	
`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）
`capacityPools`	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	""
`subnet`	委派给的子网的名称 `Microsoft.Netapp/volumes`	""

参数	说明	默认
networkFeatures	一个卷的一组vNet功能，可以是 Basic`或 `Standard。网络功能并非在所有地区都可用、可能需要在订阅中启用。如果指定 `networkFeatures`未启用此功能的时间、则会导致卷配置失败。	""
nfsMountOptions	精细控制 NFS 挂载选项。SMB卷已忽略。要使用NFS 4.1挂载卷、请在逗号分隔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。例如， \{"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。选项为 nfs、 `smb`或null。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"。	

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

所需权限和资源

如果在创建PVC时收到"No Capacity Pools"(未找到容量池)错误、则您的应用程序注册可能没有关联的所需权限和资源(子网、虚拟网络、容量池)。如果启用了调试、Trident将记录在创建后端时发现的Azure资源。验证是否正在使用适当的角色。

`netappAccounts` `capacityPools` `virtualNetwork`和 `subnet`的值 `resourceGroups`可以使用短名称或完全限定名称来指定。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。

`resourceGroups` `netappAccounts`和 `capacityPools`值是筛选器，用于将发现的资源集限制为此存储后端可用的资源集，并且可以任意组合方式指定。完全限定名称采用以下格式：

键入	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 >
子网	< 资源组 >/< 虚拟网络 >/< 子网 >

卷配置

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

参数	说明	默认
`exportRule`	新卷的导出规则。`exportRule`必须是IPv4地址或IPv4子网的任意组合的逗号分隔列表(采用CIDR表示法)。SMB卷已忽略。	"0.0.0.0/0
`snapshotDir`	控制 .snapshot 目录的可见性	对于NFSv4、为"TRUE"；对于NFSv3、为"false"
`size`	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

这是绝对的最低后端配置。使用此配置时、Trident会发现在所配置位置委派给Azure NetApp Files的所有NetApp帐户、容量池和子网、并随机将新卷放置在其中一个池和子网上。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

此后端配置会省略 `subscriptionID`、 `tenantID`、 `clientID`和 `clientSecret`，它们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: ["ultra-pool"]
  resourceGroups: ["aks-ami-eastus-rg"]
  netappAccounts: ["smb-na"]
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
```

适用于**AKS**的云身份

此后端配置会省略 `tenantID`、 `clientID`和 `clientSecret`，它们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

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

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

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: my-virtual-network
subnet: my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: 'true'
  size: 200Gi
  unixPermissions: '0777'
```

虚拟池配置

此后端配置可在一个文件中定义多个存储池。如果您有多个容量池支持不同的服务级别，并且您希望在 Kubernetes 中创建表示这些服务级别的存储类，则此功能非常有用。虚拟池标签用于根据区分池 `performance`。

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

支持的拓扑配置

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

存储类定义

以下 `StorageClass`定义涉及上述存储池。

使用字段的示例定义 `parameter.selector`

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

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

SMB卷的示例定义

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

默认命名空间上的基本配置

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

每个命名空间使用不同的密钥

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

每个卷使用不同的密钥

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

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

创建后端

创建后端配置文件后，运行以下命令：

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

如果后端创建失败，则后端配置出现问题。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。

Google Cloud NetApp卷

配置Google Cloud NetApp卷后端

现在、您可以将Google Cloud NetApp卷配置为Trident的后端。您可以使用Google Cloud NetApp卷后端连接NFS卷。

Google Cloud NetApp卷驱动程序详细信息

Trident提供了 `google-cloud-netapp-volumes`用于与集群通信的驱动程序。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
`google-cloud-netapp-volumes`	NFS	文件系统	Rwo、ROX、rwx、RWOP	`nfs`

适用于GKE的云身份

通过云身份、Kubnetes Pod可以通过作为工作负载身份进行身份验证来访问Google Cloud资源、而不是提供明确的Google Cloud凭据。

要在Google Cloud中利用云身份、您必须：

- 使用GKE部署的Kubbernetes集群。
- 在GKE集群上配置的工作负载标识以及在节点池上配置的GKE元数据服务器。
- 具有Google Cloud NetApp卷管理员(角色/GCP .admin)角色或自定义角色的NetApp服务帐户。
- 已安装Trident、其中包括用于指定"gcp"的云提供程序和用于指定新GCP服务帐户的云标识。下面给出了一个示例。

Trident 运算符

要使用Trident运算符安装Trident，请编辑 `tridentorchestrator_cr.yaml`以将设置为，并将 `cloudIdentity`设置 `cloudProvider`为 `"GCP" iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com。

例如：

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

掌舵

使用以下环境变量设置*云提供程序(CP)*和*云身份(CI)*标志的值：

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

以下示例将安装Trident并使用环境变量将设置 `cloudProvider`为`GCP `$CP`，并使用环境变量 `$ANNOTATION`设置 `cloudIdentity`：

```
helm install trident trident-operator-100.2406.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$ANNOTATION"
```

<code></code>

使用以下环境变量设置*云提供程序*和*云身份*标志的值：

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

以下示例将安装Trident并将标志设置 `cloud-provider`为 `$CP`、和 `cloud-identity` `$ANNOTATION`：

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

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卷的前提条件

如果您是首次使用Google Cloud NetApp卷或在新位置使用、则需要进行一些初始配置才能设置Google Cloud NetApp卷和创建NFS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

了解适用于Google Cloud NetApp卷的NFS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	说明	默认
`version`		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。	
`location`	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。	

参数	说明	默认
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）
serviceLevel	存储池及其卷的服务级别。值为 flex、、 standard premium`或 `extreme。	
network	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
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

您可以在配置文件的部分中控制默认卷配置 defaults。

参数	说明	默认
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	对目录的访问权限 .snapshot	对于NFSv4、为"TRUE"；对于NFSv3、为"false"
snapshotReserve	为快照预留的卷百分比	""(接受默认值0)
unixPermissions	新卷的UNIX权限(4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卷并尝试操作时、此配置是理想的选择、但实际上、您很可能需要为您配置的卷提供额外的范围界定。

```
---

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

```
    -----END PRIVATE KEY-----\n

---

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

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: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----

---

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
```

```
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '123455380079'
  location: europe-west6
  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: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
```

```
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----

---

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '123455380079'
  location: europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: '103346282737811234567'
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: '5'
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

适用于**GKE**的云身份

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

支持的拓扑配置

Trident可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。

更多示例

存储类定义示例

下面是有关上述后端的基本 `StorageClass` 定义。

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

使用字段的示例定义 **`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。

```
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。"。

PVC定义示例

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

要验证PVC是否已绑定、请运行以下命令：

```
kubectl get pvc gcnv-nfs-pvc

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

为Google Cloud后端配置Cloud Volumes Service

了解如何使用提供的示例配置将适用于Google Cloud的NetApp Cloud Volumes Service配置为Trident安装的后端。

Google Cloud驱动程序详细信息

Trident提供了 `gcp-cvs`用于与集群通信的驱动程序。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
`gcp-cvs`	NFS	文件系统	Rwo、ROX、rwx、RWOP	`nfs`

了解Trident对适用于Google Cloud的Cloud Volumes Service的支持

Trident可以使用"服务类型"以下两种方法之一创建Cloud Volumes Service卷：

- **CVS**性能：默认的Trident服务类型。这种性能优化的服务类型最适合重视性能的生产工作负载。CVS-Performance服务类型是一种硬件选项、支持的卷大小至少为100 GiB。您可以选择"三个服务级别"以下选项之一：
  - `standard`
  - `premium`
  - `extreme`
- * CVS*：CVS服务类型提供高区域可用性、性能级别限制为中等。CVS服务类型是一个软件选项、可使用存储池支持小至1 GiB的卷。存储池最多可包含50个卷、其中所有卷都共享池的容量和性能。您可以选择"两个服务级别"以下选项之一：
  - `standardsw`
  - `zoneredundantstandardsw`

您需要的内容

要配置和使用 "适用于 Google Cloud 的 Cloud Volumes Service"后端、您需要满足以下条件：

- 配置了NetApp Cloud Volumes Service 的Google Cloud帐户
- Google Cloud 帐户的项目编号
- 具有角色的Google Cloud服务帐户 `netappcloudvolumes.admin`

- Cloud Volumes Service 帐户的API密钥文件

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	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	"GCP-CVS"
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + API 密钥的一部分
`storageClass`	用于指定CVS服务类型的可选参数。 `software`用于选择CVS服务类型。否则，Trident会采用CVS性能服务类型(`hardware)。	
`storagePools`	仅限CVS服务类型。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。	
`hostProjectNumber`	如果使用共享VPC网络、则为必需项。在此方案中、`projectNumber`是服务项目、是主项目`hostProjectNumber`。	
`apiRegion`	Trident创建Cloud Volumes Service卷的Google Cloud区域。创建跨区域Kubnetes集群时、在中创建的卷`apiRegion`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 `netappcloudvolumes.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）。	
`proxyURL`	代理服务器需要连接到CVS帐户时的代理URL。代理服务器可以是 HTTP 代理，也可以是 HTTPS 代理。对于 HTTPS 代理，将跳过证书验证，以允许在代理服务器中使用自签名证书。不支持启用了身份验证的代理服务器。	
`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）
`serviceLevel`	新卷的CVS-Performance或CVS服务级别。CVS性能值为 `standard`、 `premium`或 `extreme`。CVS值为 `standardsw`或`zoneredundantstandardsw`。	CVS-Performance默认值为"standard"。CVS默认值为"standardsw"。
`network`	用于Cloud Volumes Service 卷的Google云网络。	default

参数	说明	默认
`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

`allowedTopologies`	要启用跨区域访问、的存储类定义`allowedTopologies`必须包括所有区域。例如：
	`- key: topology.kubernetes.io/region`
	`values:`
	`- us-east1`
	`- europe-west1`	

卷配置选项

您可以在配置文件的部分中控制默认卷配置 `defaults`。

参数	说明	默认
`exportRule`	新卷的导出规则。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。	"0.0.0.0/0
`snapshotDir`	对目录的访问权限 `.snapshot`	"错误"
`snapshotReserve`	为快照预留的卷百分比	"" （接受 CVS 默认值为 0 ）
`size`	新卷的大小。CVS性能最小值为100 GiB。CVS最小值为1 GiB。	CVS-Performance服务类型默认为"100GiB"。CVS服务类型未设置默认值、但至少需要1 GiB。

CVS-Performance服务类型示例

以下示例提供了CVS-Performance服务类型的示例配置。

示例 **1**： 最低配置

这是使用默认CVS-Performance服务类型以及默认"标准"服务级别的最小后端配置。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```

示例**2**：服务级别配置

此示例说明了后端配置选项、包括服务级别和卷默认值。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

示例**3**：虚拟池配置

此示例使用 `storage`配置虚拟池以及引用这些虚拟池的 `StorageClasses`。请参见[存储类定义]以了解存储类的定义方式。

此处为所有虚拟池设置了特定默认值、将设置为5%、将设置 `snapshotReserve`为 `exportRule`0.0.0.0/0。虚拟池在一节中进行了定义 `storage`。每个虚拟池都定义自己的 `serviceLevel`，而某些池会覆盖默认值。虚拟池标签用于根据和 `protection`区分池 `performance`。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
    performance: extreme
    protection: extra
  serviceLevel: extreme
  defaults:
```

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

存储类定义

以下StorageClass定义适用于虚拟池配置示例。使用 `parameters.selector`，您可以为每个StorageClass指定用于托管卷的虚拟池。卷将在选定池中定义各个方面。

存储类示例

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

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

- 第一个StorageClass(`cvs-extreme-extra-protection`)映射到第一个虚拟池。这是唯一一个可提供极高性能且 Snapshot 预留为 10% 的池。
- 最后一个StorageClass(`cvs-extra-protection`)调用提供10%快照预留的任何存储池。Trident决定选择哪个虚拟池、并确保满足快照预留要求。

CVS服务类型示例

以下示例提供了CVS服务类型的示例配置。

示例**1**：最低配置

这是用于指定CVS服务类型和默认 `standardsw`服务级别的最低后端配置 `storageClass`。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
serviceLevel: standardsw
```

示例**2**：存储池配置

此示例后端配置使用 `storagePools`配置存储池。

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

下一步是什么?

创建后端配置文件后，运行以下命令：

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

如果后端创建失败，则后端配置出现问题。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。

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。请参阅 "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	说明	默认
version		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	SolidFire + 存储（ iSCSI ） IP 地址
Endpoint	使用租户凭据的 SolidFire 集群的 MVIP	
SVIP	存储（ iSCSI ） IP 地址和端口	

参数	说明	默认
`labels`	要应用于卷的一组任意 JSON 格式的标签。	"
`TenantName`	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
`AccessGroups`	要使用的访问组 ID 列表	查找名为 "trident " 的访问组的 ID
`Types`	QoS 规范	
`limitVolumeSize`	如果请求的卷大小超过此值，则配置失败	"（默认情况下不强制实施）
`debugTraceFlags`	故障排除时要使用的调试标志。示例 ｛"api"： false ， "method "： true ｝	空

除非正在进行故障排除并需要详细的日志转储、否则请勿使用 `debugTraceFlags`。

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

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storage class参数定义要使用其中每个存储类的存储类 `IOPS`。

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

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

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、并将设置 `type`为银牌。虚拟池在一节中进行了定义 `storage`。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。通过 `parameters.selector`字段、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 (Trident 24.10中的技术预览)	块	Rwo、ROX、rwx、RWOP	无文件系统；原始块设备
`ontap-san`	基于FC的iSCSI SCSI (Trident 24.10中的技术预览)	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 请参阅NVMe/TCP的其他注意事项。	块	Rwo、ROX、rwx、RWOP	无文件系统；原始块设备
`ontap-san`	NVMe/TCP 请参阅NVMe/TCP的其他注意事项。	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

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-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`

- 只有当永久性卷使用量计数预期大于时才使用 `ontap-san-economy`"支持的ONTAP卷限制"。
- `ontap-nas-economy`仅当永久性卷使用量计数预计高于且 `ontap-san-economy`无法使用驱动程序时才"支持的ONTAP卷限制"使用。
- 如果您预计需要数据保护、灾难恢复或移动性、请勿使用 `ontap-nas-economy`。

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。

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：

- DH-HMAC-CHAP、由NVMe本机提供支持
- 设备映射程序(Device maper、DM)多路径
- 进行了加密

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要求至少为SVM分配一个聚合。

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，可以配置 `san-dev`使用驱动程序的类和使用 `ontap-san-economy`驱动程序的 `san-default`类 `ontap-san`。

所有Kubernetes工作节点都必须安装适当的iSCSI工具。有关详细信息、请参见 "准备工作节点" 。

对**ONTAP**后端进行身份验证

Trident提供了两种对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。建议使用预定义的安全登录角色、例如`admin`或、 `vsadmin`以确保与ONTAP版本的最大兼容性。
- 基于证书：Trident还可以使用后端安装的证书与ONTAP集群进行通信。此处，后端定义必须包含客户端证书，密钥和可信 CA 证书的 Base64 编码值（如果使用）（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。

> 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

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

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
```

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。然后使用包含所需执行参数的更新后端.json文件 `tridentctl backend update`。

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

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

轮换密码时，存储管理员必须先在 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** 对连接进行身份验证

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
```

`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 。

除了之外， `chapInitiatorSecret` `chapTargetUsername`后端定义中还 `useCHAP=true`必须包括、、 `chapTargetInitiatorSecret`和 `chapUsername`字段。在创建后端后，可以通过运行来更改这些密钥 `tridentctl update`。

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。

轮换凭据并更新后端

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

更新后端的CHAP密码时、必须使用 `tridentctl`更新后端。请勿通过CLI/RAID ONTAP UI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

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

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
`version`		始终为 1

参数	说明	默认
storageDriverName	存储驱动程序的名称	ontap-nas、、 ontap-nas-economy ontap-nas-flexgroup、 ontap-san ontap-san-economy
backendName	自定义名称或存储后端	驱动程序名称+"_"+ dataLIF
managementLIF	集群或SVM管理LIF的IP地址。可以指定完全限定域名(FQDN)。如果Trident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[mcc-best]。	"10.0.0.1 "， "2001： 1234：abcd：：： 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iCI LUN。如果明确定义、则会生成警告 dataLIF。*省略MetroCluster。*请参见[mcc-best]。	由SVM派生
svm	要使用的Storage Virtual Machine *省略for MetroCluster。*请参见[mcc-best]。	如果指定了SVM、则派生此参数 managementLIF
useCHAP	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" 。	false
chapInitiatorSecret	CHAP 启动程序密钥。如果需要、则为必需项 useCHAP=true	""
labels	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。如果需要、则为必需项 useCHAP=true	""
chapUsername	入站用户名。如果需要、则为必需项 useCHAP=true	""
chapTargetUsername	目标用户名。如果需要、则为必需项 useCHAP=true	""
clientCertificate	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""
trustedCACertificate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""

参数	说明	默认
username	与ONTAP 集群通信所需的用户名。用于基于凭据的身份验证。	""
password	与ONTAP 集群通信所需的密码。用于基于凭据的身份验证。	""
svm	要使用的 Storage Virtual Machine	如果指定了SVM、则派生此参数 managementLIF
storagePrefix	在 SVM 中配置新卷时使用的前缀。无法稍后修改。要更新此参数、您需要创建一个新的后端。	trident
aggregate	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。如果未分配、则 可以使用任何可用聚合来配置FlexGroup卷。 在SVM中更新聚合后、该聚合将在Trident中自动更新、方法是轮询SVM、而无需重新启动Trident控制器。在Trident中配置了特定聚合以配置卷后、如果将该聚合重命名或移出SVM、则在轮询SVM聚合时、后端将在Trident中变为故障状态。您必须将聚合更改为SVM上的聚合、或者将其全部删除、以使后端恢复联机。	""
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。	"" （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）
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、除非您正在进行故障排除并需要详细的日志转储、否则不会使用 {"api"：false、"METHO"：true} 。	null

参数	说明	默认
useREST	用于使用 ONTAP REST API 的布尔参数。useREST`设置为时 `true，Trident使用ONTAP REST API与后端通信；设置为时 false，Trident使用ONTAP ZAPI调用与后端通信。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 。从Trident 24.06版和ZAPI.151或更高版本开始、默认情况下会设置为 true；更useREST`改为 `false`以使用ONTAP 9 `useREST ONTAP调用。useREST 完全符合NVMe/TCP要求。	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。**"FCP"(基于FC的SCSI)是Trident 24.10**版本中的一项技术预览功能。	`iscsi`如果为空
formatOptions	`formatOptions`用于指定命令的命令行参数、每当对卷进行格式化时、都会应用这些参数`mkfs`。这样、您可以根据偏好格式化卷。请确保指定与mkfs命令选项类似的格式选项，但不包括设备路径。示例："-E nobdiscard" • ontap-san `ontap-san-economy`仅支持和驱动程序。*	
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）
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建议使用以下选项来加快格式化过程：

-E NODiscard：

- 保留、不要尝试在mkfs时间丢弃块(丢弃块最初在固态设备和稀疏/精简配置存储上很有用)。此选项将取代已弃用的选项"-K"、并适用于所有文件系统(xfs、ext3和ext4)。

用于配置卷的后端配置选项

您可以在配置部分使用这些选项控制默认配置 defaults。有关示例，请参见以下配置示例。

参数	说明	默认
`spaceAllocation`	LUN 的空间分配	"正确"
`spaceReserve`	空间预留模式；"无"(精简)或"卷"(厚)	"无"
`snapshotPolicy`	要使用的 Snapshot 策略	"无"
`qosPolicy`	要为创建的卷分配的 QoS 策略组。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	""
`snapshotReserve`	为快照预留的卷百分比	如果为"none"、则为"0" `snapshotPolicy`、否则为""
`splitOnClone`	创建克隆时，从其父级拆分该克隆	"错误"
`encryption`	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"。	"错误"
`luksEncryption`	启用LUKS加密。请参阅 "使用Linux统一密钥设置(LUKS)"。NVMe/TCP不支持使用此类数据加密。	""
`securityStyle`	新卷的安全模式	`unix`
`tieringPolicy`	使用"无"的层策略	对于ONTAP 9.5 SVM-DR之前的配置、为"仅快照"
`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'
```

对于使用驱动程序创建的所有卷 `ontap-san`、Trident会向FlexVol额外添加10%的容量、以容纳LUN元数据。LUN 将使用用户在 PVC 中请求的确切大小进行配置。Trident会将10%的空间添加到FlexVol中(在ONTAP中显示为可用大小)。用户现在将获得所请求的可用容量。此更改还可防止 LUN 变为只读状态，除非已充分利用可用空间。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元数据而向FlexVol额外增加的10%。对于 `snapshotReserve`= 5%、PVC请求= 5 GiB、则卷总大小为5.79 GiB、可用大小为5.5 GiB。此 `volume show`命令应显示类似于以下示例的结果：

目前，调整大小是对现有卷使用新计算的唯一方法。

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

> 如果您将Amazon FSx on NetApp ONTAP与结合使用、建议您为Trident指定DNS名称、而不是IP地址。

ONTAP SAN示例

这是使用驱动程序的基本配置 `ontap-san`。

```
---
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>
```

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>
```

1. 示例

您可以配置后端，以避免在期间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`参数。例如：

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

基于证书的身份验证示例

在此基本配置示例中 `clientCertificate`， `clientPrivateKey`、和 `trustedCACertificate`(如果使用受信任CA，则为可选)分别填充 `backend.json`并采用base64编码的客户端证书值、专用密钥值和受信任CA证书值。

```
---
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

以下示例将创建一个后端，并 `useCHAP`将设置为 `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>
```

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
```

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}}_{{.volume.R
equestName}}"
},
"labels": {"cluster": "ClusterA", "PVC":
"{{.volume.Namespace}}_{{.volume.RequestName}}"}
```

formatOptions <code> ONTAP－san－</code>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、会为所有存储池设置特定默认值、例如 `spaceReserve`、在none、`spaceAllocation`在false和 `encryption`在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 上设置。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、某些存储池会设置自己的、 `spaceAllocation`和 `encryption`值、而某些存储 `spaceReserve`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定义参见[虚拟池后端示例]。通过 `parameters.selector`字段、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。

- `protection-gold`StorageClass将映射到后端的第一个虚拟池 `ontap-san`。这是唯一提供金牌保护的池。

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

- `protection-not-gold`StorageClass将映射到后端的第二个和第三个虚拟池 `ontap-san。只有这些池提供的保护级别不是gold。

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

- `app-mysqldb`StorageClass将映射到后端的第三个虚拟池 `ontap-san-economy。这是为mysqldb类型的应用程序提供存储池配置的唯一池。

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

- `protection-silver-creditpoints-20k`StorageClass将映射到后端的第二个虚拟池 `ontap-san。这是唯一提供银牌保护和20000个信用点的池。

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

- `creditpoints-5k`StorageClass将映射到后端的第三个虚拟池 `ontap-san`和后端的第四个虚拟池 `ontap-san-economy`。这是唯一一款信用点数为5000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"
```

- `my-test-app-sc`StorageClass将映射到 `testAPP`驱动程序 `sanType: nvme`中的虚拟池 `ontap-san`。这是唯一的池选项 testApp。

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

Trident将决定选择哪个虚拟池、并确保满足存储要求。

ONTAP NAS驱动程序

ONTAP NAS驱动程序概述

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

ONTAP NAS驱动程序详细信息

Trident提供了以下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`

- 只有当永久性卷使用量计数预期大于时才使用 `ontap-san-economy`"支持的ONTAP卷限制"。
- `ontap-nas-economy`仅当永久性卷使用量计数预计高于且 `ontap-san-economy`无法使用驱动程序时才"支持的ONTAP卷限制"使用。
- 如果您预计需要数据保护、灾难恢复或移动性、请勿使用 `ontap-nas-economy`。

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。

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。

要求

- 对于所有ONTAP后端、Trident要求至少为SVM分配一个聚合。
- 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驱动程序的Gold类和一个使用驱动程序的Bronze `ontap-nas-economy`类 `ontap-nas`。
- 所有Kubernetes工作节点都必须安装适当的NFS工具。"此处"有关详细信息、请参见。

- 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。建议使用与预定义的安全登录角色关联的帐户、例如`admin`或、 `vsadmin`以确保与ONTAP版本最大程度地兼容。
- 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。

> 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

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

JSON

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "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 / 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。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。然后使用包含所需执行参数的更新后端.json文件 `tridentctl update backend`。

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

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

> 轮换密码时，存储管理员必须先在 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。

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。

在Trident 24.10中、 `ontap-nas`存储驱动程序将继续与早期版本一样工作；ONTAP NAS驱动程序未进行任何更改。在Trident 24.10中、只有 `ontap-nas-economy`存储驱动程序具有基于卷的粒度访问控制。

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
```

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。

您可以在创建后端后为后端更新 `autoExportPolicy`和 `autoExportCIDRs`。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 `autoExportPolicy`、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。

> 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、即可使用驱动程序配置SMB卷 `ontap-nas`。

> 您必须在SVM上同时配置NFS和SMB/CCIFS协议、才能为内部ONTAP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。要配置 `csi-proxy`，请参阅"GitHub：CSI代理"或了解在Windows上运行的Kubornetes"GitHub：适用于Windows的CSI代理"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。

   您可以通过以下两种方式之一创建SMB管理员共享：使用"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
```

      有关完整详细信息、请参见"创建 SMB 共享"。

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

参数	说明	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。对于**SMB**卷，必须设置为 **ntfs**` 或 `**mixed**。	`ntfs`或 `mixed`SMB卷
unixPermissions	新卷的模式。对于SMB卷、必须留空。	""

ONTAP NAS配置选项和示例

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

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
version		始终为 1
storageDriverName	存储驱动程序的名称	"ONTAP NAS "、"ONTAP NAS经济"、"ONTAP NAS灵活组"、"ONTAP SAN "、"ONTAP SAN经济"
backendName	自定义名称或存储后端	驱动程序名称+"_"+ dataLIF
managementLIF	集群或SVM管理LIF的IP地址可以指定完全限定域名(FQDN)。如果Trident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[mcc-best]。	"10.0.0.1 "， "2001： 1234：abcd：：： fefe] "
dataLIF	协议 LIF 的 IP 地址。建议指定 dataLIF。如果不提供此参数、则Trident将从SVM提取数据LUN。您可以指定用于NFS挂载操作的完全限定域名(FQDN)、从而可以创建循环DNS、以便在多个数据LIF之间实现负载平衡。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[mcc-best]。	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 *省略for MetroCluster。*请参见[mcc-best]。	如果指定了SVM、则派生此参数 managementLIF

参数	说明	默认
`autoExportPolicy`	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
`autoExportCIDRs`	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"]`
`labels`	要应用于卷的一组任意 JSON 格式的标签	""
`clientCertificate`	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
`clientPrivateKey`	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""
`trustedCACertificate`	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证	""
`username`	用于连接到集群 /SVM 的用户名。用于基于凭据的身份验证	
`password`	连接到集群 /SVM 的密码。用于基于凭据的身份验证	
`storagePrefix`	在 SVM 中配置新卷时使用的前缀。设置后无法更新 如果使用的ONTAP是包含24个或更多字符的storagePrefix、则qtrees不会嵌入存储前缀、但会显示在卷名称中。	"三级联"
`aggregate`	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。如果未分配、则 可以使用任何可用聚合来配置FlexGroup卷。 在SVM中更新聚合后、该聚合将在Trident中自动更新、方法是轮询SVM、而无需重新启动Trident控制器。在Trident中配置了特定聚合以配置卷后、如果将该聚合重命名或移出SVM、则在轮询SVM聚合时、后端将在Trident中变为故障状态。您必须将聚合更改为SVM上的聚合、或者将其全部删除、以使后端恢复联机。	""
`limitAggregateUsage`	如果使用量超过此百分比，则配置失败。* 不适用于适用于 ONTAP 的 Amazon FSx *	"" （默认情况下不强制实施）

参数	说明	默认
FlexgroupGroup GroupRegateList	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。 在SVM中更新聚合列表后、此列表将在Trident中自动更新、方法是轮询SVM、而无需重新启动Trident控制器。在Trident中配置特定聚合列表以配置卷后、如果聚合列表重命名或移出SVM、则在轮询SVM聚合时、后端将在Trident中变为故障状态。您必须将聚合列表更改为SVM上的聚合列表、或者将其全部删除、以使后端恢复联机。	""
`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管理的卷的大小上限、并且此`qtreesPerFlexvol`选项允许自定义每个FlexVol的qtrees的最大数量。	"" （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、除非您正在进行故障排除并需要详细的日志转储、否则不会使用 {"api"：false、"METHO"：true} `debugTraceFlags`。	空
`nasType`	配置NFS或SMB卷创建。选项为 `nfs`、 `smb`或null。默认情况下、将设置为空会将NFS卷设置为空。	`nfs`
`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。	""
`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`

参数	说明	默认
useREST	用于使用 ONTAP REST API 的布尔参数。 useREST` 设置为时 `true，Trident使用ONTAP REST API与后端通信；设置为时 false，Trident使用ONTAP ZAPI调用与后端通信。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 。从Trident 24.06版和ZAPI.151或更高版本开始、默认情况下会设置为 true；更 useREST`改为 `false`以使用ONTAP 9 `useREST ONTAP调用。	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.	

用于配置卷的后端配置选项

您可以在配置部分使用这些选项控制默认配置 defaults。有关示例，请参见以下配置示例。

参数	说明	默认
spaceAllocation	qtrees的空间分配	"正确"
spaceReserve	空间预留模式；"无"(精简)或"卷"(厚)	"无"
snapshotPolicy	要使用的 Snapshot 策略	"无"
qosPolicy	要为创建的卷分配的 QoS 策略组。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.	""
snapshotReserve	为快照预留的卷百分比	如果为"none"、则为"0" snapshotPolicy、否则为""
splitOnClone	创建克隆时，从其父级拆分该克隆	"错误"
encryption	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"。	"错误"
tieringPolicy	使用"无"的层策略	对于ONTAP 9.5 SVM-DR之前的配置、为"仅快照"
unixPermissions	新卷的模式	"777"表示NFS卷；空(不适用)表示SMB卷

参数	说明	默认
`snapshotDir`	控制对目录的访问 `.snapshot`	对于NFSv4、为"TRUE"；对于NFSv3、为"false"
`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。	""

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'
```

对于 `ontap-nas`和 `ontap-nas-flexgroups`，Trident现在使用新的计算方法来确保使用snapshotReserve百分比和pvc正确调整FlexVol的大小。当用户请求PVC时、Trident会使用新计算方法创建具有更多空间的原始FlexVol。此计算可确保用户在 PVC 中收到所请求的可写空间，而不是小于所请求的空间。在

v21.07 之前，如果用户请求 PVC （例如， 5GiB ），并且 snapshotReserve 为 50% ，则只会获得 2.5 GiB 的可写空间。这是因为用户请求的是整个卷、并且 `snapshotReserve`是其中的一个百分比。在`Trident 21.07`中、用户请求的是可写空间、`Trident`将该数字定义 `snapshotReserve`为整个卷的百分比。这不适用于 `ontap-nas-economy`。请参见以下示例以了解其工作原理：

计算方法如下：

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

对于 snapshotReserve = 50% ， PVC 请求 = 5GiB ，卷总大小为 2/.5 = 10GiB ，可用大小为 5GiB ，这是用户在 PVC 请求中请求的大小。此 `volume show`命令应显示类似于以下示例的结果：

先前安装的现有后端将在升级Trident时按照上文所述配置卷。对于在升级之前创建的卷，您应调整其卷的大小，以便观察到所做的更改。例如、使用较早版本的2GiB PVC `snapshotReserve=50`会导致卷提供1GiB的可写空间。例如，将卷大小调整为 3GiB 可为应用程序在一个 6 GiB 卷上提供 3GiB 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。

> 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

您可以配置后端，以避免在期间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`参数。例如：

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

SMB卷示例

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

基于证书的身份验证示例

这是一个最小的后端配置示例。 clientCertificate clientPrivateKey`和 `trustedCACertificate(如果使用受信任CA，则为可选)将分别填充 `backend.json`并采用base64编码的客户端证书值、私钥值和受信任CA证书值。

```
---
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
```

IPv6地址示例

此示例显示了 `managementLIF`如何使用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`对于使用SMB卷的FSx for ONTAP、参数是必需的。

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

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}}_{{.volume.R
equestName}}"
},
"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`、在none、`spaceAllocation`在false和 `encryption`在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、某些存储池会设置自己的、 `spaceAllocation`和 `encryption`值、而某些存储 `spaceReserve`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定义请参见[虚拟池后端示例]。通过 `parameters.selector`字段、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。

- `protection-gold`StorageClass将映射到后端的第一个和第二个虚拟池 `ontap-nas-flexgroup`。这些池是唯一提供金牌保护的池。

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

- `protection-not-gold`StorageClass将映射到后端的第三个和第四个虚拟池 `ontap-nas-flexgroup`。这些池是唯一提供黄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"
```

- `app-mysqldb`StorageClass将映射到后端的第四个虚拟池 `ontap-nas`。这是为mysqldb类型的应用程序提供存储池配置的唯一池。

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

- protection-silver-creditpoints-20k`StorageClass将映射到后端的第三个虚拟池 `ontap-nas-flexgroup。这是唯一提供银牌保护和20000个信用点的池。

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

- creditpoints-5k`StorageClass将映射到后端的第三个虚拟池 `ontap-nas`和后端的第二个虚拟池 `ontap-nas-economy。这是唯一一款信用点数为5000的池产品。

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

Trident将决定选择哪个虚拟池、并确保满足存储要求。

在初始配置后更新 dataLIF

您可以在初始配置后更改数据LIF、方法是运行以下命令、为新的后端JSON文件提供更新的数据LIF。

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

如果PVC连接到一个或多个Pod、则必须关闭所有对应Pod、然后将其恢复到、新数据LIF才能生效。

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、支持客户启动和运行由NetApp ONTAP存储操作系统提供支持的文件系统。借助适用于ONTAP 的FSx、您可以利用您熟悉的NetApp功能、性能和管理功能、同时利用在AWS上存储数据的简便性、灵活性、安全性和可扩展性。FSX for ONTAP 支持ONTAP 文件系统功能和管理API。

您可以将Amazon FSx for NetApp ONTAP文件系统与Trident进行集成、以确保在Amazon Elic Kubelnetes Service (EKS)中运行的Kubelnetes集群可以配置ONTAP支持的块和文件永久性卷。

文件系统是 Amazon FSX 中的主要资源，类似于内部部署的 ONTAP 集群。在每个 SVM 中，您可以创建一个或多个卷，这些卷是将文件和文件夹存储在文件系统中的数据容器。借助适用于 NetApp ONTAP 的 Amazon FSX ， Data ONTAP 将作为云中的托管文件系统提供。新的文件系统类型称为 * NetApp ONTAP * 。

通过将Trident与Amazon FSx for NetApp ONTAP结合使用、您可以确保在Amazon Elic Kubelnetes Service (EKS)中运行的Kubelnetes集群可以配置ONTAP支持的块和文件永久性卷。

要求

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

- 已安装的现有Amazon EKS集群或自行管理的Kubornetes集群 `kubectl`。
- 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)。
- 准备用于的工作节点"NFS或iSCSI"。

根据您的EKS AMI类型、确保按照Amazon Linux和Ubuntu (AMI)所需的节点准备步骤进行操作 "Amazon Machine 映像"。

注意事项

- SMB卷：
  - 仅使用驱动程序支持SMB卷 `ontap-nas`。
  - Trident EKS加载项不支持SMB卷。
  - 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" 。
- 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`。

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"。

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。

> 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Trident 24.10的测试。

AMI	NAS	NAS经济型	SAN	SAN经济型
AL2023_x86_64_STANDARD	是	是	是	是
AL2_x86_64	是	是	是**	是**
BOTTLEROCKET_x86_64	是 *	是	不适用	不适用
AL2023_ARM_64_STANDARD	是	是	是	是
AL2_ARM_64	是	是	是**	是**
BOTTLEROCKET_ARM_64	是 *	是	不适用	不适用

- *必须在挂载选项中使用"nolock"。
- **在不重新启动节点的情况下，无法删除PV

> 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。此列表可作为已知有效的AMI的指南。

使用以下项执行的测试：

- EKS版本：1.30
- 安装方法：Helm和作为AWS插件
- 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。

要使用AWS IAM角色进行身份验证、您必须使用EKS部署Kubenetes集群。

创建AWS机密管理器密钥

以下示例将创建一个AWS机密管理器密钥、用于存储Trident CSI凭据：

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

创建IAM策略

以下示例将使用AWS命令行界面创建IAM策略：

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

策略JSON文件：

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

为服务帐户创建**IAM**角色

AWS命令行界面

```
aws iam create-role --role-name trident-controller \
  --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`：

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

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

将**IAM**角色附加到**IAM**策略：

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**OCD**提供程序是否关联：

验证OIDC提供程序是否已与集群关联。您可以使用以下命令进行验证：

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

使用以下命令将IAM OIDC与集群关联：

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

eksc

以下示例将在EKS中为服务帐户创建IAM角色：

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

安装 **Trident**

Trident简化了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。

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**

1. 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GitHub上的"Assets"部分下载并提取最新版本的Trident安装程序。

   ```
   wget https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
   installer-24.10.0.tar.gz
   tar -xf trident-installer-24.10.0.tar.gz
   cd trident-installer/helm
   ```

2. 使用以下环境变量设置*云提供程序*和*云身份*标志的值：

以下示例将安装Trident并将标志设置 `cloud-provider`为 `$CP`、和 `cloud-identity` `$CI`：

```
helm install trident trident-operator-100.2410.0.tgz --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-100.2410.0    24.10.0
```

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 ARM 2 (AL2_x86_64)或Amazon Linux 2 ARM (AL2_AMAZON_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

eksc

以下示例命令用于安装Trident EKS加载项：

```
eksctl create addon --name netapp_trident-operator --cluster
<cluster_name> \
    --service-account-role-arn
arn:aws:iam::<account_id>:role/<role_name> --force
```

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，单击*群集*。
3. 单击要为其配置NetApp Trident CSI加载项的集群的名称。
4. 单击*Add-ones*，然后单击*Get more add-ones*。
5. 在*Select add-ons*页上，执行以下操作：
   a. 在AWS Marketplace EKS-addons部分中、选中* Trident by NetApp *复选框。
   b. 单击 * 下一步 * 。
6. 在“*配置选定的附加项*设置”页面上，执行以下操作：
   a. 选择要使用的*版本*。
   b. 对于*Select IAM Role*，保留为*not set*。
   c. 展开*可选配置设置*，遵循*附加配置架构*，并将*配置值*部分中的configurationvalues*参数设置为您在上一步中创建的roole-arn (值格式应为： `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`)。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：

```
add-on.json
{

   "clusterName": "<eks-cluster>",
   "addonName": "netapp_trident-operator",
   "addonVersion": "v24.10.0-eksbuild.1",
   "serviceAccountRoleArn": "<arn:aws:iam::123456:role/astratrident-
role>",
   "configurationValues": "{"cloudIdentity":
"'eks.amazonaws.com/role-arn:
<arn:aws:iam::123456:role/astratrident-role>'",
   "cloudProvider": "AWS"}"
}
```

2. 安装Trident EKS附加软件"

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

更新Trident EKS加载项

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    v24.10.0-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 v24.10.0-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"。

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，单击*群集*。
3. 单击要更新的NetApp Trident CSI加载项的集群的名称。
4. 单击*Add-ones*选项卡。
5. 单击Trident by NetApp，然后单击*Edit*。
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. 单击 * 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 my-cluster netapp_trident-operator
vpc-cni --addon-version v24.6.1-eksbuild.1 \
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name
--configuration-values '{}' --resolve-conflicts --preserve
```

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。
- 从您的集群中完全删除附加软件–我们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。

如果此附加项具有关联的IAM帐户、则不会删除此IAM帐户。

eksc

以下命令将卸载Trident EKS加载项：

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

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，单击*群集*。
3. 单击要删除的NetApp Trident CSI加载项的集群的名称。
4. 单击*Add-ons*选项卡，然后单击Trident by NetApp．*
5. 单击 * 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. 单击 * 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
```

配置存储后端

ONTAP SAN和NAS驱动程序集成

要创建存储后端、您需要创建JSON或YAML格式的配置文件。该文件需要指定所需的存储类型(NAS或SAN)、文件系统和用于获取该文件的SVM以及如何向其进行身份验证。以下示例显示了如何定义基于NAS的存储以及如何使用AWS密钥将凭据存储到要使用的SVM：

YAML

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

JSON

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

运行以下命令以创建和验证Trident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
```

FSx for ONTAP驱动程序详细信息

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

- `ontap-san`：配置的每个PV都是其自身Amazon FSx for NetApp ONTAP卷中的一个LUN。建议用于块存储。
- `ontap-nas`：配置的每个PV都是一个完整的Amazon FSx for NetApp ONTAP卷。建议用于NFS和SMB。
- `ontap-san-economy`：配置的每个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卷具有一个可配置数量的qtree。
- `ontap-nas-flexgroup`：配置的每个PV都是一个完整的Amazon FSx for NetApp ONTAP FlexGroup卷。

有关驱动程序的详细信息，请参阅"NAS驱动程序"和"SAN驱动程序"。

创建配置文件后、运行此命令在EKS中创建该文件：

```
kubectl create -f configuration_file
```

要验证状态、请运行以下命令：

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

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

后端高级配置和示例

有关后端配置选项，请参见下表：

参数	说明	示例
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`ontap-nas`、、 `ontap-nas-economy` `ontap-nas-flexgroup`、 `ontap-san` `ontap-san-economy`
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + dataLIF
`managementLIF`	集群或SVM管理LIF的IP地址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：abcd：：： fefe] "

参数	说明	示例
dataLIF	协议 LIF 的 IP 地址。* ONTAP NAS驱动程序*：建议指定dataLIF。如果不提供此参数、则Trident将从SVM提取数据LUN。您可以指定用于NFS挂载操作的完全限定域名(FQDN)、从而可以创建循环DNS、以便在多个数据LIF之间实现负载平衡。可以在初始设置后更改。请参阅 。* ONTAP SAN驱动程序*：不为iSCSI指定。Trident使用ONTAP选择性LUN映射来发现建立多路径会话所需的iSCI LI。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地址必须用方括号定义、例如：[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]。	
autoExportPolicy	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
autoExportCIDRs	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 "]"
labels	要应用于卷的一组任意 JSON 格式的标签	""
clientCertificate	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""
trustedCACertificate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""
username	用于连接到集群或SVM的用户名。用于基于凭据的身份验证。例如、vsadmin。	
password	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。
storagePrefix	在 SVM 中配置新卷时使用的前缀。创建后无法修改。要更新此参数、您需要创建一个新的后端。	trident

参数	说明	示例
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 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、除非您正在进行故障排除并需要详细的日志转储、否则不会使用 {"ap1"：false、"METHOU"：true } debugTraceFlags。	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null。*对于SMB卷，必须设置为 `smb。*默认情况下、将设置为空会将NFS卷设置为空。	nfs
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、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

参数	说明	示例
useREST	用于使用 ONTAP REST API 的布尔参数。技术预览 useREST 以技术预览的形式提供，建议用于测试环境，而不用于生产工作负载。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密钥。	"" "" ""
credentials	指定要存储在AWS机密管理器中的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

您可以在配置部分使用这些选项控制默认配置 defaults。有关示例，请参见以下配置示例。

参数	说明	默认
spaceAllocation	LUN 的空间分配	true
spaceReserve	空间预留模式；" 无 "（精简）或 " 卷 "（厚）	none
snapshotPolicy	要使用的 Snapshot 策略	none
qosPolicy	要为创建的卷分配的 QoS 策略组。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	说明	默认
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.	"
snapshotReserve	为快照预留的卷百分比为 "0"	如果 snapshotPolicy`为 `none、 `else`""
splitOnClone	创建克隆时，从其父级拆分该克隆	false
encryption	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。请参阅 "使用Linux统一密钥设置(LUKS)"。仅SAN。	""
tieringPolicy	要使用的层策略 none	`snapshot-only`对于ONTAP 9 5之前的SVM-DR配置
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。

准备配置**SMB**卷

您可以使用驱动程序配置SMB卷 ontap-nas。完成以下步骤之前。ONTAP SAN和NAS驱动程序集成

开始之前

在使用驱动程序配置SMB卷之前、 `ontap-nas`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 2019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。要配置 csi-proxy，请参阅"GitHub：CSI代理"或了解在Windows上运行的Kubornetes"GitHub：适用于Windows的CSI代理"节点。

步骤

1. 创建SMB共享。您可以通过以下两种方式之一创建SMB管理员共享：使用"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
```

有关完整详细信息、请参见"创建 SMB 共享"。

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

参数	说明	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
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。对于**SMB**卷，必须设置为 **ntfs**`或 `**mixed**。	`ntfs`或 `mixed`SMB卷
unixPermissions	新卷的模式。对于SMB卷、必须留空。	""

配置存储类和**PVC**

配置Kubnetes StorageClass对象并创建存储类、以指示Trident如何配置卷。创建一个使用已配置的Kubernetes StorageClass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。

创建存储类。

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、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
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
```

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。

步骤

1. 这是一个Kubbernetes对象、因此、请使用 `kubectl`在Kubbernetes中创建它。

```
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
```

创建PV和PVC

A "*PersigentVolume*" (PV)是由集群管理员在Kubbernetes集群上配置的物理存储资源。 "*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PV和PVC后、您可以将卷挂载到Pod中。

示例清单

PerfsentVolume示例清单

此示例清单文件显示了与StorageClass关联的10gi的基本PV `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/my/host/path"
```

PersentVolumeClaim示例清单

这些示例显示了基本的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: basic-csi
```

采用**NVMe/TCP**的**PVC**

此示例显示了与名为的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:
  - ReadWriteMany
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

创建**PV**和**PVC**

步骤

1. 创建PV。

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

2. 验证PV状态。

```
kubectl get pv
NAME         CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. 创建PVC。

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

4. 验证PVC状态。

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

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"。

Trident属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	键入	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

属性	键入	值	优惠	请求	支持
后端类型	string	ontap-nas 、ontap-nas-economy. ontap-nas-flexgroup 、ontap-san 、solidfire-san 、GCP-CVS 、azure-netapp-files、ontap-san-economy.	池属于此类型的后端	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 ， gcp-cvs
克隆	池	true false	Pool 支持克隆卷	启用了克隆的卷	ontap-nas ， ontap-san ， solidfire-san ， gcp-cvs
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

部署示例应用程序

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
```

您可以使用监控进度 `kubectl get pod --watch`。

2. 验证卷是否已挂载在上 `/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。

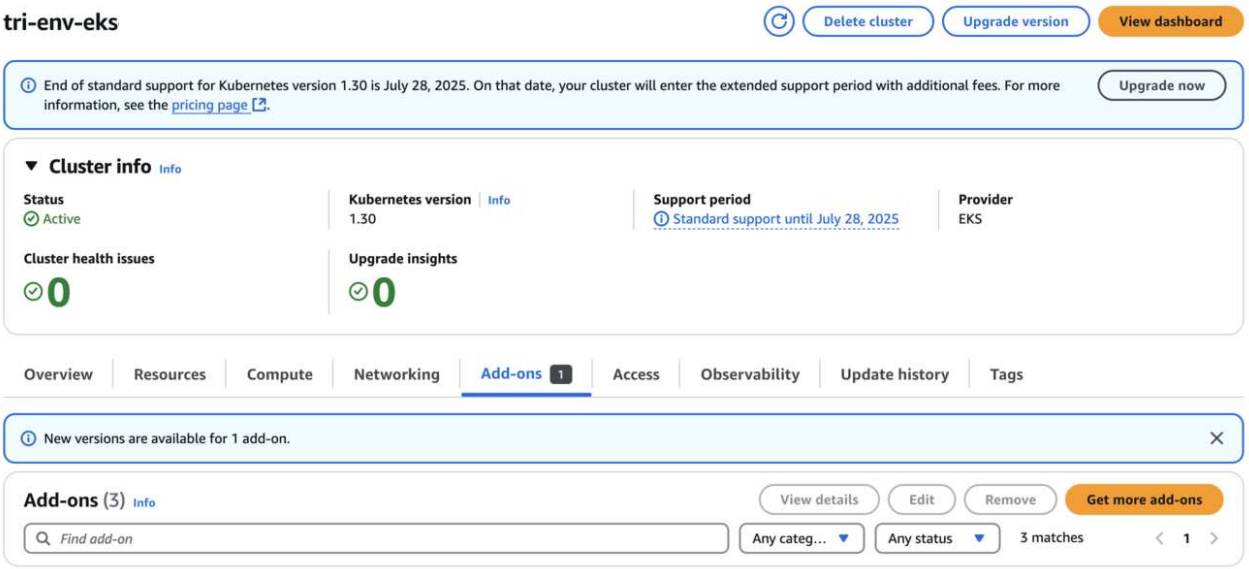
前提条件

在配置适用于AWS EKS的Trident加载项之前、请确保满足以下条件：

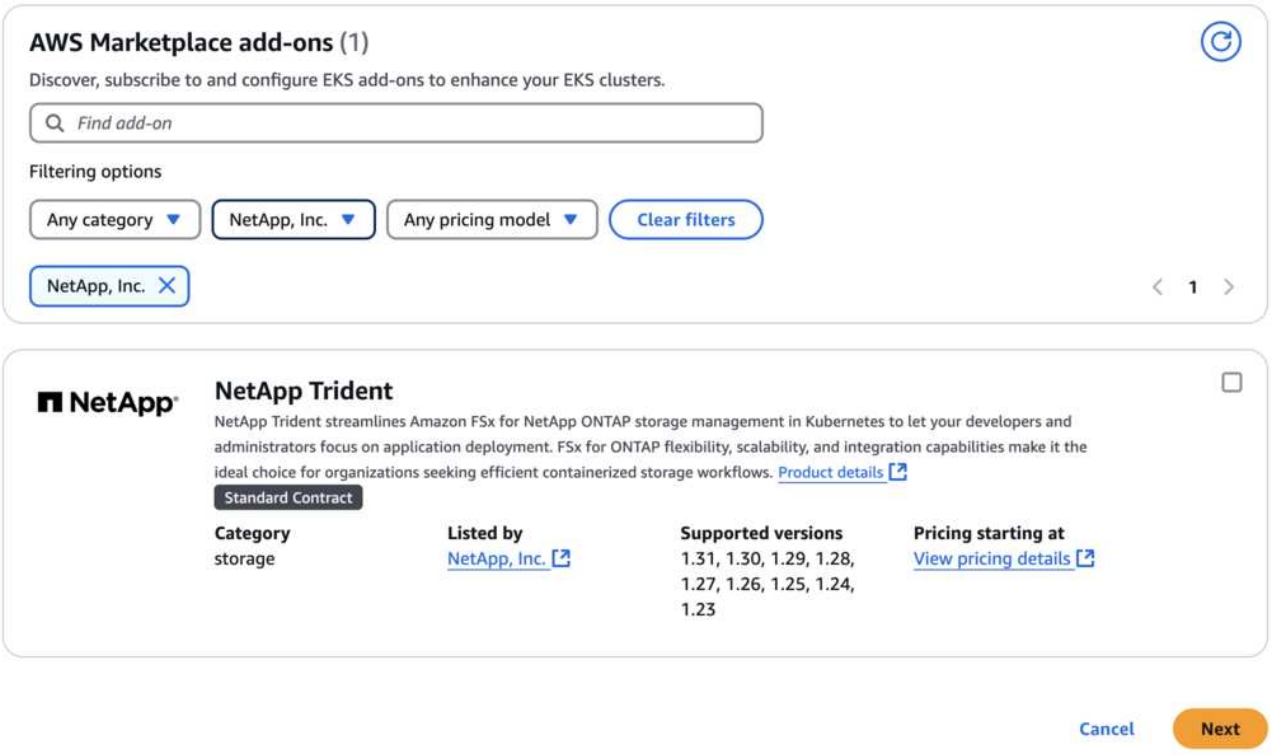
- 具有使用加载项的权限的Amazon EKS集群帐户。请参阅 "Amazon EKS附加项"。
- AWS对AWS Marketplace的权限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMI类型：Amazon ARM 2 (AL2_x86_64)或Amazon Linux 2 ARM (AL2_AMAZON_64)
- 节点类型：AMD或ARM
- 现有Amazon FSx for NetApp ONTAP文件系统

步骤

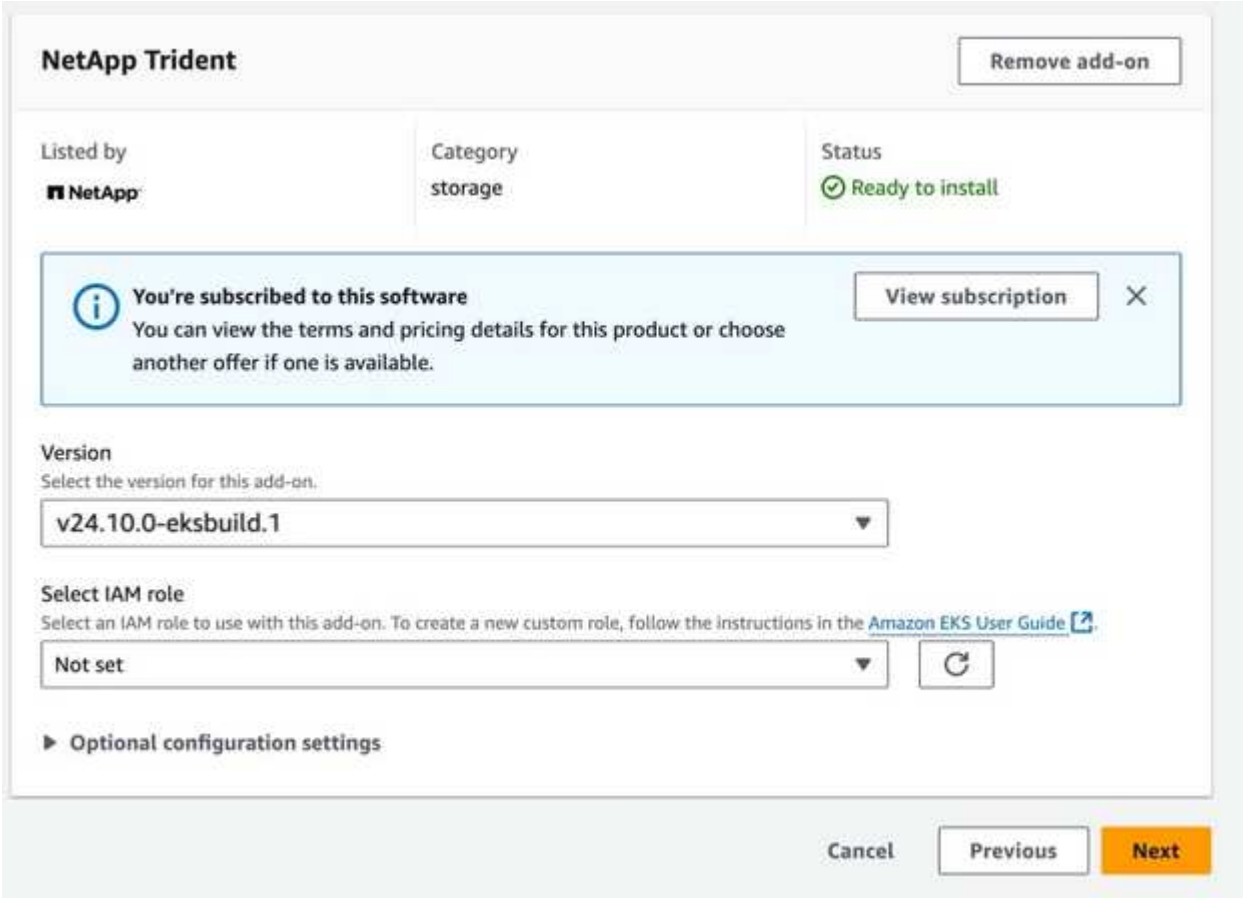
1. 请务必创建IAM角色和AWS密钥、以使EKS Pod能够访问AWS资源。有关说明，请参阅"创建IAM角色和AWS机密"。
2. 在EKS Kubernetes集群上、导航到*加载项*选项卡。



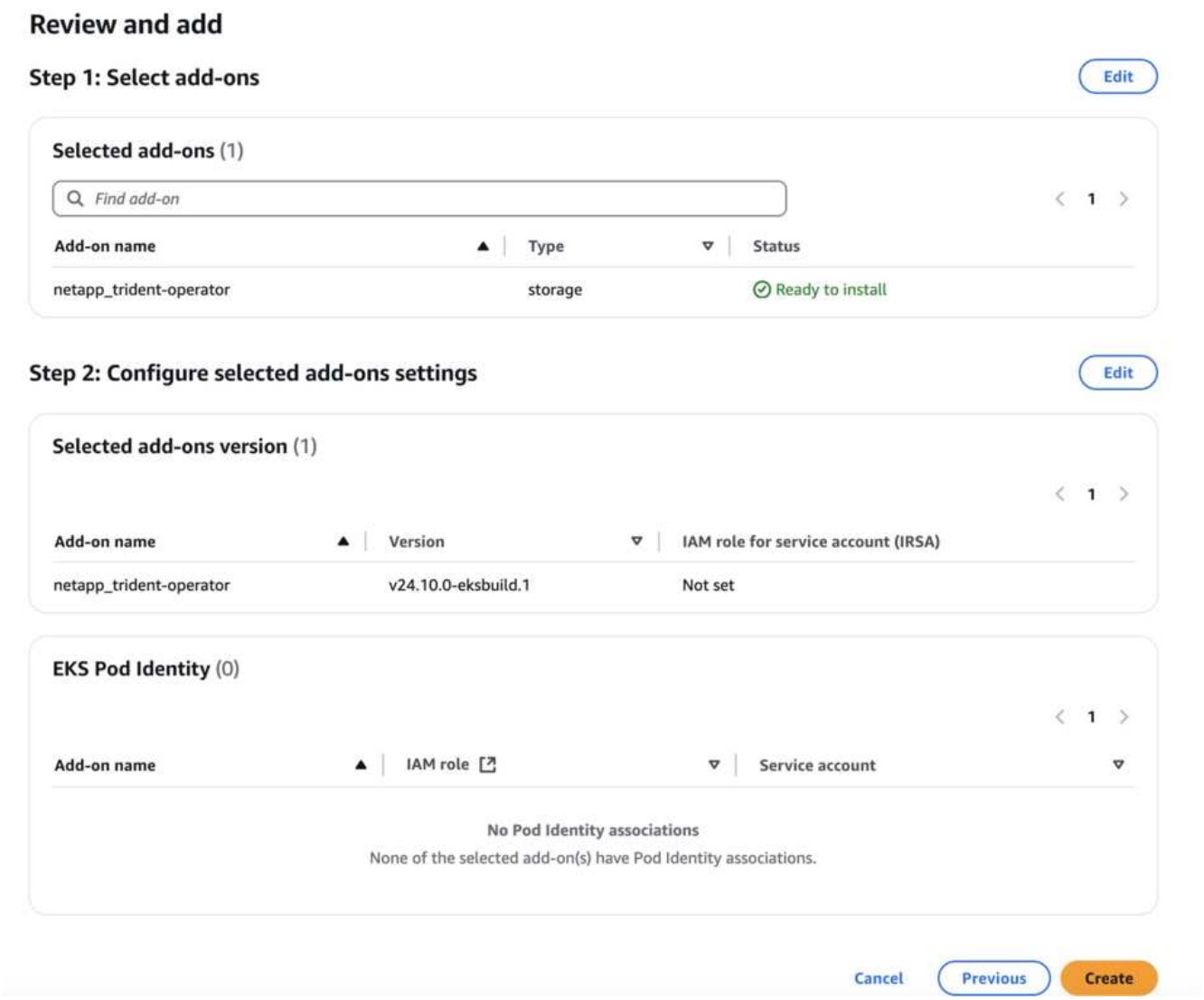
3. 转到*AWS Marketplace附加项*并选择_storage_类别。



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。



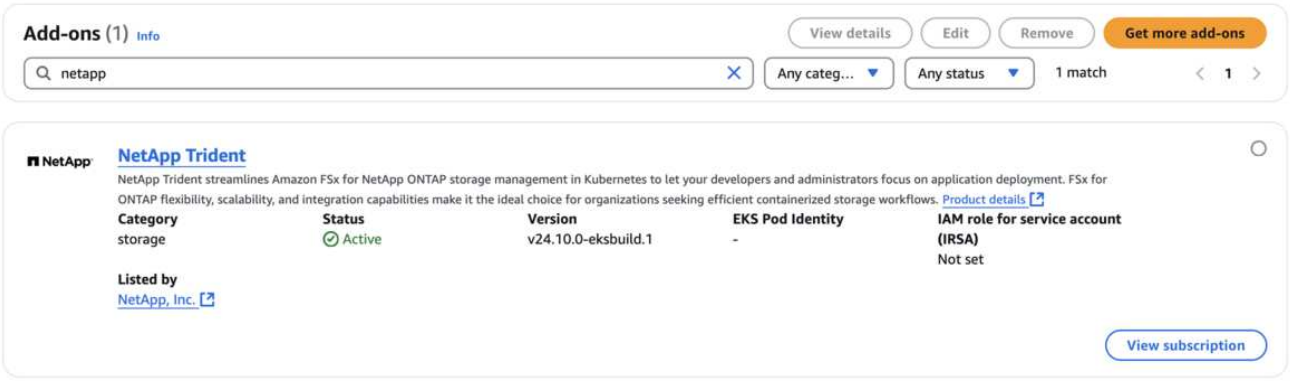
6. 选择要从节点继承的IAM角色选项。



7. 根据需要配置任何可选配置设置，然后选择*Next*。

遵循*附加配置模式*，并将*配置值*部分中的配置值参数设置为您在上一步(步骤1)中创建的ro-arn (值格式应为：)。 `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`注意：如果您为冲突解决方法选择覆盖、则现有加载项的一个或多个设置可能会被Amazon EKS加载项设置覆盖。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。

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：

以下示例命令将安装Trident EKS加载项：

```
eksctl create addon --name aws-ebs-csi-driver --cluster <cluster_name>
--service-account-role-arn arn:aws:iam::<account_id>:role/<role_name>
--force
```

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`)。

创建对象时 `TridentBackendConfig`、会发生以下情况：

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

每个都 `TridentBackendConfig`与保持一对一映射 `TridentBackend。前者是为用户提供的用于设计和配置后端的界面；后者是Trident如何表示实际后端对象。

> `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。

请参见以下CR格式示例 `TridentBackendConfig`：

```
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
```

您还可以查看目录中的示例、了解所需存储平台/服务的示例 "Trident 安装程序"配置。

`spec`采用特定于后端的配置参数。在此示例中、后端使用 `ontap-san`存储驱动程序、并使用此处表中列出的配置参数。有关所需存储驱动程序的配置选项列表，请参阅link:backends.html["存储驱动程序的后端配置信息"^]。

本 spec`节还包括 `credentials`和 `deletionPolicy`字段、这些字段是CR中新推出的 `TridentBackendConfig`：

- 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。
- deletionPolicy：此字段定义删除时应执行的操作 TridentBackendConfig。它可以采用以下两种可能值之一：
  - delete：这会导致删除 `TridentBackendConfig`CR和关联的后端。这是默认值。
  - retain： TridentBackendConfig`删除CR后，后端定义仍然存在，可以使用进行管理 `tridentctl。将删除策略设置为 retain`允许用户降级到早期版本(21.04之前的版本)并保留创建的后端。此字段的值可在创建后更新 `TridentBackendConfig。

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

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

要使用创建新的后端 kubectl，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、您可以使用观察其状态 `kubectl get tbc <tbc-name> -n <trident-namespace>`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
Cloud Volumes Service for GCP	private_key_id	专用密钥的 ID 。具有 CVS 管理员角色的 GCP 服务帐户的 API 密钥的一部分
Cloud Volumes Service for GCP	private_key	专用密钥。具有 CVS 管理员角色的 GCP 服务帐户的 API 密钥的一部分
Element （ NetApp HCI/SolidFire ）	端点	使用租户凭据的 SolidFire 集群的 MVIP
ONTAP	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
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

存储平台机密字段问题描述	机密	字段问题描述
ONTAP	用户名	入站用户名。如果 useCHAP=true ，则为必需项。对于 `ontap-san` 和 `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP 启动程序密钥。如果 useCHAP=true ，则为必需项。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true ，则为必需项。对于 `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。对于 `ontap-san`和 `ontap-san-economy`

此步骤中创建的机密将在下一步中创建的对象的字段 `TridentBackendConfig`中引用 `spec.credentials`。

第2步：创建 `TridentBackendConfig`CR

现在、您可以创建 `TridentBackendConfig`CR了。在此示例中、使用驱动程序的后端 `ontap-san`是使用以下对象创建的 `TridentBackendConfig`：

```
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CR的状态 TridentBackendConfig

创建CR后 TridentBackendConfig、您可以验证状态。请参见以下示例：

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

已成功创建后端并将其绑定到 `TridentBackendConfig`CR。

阶段可以采用以下值之一：

- Bound： `TridentBackendConfig`CR与一个后端关联，该后端包含 `configRef`设置为 `TridentBackendConfig`CR的uid。
- Unbound：使用表示 ""。 `TridentBackendConfig`对象不会绑定到后端。默认情况下、所有新创建的 `TridentBackendConfig`CRS都处于此阶段。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`已设置为删除。删除CR后 `TridentBackendConfig、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CR关联的后端 TridentBackendConfig`被意外或故意删除，而 `TridentBackendConfig`CR仍有对已删除后端的引用。 `TridentBackendConfig`无论值如何、均可删除CR `deletionPolicy。
- 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，您还可以获取的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>`命令以获取事件日志的详细信息。

> 您不能使用更新或删除包含关联对象 tridentctl`的后端 `TridentBackendConfig。要了解在和之间切换所涉及的 TridentBackendConfig`步骤 `tridentctl，"请参见此处"。

管理后端

使用 **kubectl** 执行后端管理

了解如何使用执行后端管理操作 kubectl。

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
```

您还可以运行 tridentctl get backend -n trident`或 `tridentctl get backend -o yaml -n trident`以获取所有已存在后端的列表。此列表还将包括使用创建的后端 `tridentctl。

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）。
  - 您可以使用以下命令直接通过Kubbernetes更新 `TridentBackendConfig`对象：

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

  - 或者、您也可以使用以下命令更改现有 `TridentBackendConfig`CR：

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

- 如果后端更新失败，则后端仍会保持在其上次已知配置中。您可以通过运行或 `kubectl describe tbc <tbc-name> -n trident`来查看日志以确定原因 `kubectl get tbc <tbc-name> -o yaml -n trident`。
- 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** 执行后端管理

了解如何使用执行后端管理操作 `tridentctl`。

创建后端

创建后"后端配置文件"，运行以下命令：

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

如果后端创建失败，则后端配置出现问题。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的问题后、只需再次运行命令即可 `create`。

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
```

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 " 删除 " 状态，而 Trident 将继续管理这些卷和快照，直到将其删除为止。

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的问题后、只需再次运行命令即可 `update`。

确定使用后端的存储类

以下是您可以使用为后端对象输出的JSON回答的问题示例 `tridentctl`。这将使用 `jq`您需要安装的实用程序。

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

这也适用于使用创建的后端 `TridentBackendConfig`。

在后端管理选项之间移动

了解在Trident中管理后端的不同方法。

用于管理后端的选项

随着的推出 `TridentBackendConfig`，管理员现在可以通过两种独特的方式管理后端。这会提出以下问题：

- 使用创建的后端是否 `tridentctl`可通过进行管理 `TridentBackendConfig`?
- 是否可以使用管理 `tridentctl`使用创建的后端 `TridentBackendConfig`?

使用管理 tridentctl`后端 `TridentBackendConfig

本节介绍通过创建对象直接通过Kubenetes界面创建后端所需的管理步骤 tridentctl。 TridentBackendConfig

这适用于以下情形：

- 已有的后端，因为它们是使用创建的，所以 tridentctl`没有 `TridentBackendConfig。
- 使用创建的新后端 tridentctl，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：

- 继续使用以管理使用 `tridentctl`它创建的后端。
- 将使用创建的后端绑定 tridentctl`到新 `TridentBackendConfig`对象。这样做意味着后端将使用而不是 `tridentctl`进行管理 `kubectl。

要使用管理已有的后端 kubectl，您需要创建 `TridentBackendConfig`绑定到现有后端的。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 (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

要创建 `TridentBackendConfig`绑定到现有后端的、您需要获取后端配置。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 `TridentBackendConfig`将自动绑定到已有的CR `ontap-nas-backend` (如本示例所示)。确保满足以下要求：

- 中定义了相同的后端名称 `spec.backendName`。
- 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CR的状态 `TridentBackendConfig`

创建后 `TridentBackendConfig`，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、可以使用对象完全管理后端 `tbc-ontap-nas-backend` `TridentBackendConfig`。

使用管理 `TridentBackendConfig`后端 `tridentctl`

`tridentctl`可用于列出使用创建的后端 `TridentBackendConfig`。此外，管理员还可以选择通过删除并确保 `spec.deletionPolicy`将设置为 `retain`来 `TridentBackendConfig`完全管理此类后端 `tridentctl`。

第 **0** 步：确定后端

例如，假设以下后端是使用创建的 `TridentBackendConfig`：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

从输出中可以看出、 `TridentBackendConfig`已成功创建并绑定到后端[观察后端的UUID]。

步骤**1**：确认 `deletionPolicy`设置为 `retain`

让我们来看看的价值 `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
```

除非将设置为， `retain`否则请勿继续下一步 `deletionPolicy`。

第2步：删除 `TridentBackendConfig`CR

最后一步是删除 TridentBackendConfig`CR。确认已设置为 `retain`后 `deletionPolicy，您可以继续删除：

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+------------------+---------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+------------------+---------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+------------------+---------------
+--------------------------------------+--------+---------+
```

删除对象后 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: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
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`在Kubbernetes中创建它。

```
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、您也可以编辑 `sample-input/storage-class-csi.yaml.templ`安装程序随附的文件、并替换 `*BACKEND_TYPE*`为存储驱动程序名称。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

管理存储类

您可以查看现有存储类、设置默认存储类、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设置为true来定义默认存储类 `storageclass.kubernetes.io/is-default-class`。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。

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>
```

`<storage-class>`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。

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

配置卷

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。

概述

A "*PersigentVolume*" (PV)是由集群管理员在Kubbernetes集群上配置的物理存储资源。 "*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PV和PVC后、您可以将卷挂载到Pod中。

示例清单

PerfsentVolume示例清单

此示例清单文件显示了与StorageClass关联的10gi的基本PV `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

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: 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
```

基本**NVMe/TCP**配置

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      resources: {}
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: pvc-san-nvme
```

创建**PV**和**PVC**

步骤

1. 创建PV。

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

2. 验证PV状态。

```
kubectl get pv
NAME        CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. 创建PVC。

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

4. 验证PVC状态。

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

5. 将卷挂载到Pod中。

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

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

6. 验证卷是否已挂载在上 `/my/mount/path`。

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

7. 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
```

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"。

展开卷

Trident使Kubnetes用户能够在创建卷后对其进行扩展。查找有关扩展 iSCSI 和 NFS 卷所需配置的信息。

展开 **iSCSI** 卷

您可以使用 CSI 配置程序扩展 iSCSI 永久性卷（ PV ）。

> iSCSI卷扩展受、、 `ontap-san-economy` `solidfire-san`驱动程序支持 `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定义以将字段设置 `allowVolumeExpansion`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
```

对于已有的StorageClass、请对其进行编辑以包含 `allowVolumeExpansion`参数。

第 2 步：使用您创建的 StorageClass 创建 PVC

编辑PVC定义并更新以反映新需要的 `spec.resources.requests.storage`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，创建了一个使用的POD `san-pvc`。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

第4步：展开PV

要将已创建的PV从1Gi调整为2Gi、请编辑PVC定义并将更新 `spec.resources.requests.storage`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支持对、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `gcp-cvs`和 `azure-netapp-files`后端配置的NFS PV进行卷扩展 `ontap-nas`。

第 **1** 步：配置 **StorageClass** 以支持卷扩展

要调整NFS PV的大小，管理员首先需要通过将字段设置为来将存储类配置为 `true`允许卷扩展 `allowVolumeExpansion`：

```
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
```

如果您已在不使用此选项的情况下创建了存储类、则只需使用编辑现有存储类即可 `kubectl edit storageclass`允许卷扩展。

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应为此PVC创建一个20MiB 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20MiB PV调整为1GiB、请编辑PVC并设置 `spec.resources.requests.storage`为1GiB：

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

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的持久卷声明 (PVC) 将现有存储卷导入为 Kubernetes PV。

概述和注意事项

您可以将卷导入到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。

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 。
  - 在PV中、回收策略最初设置为 `retain`。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。
  - 如果存储类的回收策略为 `delete`，则删除PV时将删除存储卷。
- 默认情况下、Trident管理PVC并在后端重命名FlexVol和LUN。您可以传递此 `--no-manage`标志以导入非受管卷。如果使用 `--no-manage`，则在对象的生命周期内，Trident不会对PVC或PV执行任何附加操作。删除PV后、不会删除存储卷、并且卷克隆和卷大小调整等其他操作也会被忽略。

如果要对容器化工作负载使用 Kubernetes ，但希望在 Kubernetes 外部管理存储卷的生命周期，则此选项非常有用。

- 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。

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
   ```

   仅包括所需的参数。PV 名称或卷大小等附加参数可能导致导入命令失败。

2. 使用 `tridentctl import`命令指定包含卷的Trident后端的名称以及在存储上唯一标识卷的名称(例如：ONTAP FlexVol、Element卷、Cloud Volumes Service路径)。 `-f`指定PVC文件的路径需要参数。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

使用 **PVC** 注释

步骤

1. 使用所需的 Trident 导入注释创建 PVC YAML 文件（例如， `pvc.yaml`）。PVC 文件应包括：

   - `name` 和 `namespace` 在元数据中
   - `accessModes`、 `resources.requests.storage` 和 `storageClassName` 在规格中
   - 标注：
     - `trident.netapp.io/importOriginalName`: 后端上的卷名称
     - `trident.netapp.io/importBackendUUID`：卷存在的后端 UUID
     - `trident.netapp.io/notManaged` (*Optional*)：为非托管卷设置为 `"true"`。默认值为 `"false"`。

     下面是导入托管卷的示例规范：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. 将 PVC YAML 文件应用到 Kubernetes 集群：

```
kubectl apply -f <pvc-file>.yaml
```

Trident 将自动导入卷并将其绑定到 PVC。

示例

查看以下卷导入示例、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。

- `ontap-nas-economy`驱动程序无法导入和管理qtrees。
- `ontap-nas`和 `ontap-nas-flexgroup`驱动程序不允许卷名称重复。

使用驱动程序创建的每个卷 `ontap-nas`都是ONTAP集群上的一个FlexVol。使用驱动程序导入FlexVol的 `ontap-nas`工作原理相同。可以将ONTAP集群上已存在的FlexVol导入为 `ontap-nas`PVC。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

以下示例将在名为的后端导入名为的 `ontap_nas`卷 `managed_volume`：

```
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。

以下示例将在 `ontap_nas`后端导入 `unmanaged_volume`：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

使用 **PVC** 注释的 **ONTAP NAS** 示例

以下示例演示如何使用 PVC 注释导入托管和非托管卷。

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>
```

ONTAP SAN

Trident支持使用和 `ontap-san-economy`驱动程序导入卷 `ontap-san`。

Trident可以导入包含单个LUN的ONTAP SAN FlexVol。这与驱动程序一致 `ontap-san`、该驱动程序会为FlexVol中的每个PVC和LUN创建一个FlexVol。Trident导入FlexVol并将其与PVC定义关联。

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重命名为格式，并将FlexVol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` `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

以下示例将在 `ontap_san`后端导入 `unmanaged_example_volume`：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

如果您将LUN映射到与Kubornetes节点IQN共享IQN的igroups (如以下示例所示)，则会收到错误消息： `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
```

Element

Trident支持使用驱动程序导入NetApp Element软件和NetApp HCI卷 `solidfire-san`。

> 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。

以下示例将在后端导入 `element-managed`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    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud Platform

Trident支持使用驱动程序导入卷 `gcp-cvs`。

> 要在Google云平台中导入NetApp Cloud Volumes Service支持的卷、请按卷路径确定该卷。卷路径是卷的导出路径中在之后的部分 `:/`。例如，如果导出路径为 `10.0.0.1:/adroit-jolly-swift`，则卷路径为 `adroit-jolly-swift`。

Google Cloud Platform示例

以下示例将在后端导入 `gcp-cvs`卷路径为的 `adroit-jolly-swift`卷 `gcpcvs_YEppr`。

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

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

Azure NetApp Files

Trident支持使用驱动程序导入卷 `azure-netapp-files`。

> 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中在之后的部分 `:/`。例如，如果挂载路径为 `10.0.0.2:/importvol1`，则卷路径为 `importvol1`。

以下示例将在后端导入 `azure-netapp-files`卷路径为的 `importvol1`卷 `azurenetappfiles_40517`。

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

自定义卷名称和标签

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：

1. 卷创建、导入和克隆操作。
2. 如果使用的是ONTA-NAS经济型驱动程序、则只有qtree卷的名称符合此名称模板。
3. 如果使用的是ONONTAP SAN经济版驱动程序、则只有LUN名称符合此名称模板。

限制

1. 可自定义的卷名称仅与ONTAP内部部署驱动程序兼容。
2. 可自定义的卷名称不适用于现有卷。

可自定义卷名称的关键行为

1. 如果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。
2. 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。

在命名空间之间共享NFS卷

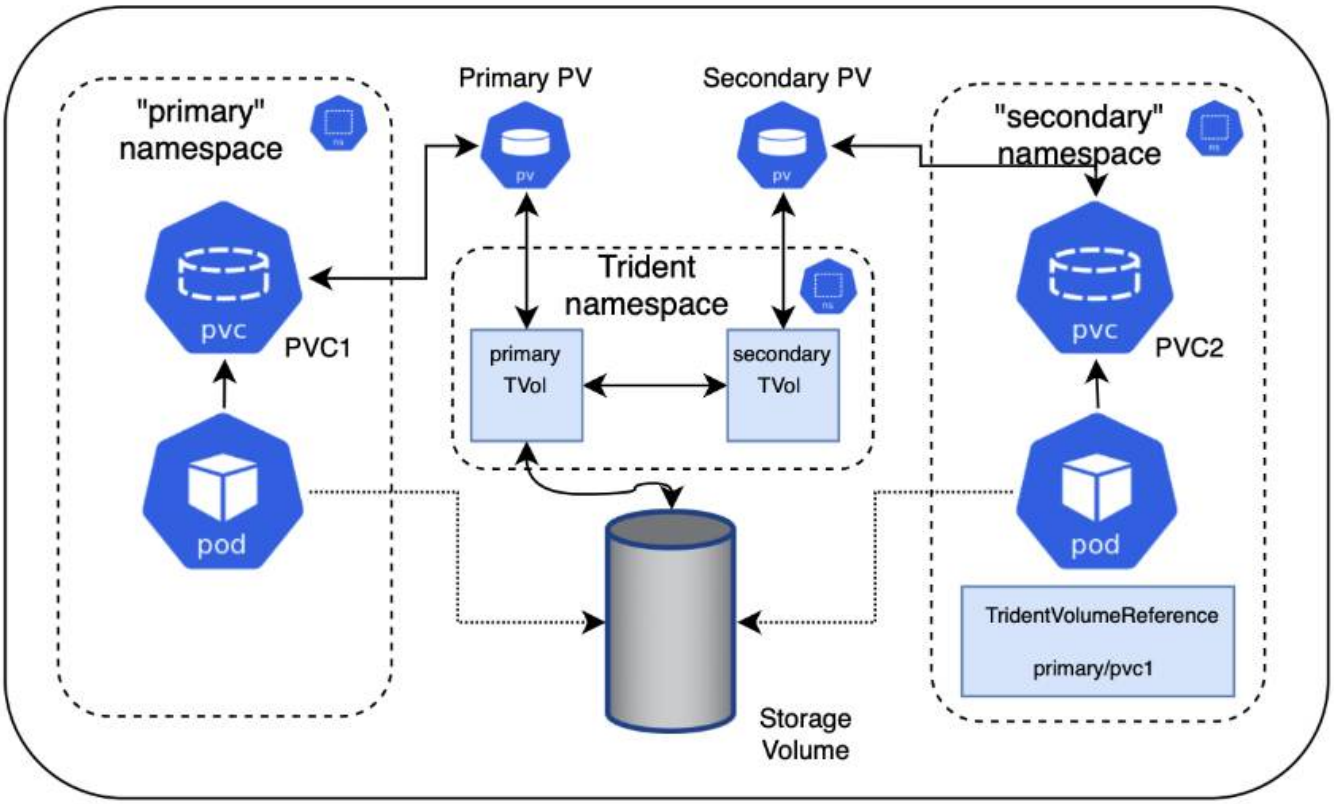
使用Trident、您可以在主命名空间中创建卷、并在一个或多个二级命名空间中共享该卷。

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**Trident**卷 引用

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. **Source**命名空间**owner:**(`pvc1`在源命名空间中创建`PVC`，该`PVC`用于授予与目标命名空间共享的权限(`namespace2`)。 `shareToNamespace`

```
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。

- 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 `*`的文件。例如、`trident.netapp.io/shareToNamespace: *`
- 您可以随时更新PVC以包含 `shareToNamespace`标注。

2. *集群管理员：*创建自定义角色并执行kubeconfig、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trident卷 引用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 `shareFromPVC`标注指定源PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

目标PVC的大小必须小于或等于源PVC。

结果

Trident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。

`tridentctl get`用于查询子卷

您可以使用[`tridentctl`实用程序运行 `get`命令以获取子卷。有关详细信息、请参阅链接：`Trident tridentctl.html[`tridentctl` commands and options]。

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

flags

- `-h, --help`：卷帮助。
- `--parentOfSubordinate string`：将查询限制为从属源卷。
- `--subordinateOf string`：将查询限制为卷的子卷。

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。
- 您不能通过删除或 `shareFromNamespace`标注或删除`CR`来撤消对源`PVC`的 `TridentVolumeReference`访问 `shareToNamespace`。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。

了解更多信息

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：

- 请访问。"在命名空间之间共享卷：对跨命名空间卷访问说Hello"
- 观看上的演示 "NetAppTV"。

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。您可以使用具有名称流的自定义资源定义(CRD)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" 。

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" 。

  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

- ONTAP -NAS和ONTAP SAN驱动程序支持卷复制。

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
        volumeMappings:
        - localPVCName: csi-nas
          remoteVolumeHandle:
      "datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
      ```

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

c. 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Kubnetes集群上、创建SnapshotInfo CR以获取内部详细信息。

   示例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. 在二级Kubernetes集群上、将_TridentMirorRelationship_ CR的_spec.state_字 段更新为_promoted_和_spec.promotedSnapshotHandle_、以成为快照的内部名称。
4. 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. 在二级Kubernetes集群上，将Trident镜像 关系的_spec.mirector_字段更新到 `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。

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。
- 将设置为 `WaitForFirstConsumer`时，为PVC创建和绑定永久性卷的操作将延迟到计划和创建使用PVC的Pod时 `VolumeBindingMode`才进行。这样，卷就会根据拓扑要求强制实施的计划限制来创建。

`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Kub并 网集群"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可引入拓扑感知的标签(`topology.kubernetes.io/region`和 `topology.kubernetes.io/zone`)。在安装Trident之前，这些标签*应出现在群集中的节点上*，以便Trident能够识别拓扑。

```
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

```
---
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

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "password",
 "supportedTopologies": [
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-a"},
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-b"}
]
}
```

`supportedTopologies`用于提供每个后端的区域和分区列表。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。

您也可以定义 `supportedTopologies`每个存储池。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.kubernetes.io/region`并 `topology.kubernetes.io/zone`指定存储池的使用来源。

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:
name: pvc-san
spec:
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Kubornetes在区域中的节点上计划POD `us-east1`、并从或 `us-east1-b`区域中的任何节点中进行选择 `us-east1-a`。

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
```

更新后端以包含 `supportedTopologies`

可以更新已有的后端以包括使用 `tridentctl backend update`列表 `supportedTopologies`。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

、 `ontap-nas-flexgroup`、 `ontap-san`、、 `ontap-san-economy`支持卷快照 `ontap-nas` `solidfire-san` `gcp-cvs`和 `azure-netapp-files`驱动程序。

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Kubnetes分发不包括快照控制器和CRD，请参阅[部署卷快照控制器]。

如果在GKE环境中创建按需卷快照、请勿创建快照控制器。GKE-使用内置的隐藏快照控制器。

创建卷快照

步骤

1. 创建 `VolumeSnapshotClass`。有关详细信息，请参阅。"VolumeSnapshotClass"

- `driver`指向Trident CSI驱动程序。
- `deletionPolicy`可以是 `Delete`或 `Retain`。如果设置为 `Retain`，则即使删除对象，存储集群上的底层物理快照也会保留 `VolumeSnapshot`。

  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
```

   - 以下示例将为名为的PVC创建卷快照对象，并且快照 `pvc1`的名称设置为 `pvc1-snap`。卷快照类似于PVC、并与表示实际快照的对象相关联 `VolumeSnapshotContent`。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

   - 您可以通过对卷快照对象进行描述来确定 `VolumeSnapshotContent`该对象 `pvc1-snap`。`Snapshot Content Name`标识提供此快照的卷SnapshotContent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`创建使用名为作为数据源的卷快照的`PVC `<pvc-name>`。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。

PVC将与源卷在同一后端创建。请参阅 "知识库文章：无法在备用后端创建从三端PVC Snapshot创建PVC"。

以下示例将使用作为数据源创建PVC `pvc1-snap`。

```
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。
   - 在中将后端快照的名称指定 `annotations`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。

     示例

     以下示例将创建一个 `VolumeSnapshotContent`引用后端Snapshot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. *Cluster admin:*创建引用对象的 `VolumeSnapshot`CR `VolumeSnapshotContent`。此操作将请求访问以在给定命名空间中使用 `VolumeSnapshot`。

   示例

   以下示例将创建一个 `VolumeSnapshot`名为的CR，该CR引用名为 `import-snap`的 `VolumeSnapshotContent 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`。

   - 外部快照程序将设置为，将 `VolumeSnapshot`设置 `VolumeSnapshotContent`为 `readyToUse true`。
   - Trident返回 `readyToUse=true`。

4. *any user:*创建 `PersistentVolumeClaim`引用新的的 `VolumeSnapshot`，其中 `spec.dataSource`(或 `spec.dataSourceRef`)名是 `VolumeSnapshot`名称。

示例

以下示例将创建一个引用名为 `import-snap`的的`PVC `VolumeSnapshot`。

```
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、以便最大程度地兼容使用和 `ontap-nas-economy`驱动程序配置的卷 `ontap-nas`。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
```

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支持在 `ontap-san`、、 `ontap-san-economy` `ontap-nas`、 `ontap-nas-flexgroup` `azure-netapp-files`、、 `gcp-cvs` `google-cloud-netapp-volumes`和 `solidfire-san`驱动程序。

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`。

> 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
```

1. 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: ""
```

- 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
```

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"

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。
- 如果您在中创建了适用于Google Cloud的Cloud Volumes Service"CVS 服务类型"，则在从Trident 23.01升级时，必须更新后端配置才能使用 `standardsw`或 `zoneredundantstandardsw`服务级别。如果无法在后端更新 `serviceLevel`、可能会导致卷失败。有关详细信息、请参见 "CVS服务类型示例" 。
- 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。
  - 这是对SAN卷强制实施的要求 "安全上下文"。
  - https://github.com/NetApp /AML/tree/master Trident /AML-installer/same-input[sSample Input^]目录包含一些示例、例如https://github.com/NetApp Trident /AML/Blob/master /AMER-installer/same-input/storage-class-sams/storage-class-Basic ^。Trident TridentTrident Trident ^)和链接：https://github.com/NetApp /AML/Blob/master /AML-installer/sSample -input/storage-class-lone -sexstorage-stamly-default][`storage-class-basic.yaml.templ`。[`storage-class-bronze-default.yaml`
  - 有关详细信息，请参阅 "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23.04 (或任何23.04 DOT版本)升级到24.06。
- 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22.01并希望升级到24.06：
  a. 首次从22.07升级到23.04。
  b. 然后从23.04升级到24.06。

在OpenShift容器平台上使用TRIDent操作程序进行升级时、应升级到TRIDent 21.01.1或更高版本。21.01.0 版发布的 Trident 运算符包含一个已知的问题描述，该 已在 21.01.1 中修复。有关详细信息，请参阅 "GitHub 上的问题描述详细信息"。

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

通常，您应该使用与初始安装相同的方法进行升级，但您可以这样做"在安装方法之间切换"。升级Trident有两种方法。

- "使用Trident操作符进行升级"

  我们建议您在升级之前与操作员一起查看"了解操作员升级工作流"。

*

使用操作员升级

了解操作员升级工作流

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

要安装和升级Trident的众多操作之一"使用啮合式操作员的优势"是自动处理Trident和Kubbernetes对象、而不会中断已挂载的现有卷。通过这种方式，Trident可以在零停机时间或"滚动更新"的情况下支持升级。尤其是、通过与Kubenetes集群进行通信、可以：

- 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。

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、也可以从安装升级 `tridentctl`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三端技术人员安装升级到另一个集群范围的三端技术人员安装。所有Trident 21.01及更高版本都使用集群范围的运算符。

> 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Kubbernetes 1.24的集群，请使用 "捆绑包_pre_1_25.yaml"。
- 对于运行Kubernetes 1.25或更高版本的集群，请使用 "捆绑包_后_ 1_25.yaml"。

开始之前

确保您使用的是运行的Kubbernetes集群"支持的Kubernetes版本"。

步骤

1. 验证Trident版本：

```
./tridentctl -n trident version
```

2. 删除用于安装当前Trident实例的Trident运算符。例如、如果要从23.07升级、请运行以下命令：

```
kubectl delete -f 23.07.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

3. 如果您使用属性自定义了初始安装 `TridentOrchestrator`、则可以编辑此 `TridentOrchestrator`对象以修改安装参数。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。
4. 使用适用于您的环境的正确捆绑包YAML文件安装Trident、其中_是 或 `bundle_post_1_25.yaml`基于您的<bundle.yaml>
`bundle_pre_1_25.yaml`版本。例如、如果要安装Trident 24.10、请运行以下命令：

```
kubectl create -f 24.10.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

升级Helm安装

您可以升级Trident Helm安装。

> 在将安装了Trident的Kubernetes集群从1.24升级到1.25或更高版本时、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"使用，则可以使用 `helm upgrade trident netapp-trident/trident-operator --version 100.2410.0` 一步升级。如果您未添加Helm repo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. 使用 `helm upgrade` 命令、其中 `trident-operator-24.10.0.tgz` 反映了要升级到的版本。

```
helm upgrade <name> trident-operator-24.10.0.tgz
```

如果您在初始安装期间设置了自定义选项(例如、为Trident和CSI映像指定专用、镜像注册表)、请使用附加 `helm upgrade`命令 `--set`以确保升级命令中包含这些选项、否则这些值将重置为默认值。

2. 运行 `helm list`以验证图表和应用程序版本均已升级。运行 `tridentctl logs`以查看任何调试消息。

从安装升级 `tridentctl`到Trident Operator

您可以从安装升级到最新版本的Trident operator `tridentctl`。现有后端和PVC将自动可用。

在切换安装方法之前，请查看"在安装方法之间移动"。

步骤

1. 下载最新版本的Trident。

```
# Download the release required [24.10.0]
mkdir 24.10.0
cd 24.10.0
wget
https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

2. 从清单创建 `tridentorchestrator`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:                v24.10.0
```

使用 **tridentctl** 进行升级

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

使用升级前请 `tridentctl`查看"选择升级方法"。

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"。

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`、或者将标志附加 `--help`到任何命令以获取该特定命令的选项和标志列表。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl`实用程序支持以下命令和全局标志。

命令

create

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。

uninstall

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。

version

打印Trident版本。

全局标志

-d、 --debug

调试输出。

-h、 --help

帮助 `tridentctl`。

-k、 --kubeconfig string

指定 `KUBECONFIG`在本地或从一个Kubornetes集群到另一个集群运行命令的路径。

或者、您也可以导出此 `KUBECONFIG`变量以指向特定Kubnetes集群、然后向该集群发出 `tridentctl`命令。

-n、 --namespace string

Trident部署的命名空间。

-o、 --output string

输出格式。json_yaml_name_wide|ps 之一（默认）。

-s、 --server string

Trident REST接口的地址/端口。

可以将 Trident REST 接口配置为仅以 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`：卷的帮助。
`--parentOfSubordinate string`：将查询限制为从属源卷。
`--subordinateOf string`：将查询限制为卷的子卷。

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：AutoSupport遥测的容器图像(默认为“NetApp / Trident AutoSupport：<current-version>”)。
--autosupport-proxy string：用于发送AutoSupport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：生成YAML文件而不安装任何内容。
-h --help：安装帮助。
--http-request-timeout：覆盖Trident控制器REST API的HTTP请求超时(默认值为1m30s)。
--image-registry string：内部映像注册表的地址/端口。
--k8s-timeout duration：所有KubeNet操作的超时(默认值为3m0)。
--kubelet-dir string：kubelet内部状态的主机位置(默认为"/var/lib/kubelet")。
--log-format string：Trident日志记录格式(文本，json)(默认为“文本”)。
--node-prep：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。**
`--pv string：Trident使用的原有PV的名称确保不存在(默认为“Trident”)。
--pvc string：Trident使用的原有PVC的名称确保不存在(默认为“Trident”)。
--silence-autosupport：不自动向NetApp发送AutoSupport分发包(默认为true)。
--silent：在安装期间禁用大多数输出。
--trident-image string：要安装的Trident映像。
--use-custom-yaml：使用安装目录中的任何现有YAML文件。
--use-ipv6：使用IPv6进行Trident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归档发送到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`命令暂停或恢复后端操作。

```
tridentctl update backend state <backend-name> [flag]
```

需要考虑的要点

- 如果使用TridentBackendConfig (tbc)创建后端、则无法使用文件更新后端 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`可暂停后端操作。设置为 `normal`可恢复后端操作。当设置为时 `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. 使用命令和更新后的文件的路径更新后端 `tridentctl backend update` `backend.json` 。

   示例： `tridentctl backend update -f /<path to backend JSON file>/backend.json`

```
{
    "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`标志可打印的版本 `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

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。

默认情况下、Trident的指标会在端点的 `/metrics`目标端口上公开 `8001`。安装 Trident 时，这些指标默认为 * 已启用 * 。

您需要的内容

- 安装了Trident的Kubnetes集群。
- 一个 Prometheus 实例。这可以是 "容器化 Prometheus 部署"，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

您应定义Prometheus目标、以收集指标并获取有关Trident管理的后端及其创建的卷等的信息。这 "博客"说明了如何将Prometheus和Grafana与Trident结合使用来检索指标。本博客介绍了如何以操作员身份在Kubelnetes集群中运行Prometheus、以及如何创建ServiceMonitor来获取Trident指标。

第 2 步：创建 Prometheus ServiceMonitor

要使用Trident指标、您应创建一个Prometheus ServiceMonitor来监控 `trident-csi`服务并侦听 `metrics`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定义检索服务返回的指标 `trident-csi`、并专门查找 `metrics`服务的端点。因此、Prometheus现在可以配置为了解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"。

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

> 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（ PiII ）或个人信息。它附带的 "EULA" 不适用于三端存储容器映像本身。您可以详细了解NetApp对数据安全和信任的承诺 "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:
    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、则可以使用标志。 `--image-registry`
- 您也可以通过生成安装 YAML 文件来配置代理 URL 。这可以通过使用创建YAML文件并在中为容器 `trident-deployment.yaml`添加 `--proxy-url`参数 `trident-autosupport`来实现 `tridentctl install --generate-custom-yaml`。

禁用Trident指标

要禁止报告**度量指标，应生成自定义YAML (使用 `--generate-custom-yaml`标志）并对其进行编辑，以删除 `--metrics`为树枝调用的标志 `trident-main`。

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}}'
   ```

   当该标志设置为 `true`时 `uninstall`，Trident操作员卸载Trident，但不会删除TridentOrchestrator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的"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，请 "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
```

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
```

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 `open-iscsi`和 `multipath-tools`已启用且正在运行：

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

NVMe工具

使用适用于您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
```

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）

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
```

如果您的版本已过时，请 "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`：默认文件名是 `config.json`，但是您可以通过指定带有文件名的选项来使用所选的任何名称 `config`。配置文件必须位于主机系统上的目录中 `/etc/netappdvp`。
   - `log-level`：指定日志记录级别(`debug`、 `info`、 `warn`、 `error` `fatal`)。默认值为 `info`。
   - `debug`：指定是否启用调试日志记录。默认值为 false 。如果为 true ，则覆盖日志级别。
     i. 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/v24.10.0/trident-
installer-24.10.0.tar.gz
tar zxf trident-installer-24.10.0.tar.gz
```

b. 移动到托箱路径中的某个位置：

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/
sudo chown root:root /usr/local/bin/trident
sudo chmod 755 /usr/local/bin/trident
```

c. 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三叉进制守护进程。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

除非指定、否则卷驱动程序的默认名称为NetApp。

启动守护进程后，您可以使用 Docker 命令行界面创建和管理卷

3. 创建卷

```
docker volume create -d netapp --name trident_1
```

4. 启动容器时配置 Docker 卷：

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
alpine ash
```

5. 删除 Docker 卷：

```
docker volume rm trident_1
docker volume rm trident_2
```

在系统启动时启动Trident

有关基于systemd的系统的单元文件示例、请参见 `contrib/trident.service.example`Git repo。要对RHEL使用此文件、请执行以下操作：

1. 将文件复制到正确的位置。

   如果正在运行多个实例，则单元文件应使用唯一名称。

   ```
   cp contrib/trident.service.example
   /usr/lib/systemd/system/trident.service
   ```

2. 编辑文件，更改问题描述（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 以载入更改：

   ```
   systemctl daemon-reload
   ```

4. 启用服务。

   此名称因您在目录中为文件命名的内容而异 `/usr/lib/systemd/system`。

   ```
   systemctl enable trident
   ```

5. 启动服务。

   ```
   systemctl start trident
   ```

6. 查看状态。

   ```
   systemctl status trident
   ```

任何时候修改单元文件时、请运行 `systemctl daemon-reload`命令使其了解所做的更改。

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
```

> 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
```

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
```

> 如果未指定任何"选项"，则使用驱动程序的默认值。

- 覆盖默认卷大小。要使用驱动程序创建 20GiB 卷，请参见以下示例：

```
docker volume create -d netapp --name my_vol --opt size=20G
```

> 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 ）。如果未指定任何单位、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
```

> 使用驱动程序时 `solidfire-san`、上述示例将删除和清除卷。

执行以下步骤升级适用于Docker的Trident。

克隆卷

使用 `ontap-nas`、、 `ontap-san` `solidfire-san`和 `gcp-cvs` `storage drivers`时，Trident可以克隆卷。使用或 `ontap-nas-economy`驱动程序时 `ontap-nas-flexgroup`、不支持克隆。从现有卷创建新卷将创建新快照。

- 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操作符提供选项和值 `-o`。这些参数将覆盖 JSON 配置文件中的任何等效值。

ONTAP 卷选项

NFS 和 iSCSI 的卷创建选项包括以下内容：

选项	说明
`size`	卷的大小默认为1 GiB。
`spaceReserve`	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)。
`snapshotPolicy`	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。除非存储管理员修改，否则所有 ONTAP 系统上都存在一个名为 "defaultion" 的策略，该策略会创建并保留六个每小时快照，两个每日快照和两个每周快照。可以通过浏览到卷中任何目录中的目录来恢复快照中保留的数据 `.snapshot`。
`snapshotReserve`	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。

选项	说明
encryption	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"。
tieringPolicy	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	说明
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，并且 `root`将是所有者。文本或数字格式均可使用。
snapshotDir	将此选项设置为 true`将使访问此卷的客户端可以看到此 `.snapshot`目录。默认值为 `false，表示默认情况下禁用目录可见性 .snapshot。某些映像(例如官方MySQL映像)在目录可见时无法按预期运行 .snapshot。
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 `mixed。

以下附加选项适用于 iSCSI * 仅 * ：

选项	说明
fileSystemType	设置用于格式化 iSCSI 卷的文件系统。默认值为 ext4。有效值为 ext3、 ext4`和 `xfs。
spaceAllocation	将此选项设置为 false`将关闭LUN的空间分配功能。默认值为 `true，表示ONTAP会在卷空间用尽且卷中的LUN无法接受写入时通知主机。此选项还允许 ONTAP 在主机删除数据时自动回收空间。

示例

请参见以下示例：

- 创建 10 GiB 卷：

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- 创建具有快照的 100GiB 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 20MiB 。

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
```

- 使用快照策略创建卷，并接受 ONTAP 的默认快照预留（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、系统会使用术语指定与其关联的QoS策略 `-o type=service_level`。

使用 Element 驱动程序定义 QoS 服务级别的第一步是至少创建一种类型，并指定与配置文件中的名称关联的最小，最大和突发 IOPS 。

其他 Element 软件卷创建选项包括：

选项	说明
`size`	卷大小、默认为1GiB或配置条目…"默认值"： {"size"：5g"} 。
`blocksize`	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 100GiB 铜牌卷：

```
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. 如果在主机上运行二进制文件本身、则日志可从主机的目录中获取 `/var/log/netappdvp`。要启用调试日志记录、请指定 `-debug`何时运行此插件。

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：

  ```
  Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
  connect: no such file or directory
  ```

  这意味着插件无法启动。幸运的是，该插件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。使用主机操作系统所需的软件包管理器、并检查是否 `rpcbind`正在运行。您可以通过运行或等效的来检查`rpc`绑 定服务的状态 `systemctl status rpcbind`。

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、使用容器化插件中的选项或在主机上实例化 `--volume-driver`Trident时的选项为其指定不同的名称 `--alias`。

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	说明	示例
`version`	配置文件版本号	`1`
`storageDriverName`	存储驱动程序的名称	`ontap-nas`、、 `ontap-san` `ontap-nas-economy`、 `ontap-nas-flexgroup` `solidfire-san`
`storagePrefix`	卷名称的可选前缀。默认值：`netappdvp_`。	`staging_`
`limitVolumeSize`	卷大小的可选限制。默认值：""(未强制实施)	`10g`

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。我们建议使用特定的租户 ID 进行 Docker 卷映射，或者在可能已使用任何名称的情况下使用 Docker 中填充的 Docker 版本，驱动程序信息和原始名称的属性数据。

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。 `size`选项适用于所有控制器类型。有关如何设置默认卷大小的示例，请参见 ONTAP 配置一节。

选项	说明	示例
size	新卷的可选默认大小。默认值： 1G	10G

ONTAP 配置

除了上述全局配置值之外，在使用 ONTAP 时，还可以使用以下顶级选项。

选项	说明	示例
managementLIF	ONTAP 管理 LIF 的 IP 地址。您可以指定完全限定域名（ FQDN ）。	10.0.0.1
dataLIF	协议 LIF 的 IP 地址。 **NAS ONTAP**驱动程序:我们建议指定 dataLIF。如果不提供此参数、则Trident将从SVM提取数据LUN。您可以指定用于NFS挂载操作的完全限定域名(FQDN)、从而可以创建循环DNS、以便在多个数据LIF之间实现负载平衡。 • ONTAP SAN驱动程序*：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。	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
username	用于连接到存储设备的用户名	vsadmin
password	用于连接到存储设备的密码	secret
aggregate	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%

选项	说明	示例
nfsMountOptions	对 NFS 挂载选项进行精细控制；默认为 -o nfsver=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
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 。 *对于 `ontap-nas-economy`驱动程序，此选项允许自定义每个FlexVol的最大qtree*数。	300
sanType	仅支持 **`ontap-san`**驱动程序。***用于为iSCSI、 `nvme`NVMe/TCP或基于光纤通道的 `fcp`SCSI (FC)**选择 **`iscsi`**。"FCP"(基于FC的SCSI)是Trident 24.10版本中的一项技术预览功能。*	`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	说明	示例
spaceReserve	空间预留模式； none(精简配置)或(厚配置) volume	none
snapshotPolicy	要使用的Snapshot策略、默认为 none	none
snapshotReserve	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

选项	说明	示例
encryption	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。	对于NFSv4、为"TRUE"；对于NFSv3、为"false"
exportPolicy	要使用的NFS导出策略的NAS选项、默认为 default	default
securityStyle	用于访问已配置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
tieringPolicy	要使用的分层策略，对于ONTAP 9 5之前的SVM-DR配置，默认为 none； snapshot-only	none

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。对于每个集群节点， ONTAP 最多支持 1000 个 FlexVol ，而集群最多支持 12 ， 000 个 FlexVol 。如果您的Docker卷要求符合此限制、则该 `ontap-nas` 驱动程序将成为首选NAS解决方案、因为它具有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限制的可支持范围、请选择或 ontap-san-economy`驱动程序。 `ontap-nas-economy

此 `ontap-nas-economy`驱动程序会在自动管理的FlexVol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。

Docker Swarm目前不支持此 `ontap-nas-economy`驱动程序、因为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。

有关适用于FlexGroup的FlexGroup和工作负载的信息，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，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>驱动程序的<code> ONTAP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－econom目</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－</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－econom何</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中<code> ONTAP－</code>驱动程序的示例

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

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	说明	示例
`Endpoint`	\https：<login><element-version>：<password>@<mvip>/json-rpC/RPC	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IP 地址和端口	10.0.0.7： 3260
`TenantName`	要使用的 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`
`Types`	QoS 规范	请参见以下示例
`LegacyNamePrefix`	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-`

此 `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. 通过提供额外参数来重新安装插件 `config`。

```
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。

这是 Docker 客户端的限制。客户端会将单个字符的名称解释为Windows路径。 "请参见错误25773"(英文)。

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。

其余驱动程序可能会受到诸如争用情况等问题的影响，这些问题可能会导致为单个请求创建大量卷，而无需明确的 " 赢家 " ；例如， Element 具有一项功能，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

部署

部署Trident时、请遵循此处列出的建议。

部署到专用命名空间

"命名空间"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"通过、管理员可以针对每个命名空间实施全局和特定于存储类的容量和对象计数消耗限制。此外、使用 "范围限制"可确保在将PVC请求转发到配置程序之前、此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。有关的信息，请参见此处 "如何利用配额"。

存储配置

NetApp产品组合中的每个存储平台都具有独特的功能、无论应用程序是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、以便最大程度地提高灵活性应与Trident结合使用。 "SVM-DR"

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。有关您的特定平台和ONTAP版本、请参见 "NetApp Hardware Universe"以确定确切的限制。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 `ontap-nas`和 `ontap-san`驱动程序会为创建的每个Kubernetes持久卷(PV)配置一个灵活卷。 `ontap-nas-economy`驱动程序大约为每200个PV(可在50到300之间配置)创建一个FlexVolume。 `ontap-san-economy`驱动程序大约为每100个PV(可在50到200之间配置)创建一个FlexVolume。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>
```

的值 `max-volumes`会因特定于您的环境的多个条件而异：

- 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

该 `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，一个双节点 ONTAP 集群最多可以托管 2000 个 FlexVolume 。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。 *

限制 **Trident** 创建的卷的最大大小

要为可由Trident创建的卷配置最大大小、请在定义中使用 `limitVolumeSize`参数 `backend.json`。

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 。在后端配置中使用 `useCHAP`参数、Trident通过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 "防止抢占资源" 或失控的容器影响Trident SVM之外的工作负载。

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。

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：

- 您应在后端配置的块中设置 `qosPolicy`密钥 `defaults`。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 9.8 QoS 命令"。

将存储资源访问限制为 **Kubernetes** 集群成员

限制对 Trident 创建的 NFS 卷和 iSCSI LUN 的访问是 Kubernetes 部署安全状况的重要组成部分。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"旨在防止此类损害。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。默认情况下不会拒绝这些功能，因此请确保使用禁用该功能 "POD 安全策略"。

对于需要从 Kubernetes 和外部主机访问的卷，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ONTAP命令行界面命令向导出策略添加规则 `vserver export-policy rule create`。

通过运行这些命令，您可以限制哪些 Kubernetes 节点可以访问数据。

对应用程序**SVM**禁用 `showmount`

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可以对数据LIF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	最小值	默认值	最大值(4KB)
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

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。

块大小和带宽会直接影响 IOPS 数量。随着块大小的增加，系统会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，系统能够达到的 IOPS 数量也会减少。有关QoS和性能的详细信息、请参见 "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。

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。在中搜索 "NetApp 库"最新版本。

- ONTAP *
- "NFS 最佳实践和实施指南"
- "《SAN 管理指南》"(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、请务必与NetApp团队合作、并使用 "NetApp 库"查找最新文档。

集成Trident

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灵活卷。
- `ontap-nas-economy`：配置的每个PV都是一个qtree，每个FlexVolume的qtree数量可配置(默认为200)。
- `ontap-nas-flexgroup`：每个PV配置为完整ONTAP FlexGroup，并使用分配给SVM的所有聚合。
- `ontap-san`：配置的每个PV都是其自身FlexVolume中的一个LUN。
- `ontap-san-economy`：配置的每个PV都是一个LUN，每个FlexVolume的LUN数量可配置(默认值为100)。

在三个 NAS 驱动程序之间进行选择会对应用程序可用的功能产生一些影响。

请注意、在下表中、并非所有功能都通过Trident公开。如果需要某些功能，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	复制
ontap-nas	是	是	是脚注：5[]	是	是脚注：1[]	是	是脚注：1[]
ontap-nas-economy	注：3[]	注：3[]	是脚注：5[]	是	注：3[]	是	注：3[]
ontap-nas-flexgroup	是脚注：1[]	否	是脚注：5[]	是	是脚注：1[]	是	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	复制
ontap-san	是	是	是脚注：4[]	是	是脚注：1[]	是	是脚注：1[]
ontap-san-economy	是	是	是脚注：4[]	是	注：3[]	是	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粒度脚注：3[]：不受Trident管理、而不是PV粒度是脚注：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、和 ontap-nas-economy`之间的许多功能 `ontap-nas`是相同的。但是、由于此 `ontap-nas-economy`驱动程序限制了按PV粒度控制计划的能力、因此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来说，只有在使用、 `ontap-san`或 `ontap-san-economy`驱动程序时才能实现这一点 `ontap-nas。

> 该 `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的兼容驱动程序为 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的兼容驱动程序为 ontap-nas、 ontap-nas-economy、 ontap-nas-flexgroup ontap-san`和 `ontap-san-economy。

NetApp HCI/SolidFire后端驱动程序

`solidfire-san`用于SolidFire平台的驱动程序可帮助管理员根据NetApp HCI限制为Trident配置Element后端。如果要将后端设计为对Trident配置的卷设置特定QoS限制、请使用后端文件中的参数。 `type`管理员还可以使用参数限制可在存储上创建的卷大小`limitVolumeSize`。目前、此驱动程序不支持卷大小调整和卷复制等Element存储功能`solidfire-san`。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	复制
solidfire-san	是	是	是脚注：2[]	是	是	是	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	复制
azure-netapp-files	是	是	是	是	是	是脚注：1[]

> 脚注：是脚注：1[]：不受Trident管理

Google Cloud上的Cloud Volumes Service 后端驱动程序

Trident使用 `gcp-cvs`驱动程序与Google Cloud上的Cloud Volumes Service链接。

该 gcp-cvs`驱动程序使用虚拟池对后端进行抽象化、并允许Trident确定卷放置。管理员在文件中定义虚拟池 `backend.json。存储类使用选择器按标签标识虚拟池。

- 如果在后端定义了虚拟池、则Trident将尝试在Google Cloud存储池中创建一个卷、而这些虚拟池仅限于此类存储池。
- 如果未在后端定义虚拟池、Trident将从该区域的可用存储池中选择Google Cloud存储池。

要在Trident上配置Google Cloud后端，必须在后端文件中指定 projectNumber、 apiRegion`和`apiKey。您可以在Google Cloud控制台中找到项目编号。API密钥来自您在Google Cloud上为Cloud Volumes Service 设置API访问时创建的服务帐户专用密钥文件。

有关Google Cloud上的Cloud Volumes Service服务类型和服务级别的详细信息，请参阅"了解Trident对CVS for GCP的支持"。

适用于**Google Cloud**的**Cloud Volumes Service** 驱动程序	快照	克隆	多连接	**QoS**	展开	复制
`gcp-cvs`	是	是	是	是	是	仅适用于CVS-Performance服务类型。

复制注释

- 复制不受Trident管理。
- 克隆将在与源卷相同的存储池中创建。

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。在存储类中可以设置三组筛选器： `storagePools`、 `additionalStoragePools`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`参数用于明确排除与这些属性匹配的列出的池集。

模拟**QoS**策略

如果要设计存储类以模拟服务质量策略，请创建一个属性为 `hdd`或 `ssd`的存储类 `media`。根据 `media`存储类中提及的属性、Trident将选择与介质属性匹配的适当后端 `hdd`或 `ssd`聚合、然后将卷的配置定向到特定聚合。因此、我们可以创建一个存储类高级、该存储类 `media`的属性设置为可归类为 `ssd`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。管理员无法使用相同的存储凭据和一组不同的参数创建另一个后端。随着虚拟池的推出、此问题描述 得以缓解。虚拟池是在后端和Kubernetes存储类之间引入的级别抽象、因此管理员可以定义参数以及标签、这些参数和标签可以通过Kubernetes存储类作为选择器进行引用、并且与后端无关。可以使用Trident为所有受支持的NetApp后端定义虚拟池。该列表包括 SolidFire/NetApp HCI ， ONTAP ， GCP 上的 Cloud Volumes Service 以及 Azure NetApp Files 。

> 定义虚拟池时、建议不要尝试在后端定义中重新排列现有虚拟池的顺序。此外，建议不要编辑 / 修改现有虚拟池的属性，而是定义新的虚拟池。

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。将Aspect设置 `servicelevel`为适当的性能级别、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。通过 `parameters.selector`字段、每个StorageClass都可以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、使用映射到不同虚拟池的字段创建不同的Kubnetes存储类 `parameters.selector`。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	是	是（原始块）
NFS	是	是	是

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。

目前， Kubernetes 树中配置程序不支持对 NFS 或 iSCSI PV 执行卷大小调整操作。Trident 支持扩展 NFS 和 iSCSI 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和iSCSI PVs的大小。这样，用户就可以直接通过 Kubernetes 层调整其卷的大小。所有主要 NetApp 存储平台均可进行卷扩展，包括 ONTAP ， SolidFire/NetApp HCI 和 Cloud Volumes Service 后端。要允许稍后进行扩展、请在与卷关联的StorageClass中将设置 `allowVolumeExpansion`为 `true`。每当需要调整永久性卷的大小时、都可以将永久性卷声明中的标注编辑 `spec.resources.requests.storage`为所需的卷大小。Trident会自动调整存储集群上卷的大小。

将现有卷导入到 **Kubernetes** 中

通过卷导入，可以将现有存储卷导入到 Kubernetes 环境中。目前、、、 `ontap-nas-flexgroup` `solidfire-san` `azure-netapp-files`和 `gcp-cvs`驱动程序均支持此 `ontap-nas`功能。在将现有应用程序移植到 Kubernetes 或在灾难恢复场景中，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`使用或驱动程序、请使用命令 `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`将卷导入到Trident要管理的Kubornetes `gcp-cvs`中。这样可以确保卷引用是唯一的。

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。

部署 OpenShift 服务

OpenShift 增值集群服务为集群管理员和要托管的应用程序提供了重要功能。这些服务使用的存储可以使用节点本地资源进行配置，但这通常会限制服务的容量，性能，可恢复性和可持续性。利用企业级存储阵列为这些服务提供容量可以显著改善服务，但是，与所有应用程序一样， OpenShift 和存储管理员应密切合作，为每个服务确定最佳选项。应大量利用 Red Hat 文档来确定要求并确保满足规模估算和性能需求。

注册表服务

中介绍了"博客"如何部署和管理注册表的存储"netapp.io"。

日志记录服务

与其他OpenShift服务一样、日志记录服务也是使用Andsight部署的、配置参数由提供给该手册的清单文件(也称为主机)提供。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。

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Ansable变量 设置 `openshift_enable_unsupported_configurations`为 `true`、以防止安装程序失败。

开始使用

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，通过将变量指定 `openshift_logging_use_ops`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。

下表仅包含与日志记录服务相关的存储配置相关的变量。您可以找到其他选项、应根据您的部署查看、配置和使用这些选项"RedHat OpenShift 日志记录文档"。

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`可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数据 LIF 。
`openshift_logging_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷的结合为 `/openshift_logging`，则此变量将使用该路径。
`openshift_logging_storage_volume_name`	要创建的PV的名称，例如 `pv_ose_logs`。
`openshift_logging_storage_volume_size`	NFS导出的大小，例如 `100Gi`。

如果 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_pvc_dynamic`	设置为 true 可使用动态配置的卷。
`openshift_logging_es_pvc_storage_class_name`	要在 PVC 中使用的存储类的名称。
`openshift_logging_es_pvc_size`	在 PVC 中请求的卷大小。
`openshift_logging_es_pvc_prefix`	日志记录服务使用的 PVC 的前缀。
`openshift_logging_es_ops_pvc_dynamic`	设置为 `true`可对操作日志记录实例使用动态配置的卷。
`openshift_logging_es_ops_pvc_storage_class_name`	操作日志记录实例的存储类的名称。
`openshift_logging_es_ops_pvc_size`	操作实例的卷请求大小。
`openshift_logging_es_ops_pvc_prefix`	操作实例 PVC 的前缀。

部署日志记录堆栈

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，如果在初始安装后进行部署，则 Ansible 需要使用组件攻略手册。此过程可能会随OpenShift的不同版本略有不同、因此请务必阅读并遵循"RedHat 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。

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`可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 的数据 LIF 。
`openshift_metrics_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷的结合为 `/openshift_metrics`，则此变量将使用该路径。
`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_pvc_prefix`	用于衡量指标 PVC 的前缀。
`openshift_metrics_cassandra_pvc_size`	要请求的卷的大小。
`openshift_metrics_cassandra_storage_type`	要用于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 。
`openshift_metrics_cassanda_pvc_storage_class_name`	要使用的存储类的名称。

部署指标服务

使用在主机 / 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 OpenShift 的每个版本而发生变化。请务必查看并遵循"RedHat 的 OpenShift 部署指南"您的版本、以便为您的环境配置此版本。

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et结合 使用"Kubenetes：备份etcd集群"。
2. 将备份项目放在FlexVol上。

> 建议您保护FlexVol所在的SVM、并将其与另一个SVM建立SnapMirror关系。

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Kubenetes集群相关的所有必需证书，并在下复制 `/etc/kubernetes/pki`et的 成员文件 `/var/lib/etcd。
3. 使用从et结合 使用"Kubornetes：还原etcd集群"的备份中还原Kubenetes集群。
4. 运行 `kubectl get crd`以验证所有Trident自定义资源是否已启动、并检索Trident对象以验证所有数据是否可用。

SVM复制和恢复

Trident无法配置复制关系、但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"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`、则在配置SVM复制时、请更新 `managementLIF`Trident后端定义文件中的和 `dataLIF`。
3. 确认 `storagePrefix`存在于Trident后端定义文件中。无法更改此参数。如果不执行此操作 `storagePrefix`、则会导致后端更新失败。
4. 使用以下命令更新所有必需的后端、以反映新的目标SVM名称：

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. 如果指定了 `-identity-preserve false`或 `discard-config network`，则必须退回所有应用程序Pod。

> 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"。

> `ontap-san-economy`或 `ontap-flexgroup-economy`驱动程序不支持导入 `ontap-nas-economy`。

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"

安全性

安全性

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。

> 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。
- 如果未在后端启用NAE、则在Trident中配置的任何卷都将启用NVE、除非您在后端配置中将NVE加密标志设置为 `false`。

> 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"

Linux统一密钥设置(LUKS)

您可以启用Linux统一密钥设置(Unified Key Setup、LKS)来对Trident上的ONTAP SAN和ONTAP SAN经济型卷进行加密。Trident支持对经过LUN加密的卷进行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"。

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、我们建议员工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后端配置选项的详细信息，请参阅"ONTAP SAN配置选项"。

```
"storage": [
    {
        "labels":{"luks": "true"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "true"
        }
    },
    {
        "labels":{"luks": "false"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "false"
        }
    },
]
```

2. `parameters.selector`用于定义使用了一个使用了此加密的存储池。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. 创建一个包含LUKS密码短语的密钥。例如：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

限制

LUKS加密的卷无法利用ONTAP 重复数据删除和数据压缩功能。

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。

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、 `previous-luks-passphrase`则会忽略参数。

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。

   卷

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

   Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. 更新卷的LUKS密钥以指定新密码短语和上一密码短语。确保 `previous-luke-passphrase-name` `previous-luks-passphrase`与先前的密码短语匹配。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. 创建一个新的装载卷的POD。这是启动轮换所必需的。
5. 验证密码短语是否已轮换。

卷

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

结果

仅在卷和快照上返回新密码短语时、才会轮换密码短语。

> 如果返回两个密码，例如 `luksPassphraseNames: ["B", "A"]`，则旋转不完整。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`特征门(beta 1.25+)。有关详细信息、请参见 "Kubernetes 1.25：使用机密进行节点驱动型CSI卷扩展" 。
2. 添加 `node-expand-secret-name`和 `node-expand-secret-namespace`StorageClass参数。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

结果

启动联机存储扩展时、kubelet会将相应的凭据传递给驱动程序。

Kerberos传输中加密

通过使用Kerberos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：

- 内部部署**Kubernet**—Trident支持通过从ONTAP OpenShift和上游Kubernetes集群到内部部署ONTAP卷的NFS3和NFSv4连接进行Kerberos加密。

您可以创建、删除、调整大小、创建快照、克隆、 只读克隆、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。

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 "在数据 LIF 上启用 Kerberos" 。
- 确保已正确配置使用Kerberos加密的任何NFSv4卷。请参阅的“NetApp NFSv4域配置”一节(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：

键入	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"。
- 确保已正确配置使用Kerberos加密的任何NFSv4卷。请参阅的“NetApp NFSv4域配置”一节(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>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

虚拟池级别示例

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

2. 使用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试用者
- VMware Tanzu产品组合
- 上游Kubernetes

确保安装Trident Protect 的集群已配置运行中的快照控制器和相关的 CRD。要安装快照控制器，请参阅 "这些说明"。

Trident Protect 存储后端兼容性

Trident Protect 支持以下存储后端：

- 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存储使用的是Astra Trident 24.02或更高版本(建议使用Trident 24.10)。
  - 如果Astra Trident的版本低于24.06.1、而您计划使用NetApp SnapMirror灾难恢复功能、则需要手动启用Astra Control配置程序。
- 确保已安装最新的Astra控件配置程序(从Astra Trident 24.06.1开始、默认情况下已安装并启用)。
- 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。

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
```

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 24.10 和 24.10.1 及更高版本在保护运行在 KubeVirt VM 上的应用程序时，行为有所不同。对于这两个版本，您都可以在数据保护操作期间启用或禁用文件系统冻结和解冻结。

对于所有Trident Protect 版本，要在 OpenShift 环境中启用或禁用自动冻结功能，您可能需要授予应用程序命名空间特权权限。例如：

```
oc adm policy add-scc-to-user privileged -z default -n
<application-namespace>
```

Trident Protect 24.10

Trident Protect 24.10 在数据保护操作期间不会自动确保 KubeVirt VM 文件系统的一致性状态。如果您想使用Trident Protect 24.10 保护您的 KubeVirt VM 数据，则需要在执行数据保护操作之前手动启用文件系统的冻结/解冻功能。这样可以确保文件系统处于一致状态。

您可以配置Trident Protect 24.10 来管理数据保护操作期间 VM 文件系统的冻结和解冻。"正在配置虚拟化"然后使用以下命令：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

Trident Protect 24.10.1 及更高版本

从 Trident Protect 24.10.1 开始，Trident Protect 在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。或者，您可以使用以下命令禁用此自动行为：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

SnapMirror复制的要求

NetApp SnapMirror可与Trident Protect 配合使用，适用于以下ONTAP解决方案：

- NetApp ASA
- NetApp AFF
- NetApp FAS
- 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：

- * Astra Control Provisioner 或Trident *：使用ONTAP作为后端的源 Kubernetes 集群和目标 Kubernetes 集群上必须同时存在Astra Control Provisioner 或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`

- ontap-san

- 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" 。

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" 。

  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。

- **Astra**控件配置程序或**Trident**和**SVM**：对等远程SVM必须可供目标集群上的Astra控件配置程序或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。
- **NVMe over TCP**： Trident Protect 不支持使用 NVMe over TCP 协议的存储后端的NetApp SnapMirror复制。

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。

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。

默认情况下， Trident Protect 会收集有助于处理您可能提交的任何NetApp支持案例的支持信息，包括集群和受管应用程序的日志、指标和拓扑信息。Trident Protect 会按计划每日向NetApp发送这些支持包。安装Trident Protect 时，您可以选择禁用此支持包集合。您可以手动操作"生成支持包"随时。

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. 安装Trident Protect CRD：

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

3. 使用 Helm 通过以下命令之一安装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：

   - 正常安装Trident Protect：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.1
--create-namespace --namespace trident-protect
```

   - 安装Trident Protect 并禁用每日定期上传的Trident Protect AutoSupport支持包：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect
```

从私有注册表安装Trident Protect

如果您的 Kubernetes 集群无法访问互联网，您可以从私有镜像仓库安装Trident Protect。在这些示例中，请将括号中的值替换为您环境中的信息：

步骤

1. 将以下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：

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exechook:24.10.1
netapp/resourcebackup:24.10.1
netapp/resourcerestore:24.10.1
netapp/resourcedelete:24.10.1
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例如：

```
docker pull netapp/controller:24.10.1
```

```
docker tag netapp/controller:24.10.1 <private-registry-
url>/controller:24.10.1
```

```
docker push <private-registry-url>/controller:24.10.1
```

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：

```
---
image:
  registry: <private-registry-url>
imagePullSecrets:
  - name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
    - name: regcred
webhooksCleanup:
  imagePullSecrets:
    - name: regcred
```

7. 安装Trident Protect CRD：

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

8. 使用 Helm 通过以下命令之一安装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：

   - 正常安装Trident Protect：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.1
--create-namespace --namespace trident-protect -f
protectValues.yaml
```

   - 安装Trident Protect 并禁用每日定期上传的Trident Protect AutoSupport支持包：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

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vaule：

```
---
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 -f <resourceLimits.yaml>
--reuse-values
```

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/24.10.1/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/24.10.1/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/24.10.1/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/24.10.1/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/
```

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. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/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. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/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。

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)。

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。

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. 使用正确的值填充文件后 `astra-support-bundle.yaml` 、应用CR：

```
kubectl apply -f trident-protect-support-bundle.yaml
```

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>
```

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。

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：

```
helm upgrade trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1  --namespace trident-protect
```

3. 升级Trident保护：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2410.1 --namespace trident-protect
```

管理和保护应用程序

使用**Trident Protect AppVault** 对象来管理存储桶。

Trident Protect 的存储桶自定义资源 (CR) 被称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含存储桶在保护操作（例如备份、快照、恢复操作和SnapMirror复制）中使用的必要配置。只有管理员才能创建应用保险库。

密钥生成和**AppVault**定义示例

定义AppVault CR时、您需要包含凭据才能访问由提供程序托管的资源。根据提供程序的不同、为凭据生成密钥的方式也会有所不同。以下是多个提供程序的命令行密钥生成示例、然后是每个提供程序的AppVault定义示例。

密钥生成示例

您可以使用以下示例为每个云提供商的凭据创建密钥。

Google Cloud

```
kubectl create secret generic <secret-name> --from-file=credentials
=<mycreds-file.json> -n trident-protect
```

Amazon S3 (AWS)

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-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=<generic-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
=<generic-s3-trident-protect-src-bucket-secret> -n trident-protect
```

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。

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  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-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  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-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

StorageGRID S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-
971f-ac4a83621922
  namespace: trident-protect
spec:
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

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。

Google Cloud

```
tridentctl-protect create vault GCP my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3 (AWS)

```
tridentctl-protect create vault AWS <vault-name> --bucket <bucket-name>
--secret  <secret-name>  --endpoint <s3-endpoint>
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> --account <account-
name> --bucket <bucket-name> --secret <secret-name>
```

通用 S3

```
tridentctl-protect create vault GenericS3 <vault-name> --bucket
<bucket-name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

StorageGRID S3

```
tridentctl-protect create vault StorageGridS3 s3vault --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

使用AppVault浏览器查看AppVault信息

您可以使用Trident Protect CLI 插件查看有关集群上创建的 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
   ```

   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。

> 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

确保已删除存储在关联存储分段中的所有快照和备份。

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
```

使用**Trident Protect** 定义管理应用程序

您可以通过创建应用程序 CR 和关联的 AppVault CR 来定义要使用Trident Protect 管理的应用程序。

创建**AppVault CR**

您需要创建一个 AppVault CR，该 CR 将在对应用程序执行数据保护操作时使用，并且 AppVault CR 需要位于安装了Trident Protect 的集群上。AppVault CR 是针对您的特定环境的；有关 AppVault CR 的示例，请参阅："AppVault自定义资源。"

定义应用程序

您需要定义要使用Trident Protect 管理的每个应用程序。您可以通过手动创建应用程序 CR 或使用Trident Protect CLI 来定义要管理的应用程序。

使用**CR**添加应用程序

步骤

1. 创建目标应用程序CR文件：

   a. 创建自定义资源(CR)文件并将其命名(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_*)使用命名空间标签或命名空间名称来指定应用程序资源所在的命名空间。应用程序命名空间必须属于此列表。

        YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: maria
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
```

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
```

使用命令行界面添加应用程序

步骤

1. 创建并应用应用应用程序定义、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以下示例中所示的参数在应用程序定义中包括名称和资源：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <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。

您可以配置Trident Protect 在数据保护操作期间冻结和解冻文件系统。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

创建按需快照

您可以随时创建按需快照。

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。

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)：要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。

```
---
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> -n
<application_namespace>
```

创建数据保护计划

保护策略通过按定义的计划创建快照，备份或这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，并且可以指定要保留的副本数。

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 *：要保留的备份数。零表示不应创建任何备份。
   - *spec.snapshotretention *：要保留的快照数。零表示不应创建任何快照。
   - 。**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`)
   - **spec.dayOfMonth**：(可 选)计划应运行的日期(1 - 31)。如果粒度设置为，则需要此字段 `monthly`。
   - **spec.dayOfWeek**：(可 选)计划应运行的日期(0到7)。值0或7表示星期日。如果粒度设置为，则需要此字段 `weekly`。
   - *spec.hour *：(可 选)计划应运行的时间(0 - 23)。如果粒度设置为、或，则需要此字段 `daily` `weekly` `monthly`。
   - *spec.minute：(可 选)计划应运行的分钟(0 - 59)。如果粒度设置为、、或，则需要此字段 `hourly` `daily` `weekly` `monthly`。

```
---
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: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"
```

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：

> 您可以使用 `tridentctl-protect create schedule --help`查看此命令的详细帮助信息。

```
tridentctl-protect create schedule <my_schedule_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> --backup
-retention <how_many_backups_to_retain> --data-mover
<kopia_or_restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain> -n <application_namespace>
```

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。

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。

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。

> 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。

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。

> 如果您使用RedHat OpenShift、请务必注意命名空间标注在OpenShift环境中的关键作用。命名空间标注可确保还原的Pod遵循OpenShift安全上下文约束(SCC)定义的适当权限和安全配置、并可在没有权限问题的情况下访问卷。有关详细信息，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：

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

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

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。

> 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

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.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。
   - **spec.storageClassMapping**：还原操作的源存储类到目标存储类的映射。将和 `sourceStorageClass`替换 `destinationStorageClass`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
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

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：
   - **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CLI

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> -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。

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：
   - **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**CLI**

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。

- 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。

步骤

1. 使用Trident Protect CLI 插件检查目标集群上 AppVault CR 的可用性：

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

确保目标集群上存在用于应用程序还原的命名空间。

2. 从目标集群查看可用AppVault的备份内容：

```
tridentctl-protect get appvaultcontent <appvault_name> --show-resources
backup --show-paths --context <destination_cluster_name>
```

运行此命令可显示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：

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**：AppVault中存储备份内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     如果BackupRestore CR不可用、您可以使用步骤2中提到的命令查看备份内容。

   - 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。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
  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**CLI**

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> -n <application_namespace>
--context <destination_cluster_name>
```

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。

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.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。
   - **spec.storageClassMapping**：还原操作的源存储类到目标存储类的映射。将和 `sourceStorageClass`替换 `destinationStorageClass`为您环境中的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

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：
   - **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CLI

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> -n <application_namespace>
```

从快照还原到原始命名空间

您可以随时将快照还原到原始命名空间。

如果您的应用程序使用多个命名空间，并且这些命名空间具有同名的PVC，则快照还原（到原始命名空间和新命名空间）将无法正常工作。所有还原的卷将具有相同的数据，而不是每个卷的正确数据。使用备份还原而不是快照还原，或升级到修复此问题的26.02版或更高版

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
```

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：
   - **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-ipr-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

使用CLI

步骤

1. 将快照还原到原始命名空间、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 <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NetApp SnapMirror和Trident Protect 复制应用程序

使用Trident Protect，您可以利用NetApp SnapMirror技术的异步复制功能，将数据和应用程序更改从一个存储后端复制到另一个存储后端，无论是在同一集群内还是在不同集群之间。

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。

> 如果您使用RedHat OpenShift、请务必注意命名空间标注在OpenShift环境中的关键作用。命名空间标注可确保还原的Pod遵循OpenShift安全上下文约束(SCC)定义的适当权限和安全配置、并可在没有权限问题的情况下访问卷。有关详细信息，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：

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

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

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"。

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>
```

2. 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)为源应用程序创建快照。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、则需要等待运行下一个计划快照、以便获得最新快照。

使用**CR**创建快照

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**：(*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的metadata.name字段匹配。
- **spec.backup**䜣 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**䜣 刭：必须设置为2。

YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-
46a3-420a-b351-45795e1b5e34
  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
```

使用命令行界面创建快照

a. 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>
```

4. 在目标集群上创建一个与源集群上应用的AppVault CR相同的源应用程序AppVault CR，并将其命名为(例如 `trident-protect-appvault-primary-destination.yaml`)。

5. 应用CR：

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
my-app-namespace
```

6. 在目标集群上为目标应用程序创建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：

   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 my-app-namespace
```

7. 创建App镜像 关系CR文件：

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。
- 。**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.storageClassName**：(*required*)选择集群上有效存储类的名称。存储类必须链接到与源环境建立对等关系的ONTAP Storage VM。
- **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App镜像 关系对象、将括号中的值替换为环境中的信息。例如：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --recurrence-rule <rule> --source-app
<my_source_app> --source-app-vault <my_source_app_vault> -n
<application_namespace>
```

8. (可 选)检查复制关系的状态：

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

故障转移到目标集群

使用Trident Protect，您可以将复制的应用程序故障转移到目标集群。此过程会停止复制关系，并将应用程序在目标集群上联机。如果源集群上的应用程序正在运行， Trident Protect 不会停止该应用程序。

步骤

1. 打开AppMirectorRelationship CR文件(例如 `trident-protect-relationship.yaml`)，并将*。spec.desiredState*的值更改为 `Promoted`。
2. 保存 CR 文件。
3. 应用CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. (可 选)在故障转移应用程序上创建所需的任何保护计划。
5. (可 选)检查复制关系的状态：

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

重新同步故障转移复制关系

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。

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. 创建源应用程序的快照。
2. 打开AppMirectorRelationship 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. 删除初始目标集群上的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. 确保每个集群上的AppVault CRS均已准备就绪。
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. 创建关闭快照：

使用**CR**创建关闭快照

a. 禁用源应用程序的保护策略计划。

b. 创建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. 快照完成后、获取快照的状态：

```
kubectl get shutdownsnapshot -n my-app-namespace
<shutdown_snapshot_name> -o yaml
```

3. 使用以下命令查找*shutdownSnapshot．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后面的所有内容)：

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 执行从目标集群到源集群的故障转移、并进行以下更改：

> 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。

> 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. 删除App镜像 关系CR：

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

使用Trident Protect 迁移应用程序

您可以通过将备份或快照数据还原到其他集群或存储类来在集群或存储类之间迁移应用程序。

> 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。

> 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

您可以通过将快照还原到不同的目标存储类来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>
```

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

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。

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。

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。

> 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"。

> 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。

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CLI

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>
```

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：

```
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
```

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?

可以、Trident可以脱机安装。请参阅 "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**?

通过使用三端技术支持安装程序、可以创建一个Kubernetes ClusterRole、此类ClusterRole可以对Kubernetes集群中的集群的PersentVolume、PersentVolumeClaim、StorageClass和机密资源具有特定访问权限。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 。

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。

Trident是否支持**Cloud Volumes**服务?

是的、Trident支持Azure中的Azure NetApp Files服务以及GCP中的Cloud Volumes Service。

故障排除和支持

NetApp是否支持**Trident**?

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. 请联系以提交支持案例 "NetApp 支持"。

如何生成支持日志包?

您可以通过运行来创建支持包 `tridentctl logs -a`。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`。将示例中的替换 `myBackend`为您的后端名称以及 `/path/to_new_backend.json`正确文件的路径 `backend.json`。

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。数据LIF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：建议指定 `dataLIF`。如果不提供此参数、则Trident将从SVM提取数据LUN。您可以指定用于NFS挂载操作的完全限定域名(FQDN)、从而可以创建循环DNS、以便在多个数据LIF之间实现负载平衡。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数据 **LIF** ？

Trident支持为以下项定义IPv6地址：

- `managementLIF`和 `dataLIF`ONTAP NAS后端。
- `managementLIF`适用于ONTAP SAN后端。您不能在ONTAP SAN后端指定 `dataLIF`。

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、可以使用命令更新后端管理LIF `tridentctl update backend`。

是否可以更新后端的数据 **LIF** ？

您只能在和 `ontap-nas-economy`上更新数据LIF `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类选择正确的池、则会使用和 `additionalStoragePools`参数选择一组特定的池 `storagePools`。

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。

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。如果不存在可为 qtree 提供服务的 FlexVol ，则会创建一个新的 FlexVol 。

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 `exportPolicy`后端定义文件中配置的参数。

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?

`StorageClasses`用于为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. 在上 StorageClass，将属性设置 `provisioningType`为thick。
2. 在后端定义文件中、通过将设置为volume来启用厚卷 backend spaceReserve parameter。

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?

是。您可以扩展由Trident创建的PVC。请注意，卷自动增长是一项 ONTAP 功能，不适用于 Trident 。

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?

截至目前，我们的、、 ontap-nas-flexgroup ontap-san、、、 ontap-san-economy solidfire-san`均可提供按需快照支持 `ontap-nas。 `gcp-cvs`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?

可在、 ontap-san`和 `ontap-nas-flexgroup`驱动程序上找到。 `ontap-nas`您还可以在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参数来访问Trident配置的卷上的Snapshot目录 `snapshotDir`。

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。

常规故障排除

- 如果Trident POD无法正常启动(例如、当Trident POD在少于两个可用容器的阶段停留时 `ContainerCreating`)、运行和 `kubectl -n trident describe pod trident-**`可以提供更多见解。 `kubectl -n trident describe deployment trident`获取`kubelet`日志(例如，通过 `journalctl -xeu kubelet`)也会很有帮助。
- 如果Trident日志中没有足够的信息、您可以根据安装选项将标志传递到install参数、以尝试启用Trident的调试模式 `-d`。

  然后、使用确认已设置调试 `./tridentctl logs -n trident`、并在日志中搜索 `level=debug msg`。

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。您可以通过观察输出中的“年龄”列来检查此 `kubectl get pod -n trident`情况。

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RedHat CoreTM OS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。
- 将Trident与结合使用时、您可能会遇到一个常见问题 "Azure NetApp Files"、即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、请参见"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。使用主机操作系统所需的软件包管理器、并检查是否 `rpcbind`正在运行。您可以通过运行或等效的来检查服务的 `systemctl status rpcbind`状态 `rpcbind`。
- 如果Trident后端报告它在以前工作过的情况下仍处于 `failed`状态、则可能是由于更改了与后端关联的SVM/admin凭据而导致的。使用或恢复Trident POD更新后端信息 `tridentctl update backend`可解决此问题。
- 如果在使用Docker作为容器运行时安装Trident时遇到权限问题、请尝试使用标志安装Trident `--in cluster=false`。这不会使用安装程序POD、并避免因用户而出现权限问题 `trident-installer`。
- 使用 `uninstall parameter <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Trident、请先运行 `tridentctl uninstall`命令以删除Trident。下载所需 "Trident 版本"、然后使用命令进行安装 `tridentctl install`。
- 成功安装后、如果PVC滞留在此阶段、则 `Pending`运行 `kubectl describe pvc`可提供有关Trident为何无法为此PVC配置PV的更多信息。

使用操作员无法成功部署TRIdent

如果使用操作员部署Trident，则状态 `TridentOrchestrator`将从更 `Installing`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、您应查看状态。 `TridentOrchestrator`

```
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Kubnetes集群只能有一个Trident实例、因此操作员可确保在任何给定时间只存在一个可创建的活动实例 `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`，然后使用修改后的准确定义创建一个新的。

使用部署Trident失败 `tridentctl`

为了帮助您确定出现了什么问题、您可以使用参数再次运行安装程序`-d`、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。

> 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></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
将添加 `hostNQN`回子系统。

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。

版本	完全支持	支持有限	自助支持
"24.10"	2025年10月	2026年10月	2027年10月
"24.06"	2025年6月	2026年6月	2027年6月
"24.02"	—	2026年2月	2027年2月
"23.10"	—	2025年10月	2026年10月

版本	完全支持	支持有限	自助支持
"23.07"	—	2025年7月	2026年7月
"23.04"	—	2025年4月	2026年4月
"23.01"	—	—	2026年1月
"22.10"	—	—	2025年10月
"22.07"	—	—	2025年7月
"22.04"	—	—	2025年4月

自助支持

有关故障排除文章的完整列表，请参阅 "NetApp 知识库（需要登录）"。

社区支持

我们的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>`。

了解更多信息

- "Trident资源"
- "Kubernetes Hub"

参考

Trident端口

详细了解Trident用于通信的端口。

Trident端口

Trident通过以下端口进行通信：

端口	目的
8443	后通道 HTTPS
8001	Prometheus 指标端点
8000	Trident REST 服务器
17546	Trident demonset Pod 使用的活动性 / 就绪性探测端口

可以在安装期间使用标志更改活动性/就绪探测端口 `--probe-port`。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"这是与Trident REST API交互的最简单方式、但您也可以根据需要直接使用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。

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

使用此选项或启用Kubornetes `-k8s_api_server`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。 `-address ""`用于使REST接口可从POD IP地址访问。

> 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. 用户创建了 `PersistentVolumeClaim`、请求从管理员先前配置的KubureNet获取特定大小的 `StorageClass`新 `PersistentVolume`。
2. Kubernetes `StorageClass`将Trident标识为其配置程序、并包含一些参数、用于告知Trident如何为请求的类配置卷。
3. Trident使用相同的名称查找自己的 `StorageClass`卷、该名称用于标识匹配项 `Backends`、并`StoragePools`可用于为类配置卷。
4. Trident会在匹配的后端配置存储并创建两个对象：一个 `PersistentVolume`位于KubeNet中、用于告知KubeNet如何查找、挂载和处理卷；另一个位于Trident中、用于保留与实际存储之间的关系 `PersistentVolume`。
5. Kubnetes会将绑定 `PersistentVolumeClaim`到新 `PersistentVolume`。包含在运行此持久卷的任何主机上挂载此持久卷的Pod `PersistentVolumeClaim`。
6. 用户使用指向Trident的创建 `VolumeSnapshot`现有PVC的 `VolumeSnapshotClass`。

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`用于指示Kubbernetes如何识别快照。
8. 用户可以使用 `VolumeSnapshot`创建 `PersistentVolumeClaim`作为源。
9. Trident会确定所需的快照，并执行与创建和 `Volume`相同的一组步骤 `PersistentVolume`。

要进一步阅读有关Kubbernetes对象的信息、我们强烈建议您阅读 "永久性卷"Kubbernetes文档的章节。

Kubbernetes `PersistentVolumeClaim`对象

Kubbernetes `PersistentVolumeClaim`对象是由Kubbernetes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 ， gcp-cvs ， ontap-san-economy.
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 ， GCP-CVS
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

如果创建的PV具有 `Delete`回收策略、则在PV释放后(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使用此 `Retain`策略、则Trident会忽略此策略、并假定管理员将从Kubbernetes和后端对其进行清理、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将其删除(`tridentctl`)。

Trident 支持使用 CSI 规范创建卷快照：您可以创建卷快照并将其用作数据源来克隆现有 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 。查看 `On-Demand Volume Snapshots`以了解其工作原理。

Trident还提供了 `cloneFromPVC`和 `splitOnClone`标注以用于创建克隆。您可以使用这些标注克隆PVC、而无需使用CSI实施。

以下是一个示例：如果用户已经有一个名为的 `mysql`PVC，则用户可以使用标注创建一个名为的新`PVC `mysqlclone，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：

- 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。
- 对于 `ontap-nas`和 `ontap-san`驱动程序，可能需要将`PVC`标注 `trident.netapp.io/splitOnClone`与结合使用 `trident.netapp.io/cloneFromPVC`。将设置为 `true`时 `trident.netapp.io/splitOnClone`，Trident会将克隆的卷从父卷中分离出来，从而使克隆卷的生命周期与其父卷完全分离，从而牺牲一些存储效率。如果不将其设置 `trident.netapp.io/splitOnClone`或设置为 `false`、则会减少后端的空间消耗、而这会影响在父卷和克隆卷之间创建依赖关系、从而导致无法删除父卷、除非先删除克隆。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。

该 `sample-input`目录包含用于Trident的PVC定义示例。有关与Trident卷关联的参数和设置的完整说明、请参见。

Kubbernetes `PersistentVolume`对象

Kubbernetes对象表示可供Kubbernetes `PersistentVolume`集群使用的一段存储。它的生命周期与使用它的 POD 无关。

> Trident会根据所配置的卷自动创建 `PersistentVolume`对象并将其注册到Kubbernetes集群中。您不应自行管理它们。

创建引用基于Trident的PVC时 `StorageClass`，Trident会使用相应的存储类配置新卷，并为该卷注册新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。

Kubbernetes `StorageClass`对象

Kubnetes `StorageClass`对象在中按名称指定 `PersistentVolumeClaims`、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Kubenetes `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	键入	必填	说明
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
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	键入	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

属性	键入	值	优惠	请求	支持
后端类型	string	ontap-nas 、ontap-nas-economy. ontap-nas-flexgroup 、ontap-san 、solidfire-san 、GCP-CVS 、azure-netapp-files、ontap-san-economy.	池属于此类型的后端	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 ， gcp-cvs
克隆	池	true false	Pool 支持克隆卷	启用了克隆的卷	ontap-nas ， ontap-san ， solidfire-san ， gcp-cvs
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`来模拟满足特定类需求所需的存储质量。`Trident`会自动发现并选择与您指定的`_all_`匹配的存储池 `attributes`。

如果您发现自己无法使用 `attributes`自动为类选择合适的池、则可以使用和 `additionalStoragePools`参数进一步细化池、甚至可以 `storagePools`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匹配的池集 `attributes`。换言之、Trident使用和 `storagePools`参数标识的池的交叉点 `attributes`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 `additionalStoragePools`参数扩展`Trident`用于配置的池集、而不管和 `storagePools`参数选择了哪些池 `attributes`。

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。

在和 `additionalStoragePools`参数中 `storagePools`，每个条目的格式为

`<backend>:<storagePoolList>`，其中 `<storagePoolList>`是指定后端的存储池的逗号分隔列表。例如，的值 `additionalStoragePools`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`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	键入	值	说明	相关驱动程序	**Kubernetes** 版本
FSType	string	ext4、ext3、xfs	块卷的文件系统类型	solidfire-san 、ontap-nas 、ontap-nas-economy. ontap-nas-flexgroup 、ontap-san 、ontap-san-economy.	全部
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 ，gcp-cvs ，azure-netapp-files	1.11多个
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

- `fsType`参数用于控制所需的SAN LUN文件系统类型。此外、Kubnetes还会使用存储类中存在的来指示文件系统存在 `fsType`。只有在设置了后、才能使用POD的安全上下文 `fsType`控制卷所有权 `fsGroup`。有关使用上下文设置卷所有权的概述、 `fsGroup`请参见"Kubernetes： 为 Pod 或容器配置安全上下文"。只有在以下情况下、Kubnetes才会应用此 `fsGroup`值：

  - `fsType`在存储类中设置。
  - 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`。您可以将其设置为或任何非空值。 `nfs`

- 有关卷扩展的详细信息、请参见"展开卷"。
- Trident安装程序包提供了几个示例存储类定义`sample-input/storage-class-*.yaml`，用于中的Trident。删除 Kubernetes 存储类也会删除相应的 Trident 存储类。

Kubbernetes `VolumeSnapshotClass`对象

Kubbernetes `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。

`VolumeSnapshotClass`要创建快照、管理员应定义。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`用于向Kub联网 指定由Trident处理对类的卷快照的请求 `csi-snapclass`。`deletionPolicy`指定在必须删除快照时要执行的操作。如果 `deletionPolicy`将设置为`Delete`，则在删除快照后，系统将删除卷快照对象以及存储集群上的底层快照。或者、将其设置为`Retain`表示将 `VolumeSnapshotContent`保留和物理快照。

Kubbernetes `VolumeSnapshot`对象

Kubnetes `VolumeSnapshot`对象是指创建卷快照的请求。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、并通过创建唯一对象来公开快照VolumeSnapshotContent。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 。

> VolumeSnapshot 的生命周期与源 PVC 无关：即使删除了源 PVC ，快照也会持续存在。删除具有关联快照的 PVC 时， Trident 会将此 PVC 的后备卷标记为 " 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。

Kubbernetes `VolumeSnapshotContent`对象

Kubbernetes `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。

`VolumeSnapshotContent`对象包含唯一标识快照的详细信息，例如 `snapshotHandle`。这`snapshotHandle`是PV名称和对象名称的唯一组合 `VolumeSnapshotContent`。

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、从而将快照公开给Kubnetes API。

> 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 `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。您可以通过运行来获取这些资源定义 `kubectl get crds`。

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。简单地说，使用创建后端时 `tridentctl`，会创建一个相应的 `tridentbackends`CRD对象供Kubbernetes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。
- 使用命令卸载Trident时 `tridentctl uninstall`、Trident Pod将被删除、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会为在其配置程序字段中指定的Kubbernetes对象 `csi.trident.netapp.io`创建匹配的存储类 `StorageClass`。存储类名称与它所代表的Kubbernetes对象的名称匹配 `StorageClass`。

> 使用Kubnetes时、将在注册使用Trident作为配置程序的Kubnetes时自动创建这些对象 `StorageClass`。

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KubeNet部署、我们希望在注册新的KubeNet对象时创建这些 `StorageClass`对象。

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。

> 这是您自己创建和管理的两种对象类型之一。另一个是Kubbernetes `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

存储池表示可在每个后端配置的不同位置。对于 ONTAP ，这些聚合对应于 SVM 中的聚合。对于 NetApp HCI/SolidFire ，这些 QoS 分段对应于管理员指定的 QoS 分段。对于 Cloud Volumes Service ，这些区域对应于云提供商区域。每个存储池都有一组不同的存储属性，用于定义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，由后端端点组成，例如 NFS 共享和 iSCSI LUN 。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。

- 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	键入	必填	说明
version	string	否	Trident API 版本（ "1" ）
name	string	是	要创建的卷的名称
存储类	string	是	配置卷时要使用的存储类
大小	string	是	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
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

属性	键入	必填	说明
块大小	string	否	SolidFire — *： 块 / 扇区大小
文件系统	string	否	文件系统类型

Trident会在创建卷时生成 `internalName`。这包括两个步骤。首先，它会在卷名称前面附加存储前缀(默认 `trident`前缀或后端配置中的前缀)，从而生成格式为的名称 `<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。

您可以使用卷配置直接使用REST API配置卷、但在Kubbernetes部署中、我们希望大多数用户使用标准Kubbernetes `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Kubnetes中、这些直接对应于 `VolumeSnapshotContent`对象。每个快照都与一个卷相关联，该卷是快照的数据源。

每个 `Snapshot`对象都包括下列属性：

属性	键入	必填	说明
version	字符串	是	Trident API 版本（ "1" ）
name	字符串	是	Trident Snapshot 对象的名称
内部名称	字符串	是	存储系统上 Trident Snapshot 对象的名称
volumeName	字符串	是	为其创建快照的永久性卷的名称
volumeInternalName	字符串	是	存储系统上关联的 Trident 卷对象的名称

在 Kubernetes 中，这些对象会自动进行管理。您可以查看它们以查看 Trident 配置的内容。

创建Kubnetes `VolumeSnapshot`对象请求后、`Trident`会通过在后备存储系统上创建`Snapshot`对象来工作。此快照对象的是通过将前缀与 `UID`该对象的 `VolumeSnapshot`组合来生成 `snapshot-`的 `internalName`(例如 `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`)。 `volumeName`和 `volumeInternalName`将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别进行更多控制、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"。

如果安装失败、请进行清理 `ResourceQuota`

在创建对象后安装失败的极少数情况下 `ResourceQuota`、请先尝试、"正在卸载"然后再重新安装。

如果不起作用、请手动删除该 `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。

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，请参阅"Kubernetes：安全性"。

所需的Kubernetes安全上下文和相关字段

权限	说明
特权	CSI要求挂载点为双向挂载点、这意味着Trident节点POD必须运行特权容器。有关详细信息，请参阅"Kubernetes：挂载传播"。
主机网络连接	对于iSCSI守护进程为必需项。 `iscsiadm`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	此 `SYS_ADMIN`功能是作为有权限的容器的默认功能的一部分提供的。例如、`Docker`可为有权限的容器设置以下功能：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

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)

标签	说明	默认
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	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 enforce-version: <version of the current cluster or highest version of PSS tested.>`

更改命名空间标签可能会导致Pod未计划、出现"创建时出错：..."或"警告：Trident CSI -..."。如果发生这种情况、请检查的命名空间标签是否 `privileged`已更改。如果是、请重新安装Trident。

POD安全策略(PSP)

字段	说明	默认
`allowPrivilegeEscalation`	有权限的容器必须允许权限升级。	`true`

字段	说明	默认
allowedCSIDrivers	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
allowedFlexVolumes	Trident不使用"FlexVolume驱动程序"，因此它们不包括在允许的卷列表中。	空
allowedHostPaths	Trident节点POD挂载节点的根文件系统、因此设置此列表没有好处。	空
allowedProcMountTypes	Trident不使用任何 ProcMountTypes。	空
allowedUnsafeSysctls	Trident不需要任何不安全的 sysctls。	空
defaultAddCapabilities	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	`sysctls`不允许。	空
fsGroup	Trident容器以root身份运行。	RunAsAny
hostIPC	挂载NFS卷需要与主机IPC进行通信 nfsd	true
hostNetwork	iscsiadm要求主机网络与iSCSI守护进程进行通信。	true
hostPID	需要主机PID来检查节点上是否`rpc-statd`正在运行。	true
hostPorts	Trident不使用任何主机端口。	空
privileged	Trident节点Pod必须运行特权容器才能挂载卷。	true
readOnlyRootFilesystem	Trident节点Pod必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点Pod运行有权限的容器、无法删除功能。	none
runAsGroup	Trident容器以root身份运行。	RunAsAny
runAsUser	Trident容器以root身份运行。	runAsAny
runtimeClass	Trident不使用 RuntimeClasses。	空
seLinux	未设置Trident seLinuxOptions、因为容器运行时和Kubnetes分发版处理SELinux的方式目前存在差异。	空
supplementalGroups	Trident容器以root身份运行。	RunAsAny

字段	说明	默认
volumes	Trident Pod需要这些卷插件。	hostPath, projected, emptyDir

安全上下文限制(SCC)

标签	说明	默认
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与进行通信nfsd。	true
allowHostNetwork	iscsiadm要求主机网络与iSCSI守护进程进行通信。	true
allowHostPID	需要主机PID来检查节点上是否`rpc-statd`正在运行。	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
allowedUnsafeSysctls	Trident不需要任何不安全的sysctls。	none
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
groups	此SCC专用于Trident并绑定到其用户。	空
readOnlyRootFilesystem	Trident节点Pod必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点Pod运行有权限的容器、无法删除功能。	none
runAsUser	Trident容器以root身份运行。	RunAsAny
seLinuxContext	未设置Trident seLinuxOptions、因为容器运行时和Kubnetes分发版处理SELinux的方式目前存在差异。	空
seccompProfiles	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

标签	说明	默认
`users`	提供了一个条目、用于将此SCC绑定到Trident命名空间中的Trident用户。	不适用
`volumes`	Trident 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/。