Astra Trident 22.10

Astra Trident

NetApp
March 17, 2025

目录

Astra Trident 22.10

发行说明

发行说明提供了有关最新版本的 Astra Trident 中的新增功能，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` 中提供的二进制文件 `/extras` 此zip文件的一部分未经过测试或不受支持。

22.10中的新增功能

在升级到**Astra Trident 22.10**之前、您必须阅读以下关键信息。

<Strong>有关Astra Trident 22.10的关键信息</Strong>

- 现在、Trident支持Kubernetes 1.25。在升级到Kubernetes 1.25之前、您必须将Astra Trident升级到22.10。
- Astra Trident现在严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

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

修复

- 已修复使用创建的ONTAP 后端专用的问题描述 `credentials` 字段在22.07.0升级期间无法联机("问题描述 #759"）。
- "** Docker："*修复了导致Docker卷插件在某些环境中无法启动的问题描述 ("问题描述 #548" 和 "问题描述 760"）。
- 修复了ONTAP SAN后端专用的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Astra Trident iSCSI工作流中的粒度重试、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Kubernetes 1.25的支持。在升级到Kubernetes 1.25之前、您必须将Astra Trident升级到22.10。
  - 为Trident部署和DemonSet添加了单独的ServiceAccount、ClusterRole和ClusterRoleBinding-以增强未来的权限。
  - 增加了对的支持 "跨命名空间卷共享"。
- 所有Trident `ontap-*` 现在、存储驱动程序可与ONTAP REST API配合使用。

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

已弃用

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

22.07中的更改

修复

*

- 修复了使用Helm或Trident运算符配置Trident时用于处理节点选择器的布尔值和数字值的问题描述。（"GitHub问题描述 700"）
- 修复了问题描述 处理非CHAP路径错误的问题、以便kubelet在失败时重试。 "GitHub问题描述 #736"）

增强功能

- 从K8s.gcr.io过渡到registry.k8s.io作为CSI映像的默认注册表
- 现在、ONTAP SAN卷将使用每个节点的igroup、并且仅将LUN映射到igroup、而将其主动发布到这些节点、以改善我们的安全状况。如果Trident确定在不影响活动工作负载的情况下安全执行此操作、现有卷将有机会切换到新的igroup方案。
- 包含一个包含Trident安装的ResourceQuota、以确保在默认情况下限制使用PriorityClass时计划Trident DemonSet。
- 为ANF驱动程序增加了对网络功能的支持。（"GitHub问题描述 #717"）
- 为ONTAP 驱动程序添加了技术预览自动MetroCluster 切换检测功能。（"GitHub问题描述 #228"）

已弃用

- 。。。。
- 后端配置不再允许在一个配置中使用多种身份验证类型。

删除

- 已删除AWS CVS驱动程序(自22.04起已弃用)。
- Kubernetes

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

文档。

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

22.04中的更改

NetApp 不断改进和完善其产品和服务。以下是 Astra Trident 中的一些最新功能。对于先前版本，请参见 "文档的早期版本"。

> 如果要从先前的任何Trident版本升级并使用Azure NetApp Files 、则会显示 `location` 现在、config参数为必填字段、即单个字段。

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问题描述 #6771"）
- 修复了不准确的 CSI Snapshot 日志。（"GitHub问题描述 #629"）
- 修复了已删除节点上的卷已取消发布的问题描述 。（"GitHub 问题描述 第 691 号"）
- 增加了对块设备上文件系统不一致问题的处理。（"GitHub问题描述 #656"）
- 修复了设置时问题描述 提取自动支持映像的问题 `imageRegistry` 安装期间标记。（"GitHub问题描述 #715"）
- 修复了问题描述 ，其中 ANF 驱动程序无法使用多个导出规则克隆卷。

增强功能

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

删除

已删除 E 系列驱动程序（自 2007 年 20 月 20 日起禁用）。

22.01.1 中的更改

修复

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

22.01.0 中的更改

修复

- * Kubernetes： * 增加大型集群的节点注册回退重试时间。
- 修复了问题描述 ，其中 azure-netapp-files 驱动程序可能会被同名的多个资源混淆。
- 如果使用括号指定 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 卷上创建超过 32 个快照时 Astra Trident 性能下降的问题描述 。
- 在创建 SSL 证书时将 SHA-1 替换为 SHA-256 。
- 固定的 ANF 驱动程序，允许重复的资源名称并将操作限制在一个位置。
- 固定的 ANF 驱动程序，允许重复的资源名称并将操作限制在一个位置。

增强功能

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

已弃用

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

21.10.1 中的更改

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

修复

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

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

21.10.0 中的更改

修复

- 修复了问题描述，其中无法将 XFS 卷的克隆挂载到与源卷相同的节点上（ GitHub 问题描述 514 ）。
- 修复了问题描述，其中 Astra Trident 在关闭时记录了致命错误（ GitHub 问题描述 597 ）。
- 与 Kubernetes 相关的修复程序：
  - 使用创建快照时、返回卷的已用空间作为最小还原大小 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序(GitHub问题描述 645)。
  - 修复了问题描述 、其中 `Failed to expand filesystem` 调整卷大小后记录了错误(GitHub问题描述 560)。
  - 修复了POD可能卡在其中的问题描述 `Terminating` 状态(GitHub问题描述 572)。
  - 修复了以下情况 `ontap-san-economy` FlexVol 可能已满快照LUN (GitHub问题描述 533)。
  - 使用不同映像修复了自定义 YAML 安装程序问题描述（ GitHub 问题描述 613" ）。
  - 固定快照大小计算（ GitHub 问题描述 611 ）。
  - 修复了问题描述，其中所有 Astra Trident 安装程序都可以将纯 Kubernetes 标识为 OpenShift （ GitHub 问题描述 639 ）。
  - 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

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

实验增强功能

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

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。

- Astra Trident现在强制使用空 `fsType` (`fsType=""`) `fsType` 在其StorageClass中指定。使用Kubernetes 1.17或更高版本时、Trident支持提供一个空 `fsType` NFS卷。对于iSCSI卷、您需要设置 `fsType` 在StorageClass上执行 `fsGroup` 使用安全上下文。
- 在多个Astra Trident实例之间使用后端时、每个后端配置文件都应具有不同的 `storagePrefix` ONTAP 后端值或使用其他值 `TenantName` 适用于SolidFire 后端。Astra Trident 无法检测其他 Astra Trident 实例创建的卷。尝试在 ONTAP 或 SolidFire 后端创建现有卷会成功，因为 Astra Trident 会将卷创建视为一项幂等操作。条件 `storagePrefix` 或 `TenantName` 请勿有所不同、在同一后端创建的卷可能存在名称冲突。
- 安装Astra Trident时(使用 `tridentctl` 或Trident运算符)并使用 `tridentctl` 要管理Astra Trident、您应确保 `KUBECONFIG` 已设置环境变量。这一点对于指示Kubernetes集群来说是必要的 `tridentctl` 应采取应对措施。在使用多个Kubernetes环境时、您应确保 `KUBECONFIG` 文件来源准确。
- 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。对于需要的RHEL/RedHat CoreOS实例来说、情况就是如此 `discard` "挂载选项"；确保中包含 Discard mountOption[`StorageClass`^)以支持联机块丢弃。
- 如果每个 Kubernetes 集群有多个 Astra Trident 实例，则 Astra Trident 将无法与其他实例通信，也无法发现它们创建的其他卷，如果集群中运行多个实例，则会导致意外的错误行为。每个 Kubernetes 集群只能有一个 Astra Trident 实例。
- 如果基于Astra Trident `StorageClass` 对象将从Kubernetes中删除当Astra Trident脱机时、Astra Trident不会在其数据库恢复联机时从其数据库中删除相应的存储类。您应使用删除这些存储类 `tridentctl` 或REST API。
- 如果用户在删除相应的 PVC 之前删除了由 Astra Trident 配置的 PV ，则 Astra Trident 不会自动删除后备卷。您应通过删除此卷 `tridentctl` 或REST API。
- 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Astra Trident时、应指定 `managementLIF` 和 `dataLIF` 后端定义中方括号内。例如：`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。
- 如果使用 `solidfire-san` 驱动程序对于OpenShift 4.5、请确保底层工作节点使用MD5作为CHAP身份验证算法。Element 12.7提供了符合FIPS的安全CHAP算法SHA1、SHA-256和SHA3-256。

了解更多信息

- "Astra Trident GitHub"
- "Astra Trident 博客"

概念

了解 Astra Trident

Astra Trident 是一个完全受支持的开源项目，由 NetApp 在中维护 "Astra 产品系列"。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。

概述

Astra Trident 作为 Pod 部署在 Kubernetes 集群中，并为您的 Kubernetes 工作负载提供动态存储编排服务。它可以让您的容器化应用程序快速轻松地使用NetApp广泛产品组合中的永久性存储、这些产品组合包括ONTAP (AFF/FAS/Select/Cloud/Amazon FSX for NetApp ONTAP)、Element软件(NetApp HCI/SolidFire)以及Azure NetApp Files 服务以及Google Cloud上的Cloud Volumes Service。

Astra Trident 也是 NetApp Astra 的一项基础技术，利用 NetApp 行业领先的快照，备份，复制和克隆数据管理技术，可满足 Kubernetes 工作负载的数据保护，灾难恢复，可移动性和迁移用例。

支持的 **Kubernetes** 集群架构

以下 Kubernetes 架构支持 Astra Trident：

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

什么是 **Astra** ？

借助 Astra ，企业可以更轻松地在公有云内部和内部环境中管理，保护和移动 Kubernetes 上运行的数据丰富的容器化工作负载。Astra 使用 NetApp 在公有云和内部环境中成熟而广泛的存储产品组合中的 Astra Trident 配置和提供永久性容器存储。此外，它还提供了一组丰富的高级应用程序感知型数据管理功能，例如快照，备份和还原，活动日志和主动克隆，用于数据保护，灾难 / 数据恢复，数据审核以及 Kubernetes 工作负载的迁移用例。

您可以在 Astra 页面上注册免费试用版。

有关详细信息 ...

- "NetApp Astra 产品系列"
- "Astra Control Service 文档"
- "Astra 控制中心文档"

- "Astra API 文档"

ONTAP 驱动程序

Astra Trident 提供了五个唯一的 ONTAP 存储驱动程序，用于与 ONTAP 集群进行通信。详细了解每个驱动程序如何处理卷的创建，访问控制及其功能。

了解ONTAP 存储驱动程序

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	文件系统	rwo ， rox ， rwx	"" ， NFS
`ontap-nas-economy`	NFS	文件系统	rwo ， rox ， rwx	"" ， NFS
`ontap-nas-flexgroup`	NFS	文件系统	rwo ， rox ， rwx	"" ， NFS
`ontap-san`	iSCSI	块	rwo ， rox ， rwx	无文件系统；原始块设备
`ontap-san`	iSCSI	文件系统	工单， ROX rwx在文件系统卷模式下不可用。	`xfs， ext3， ext4`
`ontap-san-economy`	iSCSI	块	rwo ， rox ， rwx	无文件系统；原始块设备
`ontap-san-economy`	iSCSI	文件系统	工单， ROX rwx在文件系统卷模式下不可用。	`xfs， ext3， ext4`

可以使用安全角色的登录凭据(用户名/密码)或使用ONTAP 集群上安装的私钥和证书对ONTAP 后端进行身份验证。您可以使用更新现有后端以从一种身份验证模式移至另一种身份验证模式 `tridentctl update backend`。

配置

在 Astra Trident 中配置有两个主要阶段。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关联取决于存储类请求的属性及其属性 `storagePools`， `additionalStoragePools` ，和 `excludeStoragePools` 列表。创建存储类时， Trident 会将其每个后端提供的属性和池与存储类请求的属性和池进行比较。如果存储池的属性和名称与请求的所有属性和池名称匹配，则 Astra 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、Astra Trident会添加中列出的所有存储池 `additionalStoragePools` 列出到该集、即使其属性不满足存储类请求的所有或任何属性也是如此。您应使用 `excludeStoragePools` 用于覆盖和删除存储类使用的存储池的列表。每次添加新后端时， Astra Trident 都会执行类似的过程，检查其存储池是否满足现有存储类的要求，并删除任何已标记为已排除的。

卷创建

然后， Astra Trident 会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， Astra Trident 会首先获取该卷的存储类的一组存储池， 此外，如果为卷指定协议，则 Astra Trident 会删除无法提供所请求协议的存储池（例如， NetApp HCI/SolidFire 后端无法提供基于文件的卷，而 ONTAP NAS 后端无法提供基于块的卷）。Astra Trident 会随机分配此结果集的顺序，以便均匀分布卷，然后迭代并依次尝试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。只有在 * 无法在 * 所有 * 可用于请求的存储类和协议的存储池上配置时， Astra Trident 才会返回故障。

卷快照

详细了解 Astra Trident 如何为其驱动程序创建卷快照。

了解如何创建卷快照

- 。 `ontap-nas`， `ontap-san`， `gcp-cvs`，和 `azure-netapp-files` 驱动程序、每个永久性卷(PV)都会映射到一个FlexVol。因此，卷快照会创建为 NetApp 快照。与竞争对手的 Snapshot 技术相比， NetApp 的 Snapshot 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。
- 。 `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 Snapshot 技术相比， NetApp 的 Snapshot 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。
- 。 `ontap-san-economy` 驱动程序、PV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 的 FlexClone 技术，即使是最大的数据集，也可以几乎即时创建副本。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。
- 。 `solidfire-san` 驱动程序、每个PV都会映射到在NetApp Element 软件/NetApp HCI集群上创建的 LUN。VolumeSnapshot 由底层 LUN 的 Element Snapshot 表示。这些快照是时间点副本，只占用少量系统资源和空间。
- 使用时 `ontap-nas` 和 `ontap-san` 驱动程序、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 Kubernetes 创建的卷快照后， Astra 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。

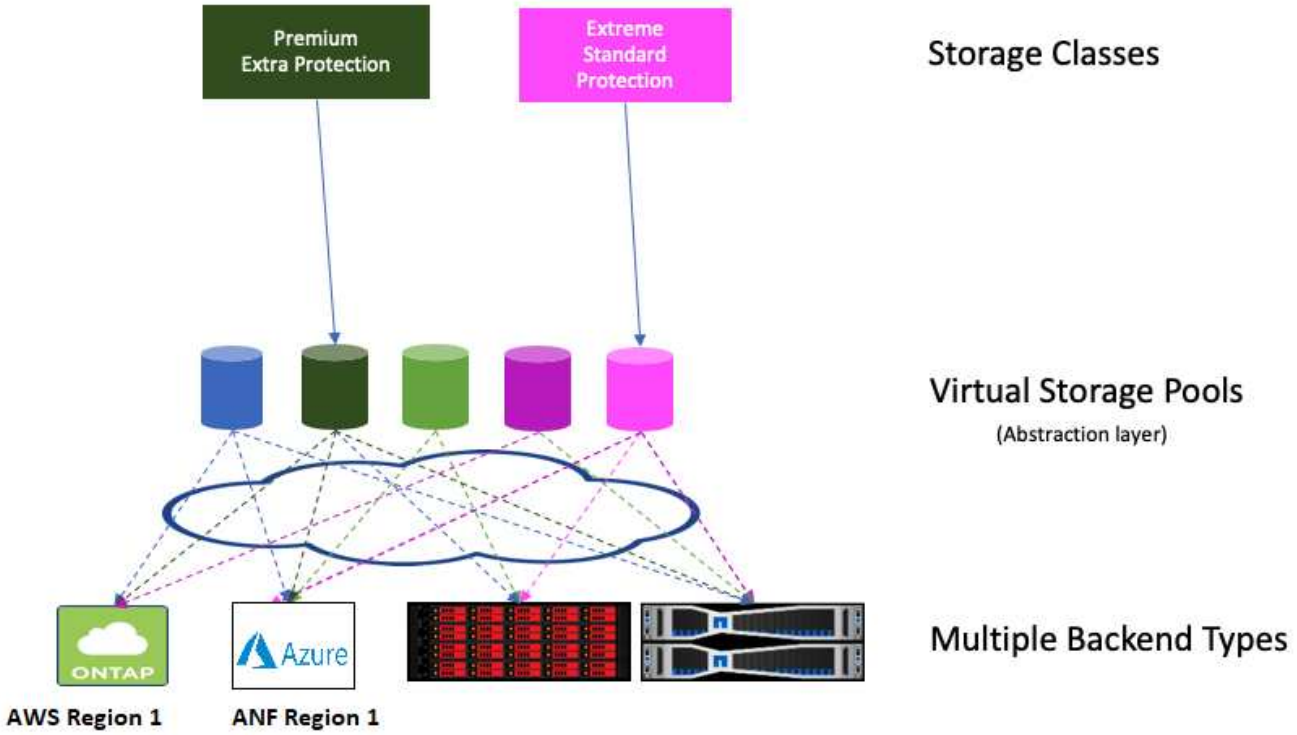
借助 Astra Trident ，您可以使用 VolumeSnapshots 创建新的 PV 。通过对支持的 ONTAP 和 CVS 后端使用 FlexClone 技术，可以从这些快照创建 PV 。从快照创建 PV 时，后备卷是快照父卷的 FlexClone 。。 `solidfire-san` 驱动程序使用Element软件卷克隆从快照创建PV。此时，它将从 Element 快照创建一个克隆。

虚拟存储池

虚拟存储池在Astra Trident的存储后端和Kubernetes之间提供了一个抽象层 `StorageClasses`。管理员可以通过这些协议以一种通用的、与后端无关的方式定义各个方面、例如每个后端的位置、性能和保护、而无需创建 `StorageClass` 指定要用于满足所需条件的物理后端、后端池或后端类型。

了解虚拟存储池

存储管理员可以在 JSON 或 YAML 定义文件中的任一 Astra Trident 后端定义虚拟存储池。



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）。

> 定义虚拟存储池时，请勿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。

大多数方面都以后端特定术语来指定。重要的是、宽高值不会公开在后端驱动程序之外、也不能在中进行匹配 `StorageClasses`。而是管理员为每个虚拟池定义一个或多个标签。每个标签都是一个键：值对，标签可能在唯一的后端通用。与其他方面一样，可以为每个池指定标签，也可以为后端指定全局标签。与具有预定义名称和值的方面不同，管理员可以根据需要全权定义标签键和值。

答 `StorageClass` 通过引用选择器参数中的标签来标识要使用的虚拟池。虚拟池选择器支持以下运算符：

运算符	示例	池的标签值必须：
=	性能 = 高级	匹配
!=	性能！ = 至高	不匹配
in	位置（东部，西部）	位于一组值中
notin	性能注释（银牌，铜牌）	不在值集内
<key>	protection	存在任何值
!<key>	！保护	不存在

卷访问组

了解有关 Astra Trident 如何使用的更多信息 "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 Astra Trident ，则可以忽略此部分。在作为增强型 CSI 配置程序安装时， Astra Trident 会使用 CHAP 。

了解卷访问组

Astra 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、则它希望找到一个名为的访问组 `trident` 除非在配置中指定一个或多个访问组ID。

虽然 Astra Trident 会将新卷与已配置的访问组相关联，但它不会自行创建或管理访问组。在将存储后端添加到 Astra Trident 之前，访问组必须存在，并且这些访问组必须包含 Kubernetes 集群中每个节点的 iSCSI IQN ，这些节点可能会挂载该后端配置的卷。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"。

修改使用默认值的配置时 `trident` 访问组到也使用其他的访问组、并包括的ID `trident` 列表中的访问组。

入门

试用

NetApp 提供了一个可随时使用的实验室映像，您可以通过该映像进行请求 "NetApp 试用"。

了解测试活动

此测试驱动器为您提供了一个沙盒环境，该环境附带安装和配置了三节点 Kubernetes 集群和 Astra Trident 。这是熟悉 Astra Trident 并了解其功能的好方法。

另一个选项是查看 "《 kubeadm 安装指南》" 由 Kubernetes 提供。

> 您不应在生产环境中使用使用这些说明构建的 Kubernetes 集群。使用您的分发版提供的生产部署指南创建可随时投入生产的集群。

如果这是您第一次使用 Kubernetes ，请熟悉相关概念和工具 "此处"。

要求

首先查看支持的前端，后端和主机配置。

> 要了解 Astra Trident 使用的端口，请参见 "此处"。

有关**Astra Trident 22.10**的关键信息

在升级到**Astra Trident 22.10**之前、您必须阅读以下关键信息。

> **<Strong>有关Astra Trident 22.10的关键信息</Strong>**
>
> - 现在、Trident支持Kubernetes 1.25。在升级到Kubernetes 1.25之前、您必须先升级到Astra Trident 22.10。
> - Astra Trident现在严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。
>
> 使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

支持的前端（编排程序）

Astra 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 1.9、1.10、1.11
- Kubernetes 1.20 - 1.25

- Mirantis Kubernetes Engine 3.5
- OpenShift 4.8、4.9、4.10、4.11

以下版本支持 Trident 操作符：

- Anthos on-Prem (VMware)和Anthos on bare metal 1.9、1.10、1.11
- Kubernetes 1.20 - 1.25
- OpenShift 4.8、4.9、4.10、4.11

Astra Trident 还可与许多其他完全托管和自我管理的 Kubernetes 产品结合使用，包括 Google Kubernetes Engine （ GKEE ）， Amazon Elastic Kubernetes Services （ EKS ）， Azure Kubernetes Service （ AKS ）， Rancher 和 VMware Tanzu Portfolio 。

支持的后端（存储）

要使用 Astra Trident ，您需要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
- 适用于 GCP 的 Cloud Volumes Service
- FAS/AFF/ 选择 9.3 或更高版本
- 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 Astra 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?
CSI Trident	1.20 - 1.25	否
卷快照	1.20 - 1.25	否
卷快照中的 PVC	1.20 - 1.25	否
iSCSI PV 调整大小	1.20 - 1.25	否
ONTAP 双向 CHAP	1.20 - 1.25	否
动态导出策略	1.20 - 1.25	否
Trident 运算符	1.20 - 1.25	否

功能	**Kubernetes** 版本	是否需要功能安全门?
自动工作节点准备（测试版）	1.20 - 1.25	否
CSI 拓扑	1.20 - 1.25	否

已测试主机操作系统

虽然Astra Trident不会正式"支持"特定的操作系统、但已知以下操作系统有效：

- OpenShift 容器平台支持的 RedHat CoreOS （ RHCOS ）版本
- RHEL或CentOS 7.
- Ubuntu 18.04或更高版本(最新版本22.04)
- Windows Server 2019

默认情况下， Astra Trident 在容器中运行，因此将在任何 Linux 工作程序上运行。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 Astra Trident 提供的卷。

。 `tridentctl` 实用程序还可以在Linux的任何这些分发版上运行。

主机配置

根据所使用的后端，应在集群中的所有工作人员上安装 NFS 和 / 或 iSCSI 实用程序。请参见 "此处" 有关详细信息 …

存储系统配置：

Astra Trident 可能需要先对存储系统进行一些更改，然后后端配置才能使用它。请参见 "此处" 了解详细信息。

容器映像以及相应的 **Kubernetes** 版本

对于带气的安装，下面列出了安装 Astra Trident 所需的容器映像。使用 `tridentctl images` 用于验证所需容器映像列表的命令。

Kubernetes 版本	容器映像
v1.20.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)
v1.21.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)
v1.22.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)

Kubernetes 版本	容器映像
v1.23.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)
v1.24.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)
v1.25.0	• dDocker。io/NetApp/trident：22.10.0 • docer.io/NetApp/trident-autostsupport：22.10 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.3.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.0.0 • 注册表.k8s.io/sig-storage/CsI-s不同：v1.6.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.1.0 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.5.1 • dkoper.io/NetApp/trident-operator：22.10.0 (可选)

在Kubernetes 1.20及更高版本上、使用经验证的 `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x` 仅当出现时才创建映像 `v1` 版本正在提供 `volumesnapshots.snapshot.storage.k8s.gcr.io` CRD。如果 `v1beta1` 版本正在为CRD提供支持/不提供 `v1` 版本、请使用已验证的 `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x` 图像。

部署概述

您可以使用Trident操作员或与一起部署Astra Trident `tridentctl`。

从 22.04 版开始，每次安装 Astra Trident 时， AES 密钥将不再重新生成。在此版本中， Astra Trident 将安装一个新的机密对象，该对象会在安装之间持续存在。这意味着、 `tridentctl` 在22.04中、可以卸载先前版本的Trident、但早期版本无法卸载22.04安装。

有关Astra Trident 22.10的关键信息

在升级到**Astra Trident 22.10**之前、您必须阅读以下关键信息。

<Strong>有关Astra Trident 22.10的关键信息</Strong>

- 现在、Trident支持Kubernetes 1.25。在升级到Kubernetes 1.25之前、您必须先升级到Astra Trident 22.10。
- Astra Trident现在严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

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

选择部署方法

要确定要使用的部署方法，请考虑以下几点：

何时使用Trident运算符

。 "Trident 运算符" 是动态管理 Astra Trident 资源并自动完成设置阶段的绝佳方式。必须满足某些前提条件。请参见 "要求"。

Trident 运算符提供了以下几项优势。

自我修复功能

您可以监控 Astra Trident 安装并主动采取措施来解决问题，例如删除部署或意外修改部署的时间。将操作员设置为部署时、会显示 `trident-operator-<generated-id>` POD已创建。此POD关联A `TridentOrchestrator` 安装了Astra Trident的CR、并始终确保只有一个处于活动状态 `TridentOrchestrator`。换言之，操作员可确保集群中只有一个 Astra Trident 实例并控制其设置，从而确保安装有效。对安装进行更改（例如删除部署或节点取消设置）时，操作员会识别这些更改并逐个修复它们。

轻松更新现有安装

您可以使用操作员轻松更新现有部署。您只需编辑 `TridentOrchestrator` cr以更新安装。例如，请考虑需要启用 Astra Trident 以生成调试日志的情形。

为此、请修补 `TridentOrchestrator` 设置 `spec.debug` to `true`：

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

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

自动处理 **Kubernetes** 升级

当集群的 Kubernetes 版本升级到受支持的版本时，操作员会自动更新现有的 Astra Trident 安装并进行更改，以确保其满足 Kubernetes 版本的要求。

> 如果集群升级到不受支持的版本，则操作员会阻止安装 Astra Trident 。如果已随操作员安装了 Astra Trident ，则会显示一条警告，指示 Astra Trident 安装在不受支持的 Kubernetes 版本上。

使用**BlueXP (**以前称为**Cloud Manager)**管理**Kubernetes**集群

使用 "使用BlueXP的Astra Trident"、您可以升级到最新版本的Astra Trident、添加和管理存储类并将其连接到工作环境、以及使用Cloud Backup Service 备份永久性卷。BlueXP支持使用Trident操作员手动或使用Helm部署Astra Trident。

何时使用**Helm**

如果您还有其他要使用 Helm 管理的应用程序，从 Astra Trident 21.01 开始，您也可以使用 Helm 管理您的部署。

何时使用 `tridenctl`

如果您的现有部署必须升级到、或者您希望对部署进行高度自定义、则应考虑使用 "Tridentctl"。这是部署 Astra Trident 的传统方法。

在部署方法之间移动时的注意事项

不难想象需要在部署方法之间移动的情形。在尝试从移动之前、应考虑以下事项 `tridentctl` 部署到基于操作员的部署、反之亦然：

- 卸载 Astra Trident 时，请始终使用相同的方法。如果您已使用部署 `tridentctl`、您应使用的相应版本 `tridentctl` 用于卸载Astra Trident的二进制文件。同样、如果要使用操作员进行部署、则应编辑 `TridentOrchestrator` CR和设置 `spec.uninstall=true` 卸载Astra Trident。
- 如果您的部署基于操作员、则要删除并使用 `tridentctl` 要部署Astra Trident、您应先编辑 `TridentOrchestrator` 并设置 `spec.uninstall=true` 卸载Astra Trident。然后删除 `TridentOrchestrator` 和操作员部署。然后、您可以使用安装 `tridentctl`。
- 如果您使用的是基于操作员的手动部署，并且要使用基于 Helm 的 Trident 操作员部署，则应先手动卸载此操作员，然后再执行 Helm 安装。这样， Helm 就可以使用所需的标签和标注来部署 Trident 操作员。如果不执行此操作，则基于 Helm 的 Trident 操作员部署将失败，并显示标签验证错误和标注验证错误。如果您有 `tridentctl`基于部署、您可以使用基于Helm的部署、而不会遇到问题。

了解部署模式

部署 Astra Trident 的方法有三种。

标准部署

在 Kubernetes 集群上部署 Trident 会导致 Astra Trident 安装程序执行以下两项操作：

- 通过 Internet 提取容器映像
- 创建部署和 / 或节点取消设置，以便在 Kubernetes 集群中所有符合条件的节点上启动 Astra Trident Pod 。

此类标准部署可以通过两种不同的方式执行：

- 使用 `tridentctl install`
- 使用 Trident 运算符。您可以手动或使用 Helm 部署 Trident 操作员。

此安装模式是安装 Astra Trident 的最简单方法，适用于大多数不会实施网络限制的环境。

脱机部署

要执行带风的部署、您可以使用 `--image-registry` 调用时标记 `tridentctl install` 指向私有映像注册表。如果使用Trident操作符进行部署、则也可以指定 `spec.imageRegistry` 在中 `TridentOrchestrator`。此注册表应包含 "Trident 映像"， "Trident AutoSupport 映像"和 Kubernetes 版本所需的 CSI sidecar 映像。

要自定义部署、您可以使用 `tridentctl` 为Trident的资源生成清单。其中包括 Astra Trident 在安装过程中创建的部署，取消设置，服务帐户和集群角色。

有关自定义部署的详细信息，请参见以下链接：

- "自定义基于操作员的部署"
  *

> 如果您使用的是私有映像存储库、则应添加 `/sig-storage` 到专用注册表URL的末尾。使用私有注册表时 `tridentctl` 部署、您应使用 `--trident-image` 和 `--autosupport-image` 与结合使用 `--image-registry`。如果要使用Trident操作符部署Astra Trident、请确保Orchestrator CR包括在内 `tridentImage` 和 `autosupportImage` 在安装参数中。

远程部署

下面简要概述了远程部署过程：

- 部署适当版本的 `kubectl` 在要部署Astra Trident的远程计算机上。
- 从Kubernetes集群复制配置文件并设置 `KUBECONFIG` 远程计算机上的环境变量。
- 启动 `kubectl get nodes` 命令以验证是否可以连接到所需的Kubernetes集群。
- 使用标准安装步骤从远程计算机完成部署。

其他已知配置选项

在 VMware Tanzu Portfolio 产品上安装 Astra Trident 时：

- 集群必须支持有权限的工作负载。
- 。 `--kubelet-dir` 标志应设置为kubelet目录的位置。默认情况下、此值为 `/var/vcap/data/kubelet`。

  使用指定kubelet位置 `--kubelet-dir` 已知适用于Trident操作员、Helm和 `tridentctl` 部署。

使用 Trident 操作员进行部署

您可以使用Trident操作员部署Astra Trident。

有关Astra Trident 22.10的关键信息

在升级到**Astra Trident 22.10**之前、您必须阅读以下关键信息。

<Strong>有关Astra Trident 22.10的关键信息</Strong>

- 现在、Trident支持Kubernetes 1.25。在升级到Kubernetes 1.25之前、您必须先升级到Astra Trident 22.10。
- Astra Trident现在严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

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

Trident操作员部署选项

您可以通过以下两种方式之一部署Trident操作员：

- 使用Trident "Helm图表"：Helm图表将部署Trident操作符并分步安装Trident。
- 手动：Trident提供了一个文件、可用于安装操作员和创建关联对象。
  - 对于运行Kubernetes 1.24或更低版本的集群、请使用 "bundle_pre_1_25.yaml"。
  - 对于运行Kubernetes 1.25或更高版本的集群、请使用 "bundle_post_1_25.yaml"。

如果您尚未熟悉 "基本概念"，现在是一个实现这一目标的好时机。

验证前提条件

要部署 Astra Trident ，应满足以下前提条件：

- 您对运行受支持的Kubernetes版本的受支持Kubernetes集群具有完全权限。查看 "要求"。

- 您可以访问受支持的 NetApp 存储系统。
- 您可以从所有 Kubernetes 工作节点挂载卷。
- 您有一个Linux主机 `kubectl` (或 `oc`(如果您使用的是OpenShift)已安装并配置为管理要使用的Kubernetes集群。
- 您已设置 `KUBECONFIG` 环境变量以指向您的Kubernetes集群配置。
- 您已启用 "Astra Trident 所需的功能门"。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。

明白了吗？太棒了！让我们开始吧。

部署Trident操作员并使用Helm安装Astra Trident

执行列出的步骤以使用 Helm 部署 Trident 操作员。

您需要的内容

除了上述前提条件之外，要使用 Helm 部署 Trident 操作员，还需要满足以下条件：

- 答 "支持的Kubernetes版本"
- Helm 版本 3

步骤

1. 添加 Trident 的 Helm 存储库：

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

2. 使用 `helm install` 命令并为您的部署指定一个名称。请参见以下示例：

```
helm install <name> netapp-trident/trident-operator --version 22.10.0
--create-namespace --namespace <trident-namespace>
```

> 如果您已为Trident创建命名空间、则会显示 `--create-namespace` 参数不会创建其他命名空间。

在安装期间，可以通过两种方式传递配置数据：

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

例如、要更改的默认值 `debug`、运行以下命令 `--set` 命令：

```
helm install <name> netapp-trident/trident-operator --version 22.10.0
--create-namespace --namespace --set tridentDebug=true
```

。 `values.yaml` 文件(属于Helm图表的一部分)提供了密钥列表及其默认值。

`helm list` 显示有关安装的详细信息、例如名称、命名空间、图表、状态、 应用程序版本、修订版号等。

手动部署Trident操作员并安装Trident

执行列出的步骤以手动部署 Trident 操作员。

第 1 步：确定 Kubernetes 集群的资格

首先，您需要登录到 Linux 主机并验证它是否正在管理 _b工作 _ ， "支持的 Kubernetes 集群" 您具有所需权限。

> 使用OpenShift `oc` 而不是 `kubectl` 在下面的所有示例中、运行以*系统：admin*身份登录 `oc login -u system:admin` 或 `oc login -u kube-admin`。

要验证Kubernetes版本、请运行以下命令：

```
kubectl version
```

要查看您是否具有 Kubernetes 集群管理员权限，请运行以下命令：

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

要验证是否可以从 Docker Hub 启动使用映像的 POD 并通过 Pod 网络访问存储系统，请运行以下命令：

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

第 2 步：下载并设置操作员

> 从 21.01 开始， Trident 操作符的范围为集群范围。使用Trident操作符安装Trident需要创建 `TridentOrchestrator` 自定义资源定义(CRD)和定义其他资源。在安装 Astra Trident 之前，您应执行以下步骤来设置操作员。

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

```
wget
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
```

2. 使用相应的CRD清单创建 `TridentOrchestrator` CRD。然后、您将创建 `TridentOrchestrator` 稍后自定义资源以通过操作员实例化安装。

   运行以下命令：

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

3. 在之后 `TridentOrchestrator` 创建CRD后、创建部署操作员所需的以下资源：

   - 操作员的 ServiceAccount
   - 对 ServiceAccount 执行 ClusterRole 和 ClusterRoleBinding
   - 专用的 PodSecurityPolicy
   - 运算符本身

     Trident 安装程序包含用于定义这些资源的清单。默认情况下、操作员部署在中 `trident` 命名空间。如果 `trident` 命名空间不存在、请使用以下清单创建一个。

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

4. 在非默认命名空间中部署运算符 `trident` 命名空间、您应更新 `serviceaccount.yaml`，`clusterrolebinding.yaml` 和 `operator.yaml` 执行清单并生成 `bundle.yaml`。

   运行以下命令以更新YAML清单并生成 `bundle.yaml` 使用 `kustomization.yaml`：

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

   运行以下命令以创建资源并部署操作员：

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

5. 要在部署后验证操作员的状态，请执行以下操作：

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

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator    1/1     1            1           3m
```

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

NAME                                READY   STATUS              RESTARTS
AGE
trident-operator-54cb664d-lnjxh     1/1     Running             0
3m
```

操作员部署成功创建了一个在集群中的一个工作节点上运行的 POD 。

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

第3步：创建 `TridentOrchestrator` 并安装Trident

现在，您可以使用操作员安装 Astra Trident 了！这需要创建 `TridentOrchestrator`。Trident安装程序附带了用于创建的示例定义 `TridentOrchestrator`。此操作将在中启动安装 `trident` 命名空间。

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

kubectl describe torc trident
Name:         trident
Namespace:
Labels:       <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:22.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:21.04.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v21.04.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

使用Trident操作员可以使用中的属性自定义Astra Trident的安装方式 `TridentOrchestrator` 规格请参见 "自定义 Trident 部署"。

的状态 `TridentOrchestrator` 指示安装是否成功、并显示已安装的Trident版本。

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

在安装期间、的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态和操作员无法自行恢复、您应检查操作员的日志。请参见 "故障排除" 部分。

您可以通过查看已创建的 Pod 来确认 Astra Trident 安装是否已完成：

```
kubectl get pod -n trident

NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-7d466bf5c7-v4cpw        5/5     Running   0          1m
trident-csi-mr6zc                   2/2     Running   0          1m
trident-csi-xrp7w                   2/2     Running   0          1m
trident-csi-zh2jt                   2/2     Running   0          1m
trident-operator-766f7b8658-ldzsv   1/1     Running   0          3m
```

您也可以使用 `tridentctl` 检查安装的Astra Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.04.0        | 21.04.0        |
+----------------+----------------+
```

现在，您可以继续创建后端。请参见 "部署后任务"。

有关在部署期间排除问题的信息，请参见 "故障排除" 部分。

自定义 Trident 操作员部署

使用Trident操作员可以使用中的属性自定义Astra Trident安装 `TridentOrchestrator` 规格

如果您要对安装进行自定义、使其超出预期范围 `TridentOrchestrator` 参数允许、您应考虑使用 `tridentctl` 生成可根据需要修改的自定义YAML清单。

> `spec.namespace` 在中指定 `TridentOrchestrator` 表示安装了Astra Trident的命名空间。此参数 * 安装 Astra Trident 后无法更新 * 。如果尝试执行此操作、则会导致 `TridentOrchestrator` 要更改为的状态 `Failed`。Astra Trident不能跨命名空间迁移。

配置选项

此表详细介绍了相关信息 `TridentOrchestrator` 属性：

参数	Description	Default
`namespace`	用于安装 Astra Trident 的命名空间	default
`debug`	为 Astra Trident 启用调试	false
`windows`	设置为 `true` 启用在Windows工作节点上的安装。	false
`IPv6`	安装基于 IPv6 的 Astra Trident	false
`k8sTimeout`	Kubernetes 操作超时	30 秒
`silenceAutosupport`	不要自动向 NetApp 发送 AutoSupport 捆绑包	false
`enableNodePrep`	自动管理工作节点依赖关系（ * 测试版 * ）	false
`autosupportImage`	AutoSupport 遥测的容器映像	"NetApp/trident自动支持：22.10.0"
`autosupportProxy`	用于发送 AutoSupport 遥测的代理的地址 / 端口	"http://proxy.example.com:8888"
`uninstall`	用于卸载 Astra Trident 的标志	false
`logFormat`	要使用的 Astra Trident 日志记录格式 [text ， json]	文本
`tridentImage`	要安装的 Astra Trident 映像	"NetApp/Trident ： 21.04"
`imageRegistry`	内部注册表的路径、格式 `<registry FQDN>[:port][/subpath]`	"K8s.gcr.io/SIG-storage (K8s 1.19+)或quay.io/k8scsi "
`kubeletDir`	主机上的 kubelet 目录的路径	"/var/lib/kubelet"
`wipeout`	要删除以执行 Astra Trident 完全删除的资源列表	

参数	Description	Default
imagePullSecrets	从内部注册表中提取映像的机密信息	
controllerPluginNodeSelector	运行 Trident 控制器 CSI 插件的 Pod 的其他节点选择器。格式与 pod.spec.nodeSelector 相同。	无默认值；可选
controllerPluginTolerations	覆盖运行 Trident 控制器 CSI 插件的 Pod 的容错。格式与 po.spec.Tolerations 相同。	无默认值；可选
nodePluginNodeSelector	运行 Trident Node CSI 插件的 Pod 的其他节点选择器。格式与 pod.spec.nodeSelector 相同。	无默认值；可选
nodePluginTolerations	覆盖运行 Trident Node CSI 插件的 Pod 的容错。格式与 po.spec.Tolerations 相同。	无默认值；可选

有关格式化 POD 参数的详细信息，请参见 "将 Pod 分配给节点"。

配置示例

您可以在定义时使用上述属性 TridentOrchestrator 自定义安装。

示例1：基本自定义配置

这是一个基本自定义配置示例。

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

示例**2**：使用节点选择器部署

此示例说明了如何使用节点选择器部署Trident：

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

示例**3**：在**Windows**工作节点上部署

此示例说明了如何在Windows工作节点上部署。

```
$ cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true
```

使用 tridentctl 进行部署

您可以使用部署Astra Trident `tridentctl`。最好熟悉一下 "基本概念"。以自定义 `tridentctl` 部署、请参见 "自定义 tridentctl 部署"。

有关**Astra Trident 22.10**的关键信息

在升级到**Astra Trident 22.10**之前、您必须阅读以下关键信息。

<Strong>有关Astra Trident 22.10的关键信息</Strong>

- 现在、Trident支持Kubernetes 1.25。在升级到Kubernetes 1.25之前、您必须先升级到Astra Trident 22.10。
- Astra Trident现在严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

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

验证前提条件

要部署 Astra Trident ，应满足以下前提条件：

- 对受支持的Kubernetes集群具有完全权限。
- 访问受支持的NetApp存储系统。
- 能够从所有Kubernetes工作节点挂载卷。
- 具有的Linux主机 `kubectl` (或 `oc`(如果您使用的是OpenShift)已安装并配置为管理要使用的Kubernetes集群。
- 。 `KUBECONFIG` 环境变量指向您的Kubernetes集群配置。
- 。 "Astra Trident 所需的功能门" 已启用。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。

第 1 步：确定 Kubernetes 集群的资格

登录到Linux主机并验证它是否正在管理一个正常运行的、 "支持的 Kubernetes 集群" 您拥有必要的特权。

您可以使用OpenShift `oc` 而不是 `kubectl` 在下面的所有示例中、您应先以*系统：admin*身份运行登录 `oc login -u system:admin` 或 `oc login -u kube-admin`。

要检查 Kubernetes 版本，请运行以下命令：

```
kubectl version
```

要验证Kubernetes集群管理员权限、请运行以下命令：

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

要验证是否可以从 Docker Hub 启动使用映像的 POD 并通过 Pod 网络访问存储系统，请运行以下命令：

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

确定 Kubernetes 服务器版本。您将在安装 Astra Trident 时使用它。

第 2 步：下载并提取安装程序

Trident 安装程序将创建 Trident Pod ，配置用于保持其状态的 CRD 对象，并初始化执行配置卷和将卷附加到集群主机等操作的 CSI sidecars 。

您可以从下载并提取最新版本的Trident安装程序包 "GitHub上的_assets_部分"。

例如、如果最新版本为22.10.0：

```
wget https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
```

第 3 步：安装 Astra Trident

通过执行在所需命名空间中安装Astra Trident `tridentctl install` 命令：

```
./tridentctl install -n trident
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-csi-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=22.10.0
INFO Trident installation succeeded.
....
```

要使Astra Trident能够在Windows节点上运行、请添加 `--windows` 安装命令的标志： `$ ./tridentctl install --windows -n trident`。

安装程序完成后、将显示类似于以下内容的输出。根据Kubernetes集群中的节点数、可能存在更多的Pod：

```
kubectl get pod -n trident
NAME                           READY   STATUS    RESTARTS   AGE
trident-csi-679648bd45-cv2mx   4/4     Running   0          5m29s
trident-csi-vgc8n              2/2     Running   0          5m29s

./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 22.10.0        | 22.10.0        |
+----------------+----------------+
```

要完成Astra Trident配置、请继续 "部署后任务"。

如果安装程序未成功完成或 `trident-csi-<generated id>` 未处于*正在运行*状态、未安装平台。

有关部署期间的故障排除问题、请参见 "故障排除"。

自定义 tridentctl 部署

您可以使用Astra Trident安装程序自定义部署。

了解安装程序

使用Astra Trident安装程序可以自定义属性。例如、如果已将Trident映像复制到专用存储库、则可以使用指定映像名称 `--trident-image`。如果已将Trident映像以及所需的CSI sidecar映像复制到专用存储库、则最好使用指定该存储库的位置 `--image-registry` 交换机、其形式为 `<registry FQDN>[:port]`。

如果您使用的是Kubernetes的分发版、其中 `kubelet` 将其数据保留在非正常路径上 `/var/lib/kubelet`、您可以使用指定备用路径 `--kubelet-dir`。

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

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`

- `trident-service.yaml`
- `trident-namespace.yaml`
- `trident-serviceaccount.yaml`
- `trident-resourcequota.yaml`

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

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

下一步是什么?

部署 Astra Trident 后，您可以继续创建后端，创建存储类，配置卷以及将卷挂载到 Pod 中。

第 1 步：创建后端

现在，您可以继续创建一个后端，供 Astra Trident 配置卷使用。为此、请创建 `backend.json` 包含必要参数的文件。可在中找到不同后端类型的示例配置文件 `sample-input` 目录。

请参见 "此处" 有关如何为后端类型配置文件的更多详细信息。

```
cp sample-input/<backend template>.json backend.json
vi backend.json
```

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

如果创建失败，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 -n trident logs
```

解决问题后，只需返回到此步骤的开头并重试。有关更多故障排除提示，请参见 "故障排除" 部分。

第 2 步：创建存储类

Kubernetes 用户使用指定的永久性卷声明（ Persistent Volume Claim ， PVC ）配置卷 "存储类" 按名称。详细信息对用户隐藏，但存储类可标识用于该类的配置程序（在本例中为 Trident ）以及该类对配置程序的含义。

创建存储类 Kubernetes 用户将指定何时需要卷。该类的配置需要为上一步创建的后端建模，以便 Astra Trident 可以使用它来配置新卷。

首先要使用的最简单存储类是基于的存储类 `sample-input/storage-class-csi.yaml.templ` 安装程序随附的文件、替换 *`BACKEND_TYPE`* 和存储驱动程序名称。

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

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

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

这是一个Kubernetes对象、因此您可以使用 `kubectl` 以在Kubernetes中创建。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

现在， Kubernetes 和 Astra Trident 都应显示 * 基本 -CSI * 存储类， Astra 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"
        ]
      }
    }
  ]
}
```

第 3 步：配置第一个卷

现在，您已准备好动态配置第一个卷。可通过创建 Kubernetes 来完成此操作 "永久性卷声明" （ PVC ）对象。

为使用刚刚创建的存储类的卷创建 PVC 。

请参见 `sample-input/pvc-basic-csi.yaml` 例如。确保存储类名称与您创建的名称匹配。

```
kubectl create -f sample-input/pvc-basic-csi.yaml

kubectl get pvc --watch
NAME      STATUS    VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS   AGE
basic     Pending
basic          1s
basic     Pending   pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   0
basic          5s
basic     Bound     pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   1Gi
RWO           basic          7s
```

第 4 步：将卷挂载到 Pod 中

现在，让我们挂载卷。我们将启动一个nginx POD、将PV挂载到下 `/usr/share/nginx/html`。

```
cat << EOF > task-pv-pod.yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
EOF
kubectl create -f task-pv-pod.yaml
```

```
# Wait for the pod to start
kubectl get pod --watch

# Verify that the volume is mounted on /usr/share/nginx/html
kubectl exec -it task-pv-pod -- df -h /usr/share/nginx/html

# Delete the pod
kubectl delete pod task-pv-pod
```

此时， Pod （应用程序）不再存在，但卷仍在。如果需要，您可以从另一个 POD 使用它。

要删除卷，请删除声明：

```
kubectl delete pvc basic
```

现在，您可以执行其他任务，例如：

- "配置其他后端。"
- "创建其他存储类。"

管理 Astra Trident

升级 Astra Trident

Astra Trident 遵循季度发布节奏，每个日历年提供四个主要版本。每个新版本都是在先前版本的基础上构建的，可提供新功能和性能增强以及错误修复和改进功能。我们建议您每年至少升级一次，以利用 Astra Trident 中的新功能。

要升级到之前的五个版本，您需要执行多步升级。

确定要升级到的版本

- 您可以升级到 `YY.MM` 从释放 `YY-1.MM` 版本和任何介于两者之间的版本。例如，您可以从 19.07 及更高版本直接升级到 20.07 （包括 DOT 版本，例如 19.07.1 ）。
- 如果您使用的是早期版本，则应执行多步骤升级。这要求您首先升级到适合您的四个版本窗口的最新版本。例如，如果您运行的是 18.07 ，并希望升级到 20.07 版本，请按照以下所述执行多步骤升级过程：
  - 首次从 18.07 升级到 19.07 。有关升级的具体说明，请参见相应版本的文档。
  - 然后从 19.07 升级到 20.07 。

对于19.04及更早版本的所有升级、都需要迁移Astra Trident自己的元数据 `etcd` 到CRD对象。请务必查看此版本的文档，了解升级的工作原理。

升级时、请务必提供 `parameter.fsType` 在中 `StorageClasses` 由Astra Trident使用。您可以删除并重新创建 `StorageClasses` 而不会中断已有卷。这是执行的一项 * 要求 * "安全上下文" SAN 卷。。 "输入示例：" 目录包含示例，例如[`storage-class-basic.yaml.templ`]和[`storage-class-bronze-default.yaml`]。

有关详细信息，请参见 "已知问题"。

我应选择哪种升级路径?

您可以使用以下路径之一进行升级：

- 使用 Trident 运算符。
- 使用 `tridentctl`。

从 Kubernetes 1.20 开始， CSI 卷快照现在是 GA 功能。在升级 Astra Trident 时，在执行升级之前，必须删除所有先前的 alpha snapshot CRS 和 CRD （卷快照类，卷快照和卷快照内容）。请参见 "本博客" 了解将 alpha 快照迁移到测试版 /GA 规范所涉及的步骤。

如果满足以下条件，则可以使用 Trident 操作符进行升级：

- 您正在运行 CSI Trident （ 19.07 及更高版本）。
- 您已安装基于 CRD 的 Trident 版本（ 19.07 及更高版本）。
- 您正在执行自定义安装（使用自定义 YAML ）。

如果您使用的是、请勿使用运算符升级Trident `etcd`基于Trident的版本(19.04或更早版本)。

如果您不想使用此操作员、或者您的自定义安装无法由操作员支持、则可以使用进行升级 `tridentctl`。这是 Trident 19.04 及更早版本的首选升级方法。

对运算符进行了更改

Astra Trident 21.01 版为操作员引入了一些关键的架构变更，即：

- 操作符现在为 * 集群范围 * 。以前的 Trident 运算符实例（版本 20.04 到 20.10 ）为 * 命名空间范围 * 。集群范围内的运算符具有优势，原因如下：
  - 资源责任：操作员现在可以在集群级别管理与 Astra Trident 安装相关的资源。在安装Astra Trident过程中、操作员使用创建和维护多个资源 `ownerReferences`。维护 `ownerReferences` 在集群范围的资源上、某些Kubernetes分销商可能会引发错误、例如OpenShift。使用集群范围的运算符可缓解此问题。对于 Trident 资源的自动修复和修补，这是一项基本要求。
  - 卸载期间清理：要完全删除 Astra Trident ，需要删除所有关联的资源。命名空间范围的运算符可能会在删除集群范围的资源（例如 clusterRole ， ClusterRoleBinding-and PodSecurityPolicy ）时遇到问题，并导致清理不完整。集群范围的运算符可消除此问题描述。用户可以完全卸载 Astra Trident 并在需要时重新安装。
- `TridentProvisioner` 现已替换为 `TridentOrchestrator` 作为用于安装和管理Astra Trident的自定义资源。此外、还会在中引入一个新字段 `TridentOrchestrator` 规格用户可以指定必须使用安装/升级命名空间Trident `spec.namespace` 字段。您可以查看一个示例 "此处"。

了解更多信息

- "使用 Trident 操作符进行升级"
  *

使用操作员升级

您可以使用操作员轻松升级现有的 Astra Trident 安装。

您需要的内容

要使用运算符进行升级，应满足以下条件：

- 您应安装基于 CSI 的 Astra Trident 。要检查是否正在运行 CSI Trident ，请检查 Trident 命名空间中的 Pod 。如果他们遵循 `trident-csi-*` 命名模式下、您正在运行CSI Trident。
- 您应安装基于 CRD 的 Trident 。这表示 19.07 及更高版本的所有版本。如果您安装的是基于 CSI 的安装，则很可能是基于 CRD 的安装。
- 如果您已卸载 CSI Trident ，并且安装中的元数据仍然存在，则可以使用操作员进行升级。
- 在给定 Kubernetes 集群中的所有命名空间中，只应安装一个 Astra Trident 。
- 您应使用运行的 Kubernetes 集群 "支持的Kubernetes版本"。
- 如果存在alpha snapshot CRD、则应使用将其删除 `tridentctl obliviate alpha-snapshot-crd`。此操作将删除 alpha snapshot 规范的 CRD 。有关应删除 / 迁移的现有快照，请参见 "本博客"。

在 OpenShift 容器平台上使用操作符升级 Trident 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，请参见 "GitHub 上的问题描述详细信息"。

升级集群范围的操作员安装

按照以下步骤从* Trident 21.01及更高版本*升级。

步骤

1. 删除用于安装当前 Astra Trident 实例的 Trident 运算符。例如，如果要从 21.01 升级，请运行以下命令：

```
kubectl delete -f 22.01/trident-installer/deploy/BUNDLE.YAML -n trident
```

2. 您也可以编辑 `TridentOrchestrator` 安装Trident时创建的对象、用于修改安装参数。这可能包括修改自定义Trident映像、从中提取容器映像的专用映像注册表、启用调试日志或指定映像提取密钥等更改。
3. 使用适用于您的环境的正确捆绑包YAML文件安装Astra Trident、并从安装Astra Trident版本 `https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/BUNDLE.YAML` 其中：*vXX.XX* 是版本号(例如 `v22.10`)和 *BUNDLE.YAML* 是捆绑包YAML文件名。

   - 对于运行Kubernetes 1.24或更低版本的集群、请使用 "bundle_pre_1_25.yaml"。
   - 对于运行Kubernetes 1.25或更高版本的集群、请使用 "bundle_post_1_25.yaml"。

   例如、如果要为Kubernetes 1.25安装Astra Trident 22.10、请运行以下命令：

```
kubectl create -f 22.10.0/trident-installer/deploy/bundle_post_1_25.yaml
-n trident
```

   在此步骤中、Trident操作员将确定现有的Astra Trident安装并将其升级到与操作员相同的版本。

升级命名空间范围的操作员安装

要从使用命名空间范围的运算符（版本 20.07 至 20.10 ）安装的 Astra Trident 实例进行升级，需要遵循以下步骤：

步骤

1. 验证现有 Trident 安装的状态。要执行此操作、请检查的*状态* `TridentProvisioner`。状态应为 `Installed`。

```
kubectl describe tprov trident -n trident | grep Message: -A 3
Message:  Trident installed
Status:   Installed
Version:  v20.10.1
```

如果状态显示 `Updating`、请确保先解决此问题、然后再继续。有关可能的状态值列表，请参见 "此处"。

2. 创建 `TridentOrchestrator` 使用Trident安装程序随附的清单创建CRD。

```
# Download the release required [22.10.0]
mkdir 22.10.0
cd 22.10.0
wget
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. 使用其清单删除命名空间范围的运算符。要完成此步骤、您需要使用捆绑包YAML文件从部署命名空间范围的运算符 `https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/BUNDLE.YAML` 其中： *`vXX.XX`* 是版本号(例如 `v22.10`)和 *`BUNDLE.YAML`* 是捆绑包YAML文件名。

您应对Trident安装参数进行必要的更改(例如、更改的值 `tridentImage`，`autosupportImage`、私有映像存储库和提供 `imagePullSecrets`)。有关可更新的完整参数列表、请参见 "配置选项"。

```
#Ensure you are in the right directory
pwd
/root/20.10.1/trident-installer

#Delete the namespace-scoped operator
kubectl delete -f deploy/<BUNDLE.YAML>
serviceaccount "trident-operator" deleted
clusterrole.rbac.authorization.k8s.io "trident-operator" deleted
clusterrolebinding.rbac.authorization.k8s.io "trident-operator" deleted
deployment.apps "trident-operator" deleted
podsecuritypolicy.policy "tridentoperatorpods" deleted

#Confirm the Trident operator was removed
kubectl get all -n trident
NAME                                READY   STATUS    RESTARTS   AGE
pod/trident-csi-68d979fb85-dsrmn    6/6     Running   12         99d
pod/trident-csi-8jfhf               2/2     Running   6          105d
pod/trident-csi-jtnjz               2/2     Running   6          105d
pod/trident-csi-lcxvh               2/2     Running   8          105d

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/trident-csi   ClusterIP   10.108.174.125   <none>
34571/TCP,9220/TCP   105d

NAME                           DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR                                        AGE
daemonset.apps/trident-csi   3         3         3       3            3
kubernetes.io/arch=amd64,kubernetes.io/os=linux   105d

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/trident-csi   1/1     1            1           105d

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/trident-csi-68d979fb85   1         1         1
105d
```

在此阶段、将显示 `trident-operator-xxxxxxxxxx-xxxxx` POD已删除。

4. (可选)如果需要修改安装参数、请更新 `TridentProvisioner` 规格这些更改可能包括修改私有映像注册表以从中提取容器映像，启用调试日志或指定映像提取密钥等。

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

5. 安装集群范围的运算符。

安装集群范围的运算符可启动的迁移 `TridentProvisioner` 对象 `TridentOrchestrator` 对象、删除 `TridentProvisioner` 对象和 `tridentprovisioner` CRD、并将Astra Trident升级到所使用的集群范围运算符版本。在以下示例中、Trident会升级到22.10.0。

使用集群范围的运算符升级Astra Trident会导致迁移 `tridentProvisioner` 到A `tridentOrchestrator` 同名对象。此操作由操作员自动处理。在升级过程中， Astra Trident 也会安装在与之前相同的命名空间中。

```
#Ensure you are in the correct directory
pwd
/root/22.10.0/trident-installer

#Install the cluster-scoped operator in the **same namespace**
kubectl create -f deploy/<BUNDLE.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

#All tridentProvisioners will be removed, including the CRD itself
kubectl get tprov -n trident
Error from server (NotFound): Unable to list "trident.netapp.io/v1,
Resource=tridentprovisioners": the server could not find the requested
resource (get tridentprovisioners.trident.netapp.io)

#tridentProvisioners are replaced by tridentOrchestrator
kubectl get torc
NAME      AGE
trident   13s

#Examine Trident pods in the namespace
kubectl get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m41s
trident-csi-xrst8                   2/2     Running   0          1m41s
trident-operator-5574dbbc68-nthjv   1/1     Running   0          1m52s

#Confirm Trident has been updated to the desired version
kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v22.10.0
```

升级基于 **Helm** 的操作员安装

要升级基于 Helm 的操作员安装，请执行以下步骤。

步骤

1. 下载最新的 Astra Trident 版本。
2. 使用 `helm upgrade` 命令：请参见以下示例：

```
helm upgrade <name> trident-operator-22.10.0.tgz
```

其中： `trident-operator-22.10.0.tgz` 反映了要升级到的版本。

3. 运行 `helm list` 验证图表和应用程序版本均已升级。

要在升级期间传递配置数据、请使用 `--set`。

例如、要更改的默认值 `tridentDebug`下，运行以下命令：

```
helm upgrade <name> trident-operator-22.10.0-custom.tgz --set
tridentDebug=true
```

如果您正在运行 `tridentctl logs`、您可以查看调试消息。

如果在初始安装期间设置了任何非默认选项，请确保这些选项包含在 upgrade 命令中，否则，这些值将重置为其默认值。

从非操作员安装升级

如果您的 CSI Trident 实例满足上述前提条件，则可以升级到最新版本的 Trident 操作符。

步骤

1. 下载最新的 Astra Trident 版本。

```
# Download the release required [22.10.0]
mkdir 22.10.0
cd 22.10.0
wget
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.tar.gz
cd trident-installer
```

2. 创建 `tridentorchestrator` 清单中的CRD。

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

3. 部署操作员。

```
#Install the cluster-scoped operator in the **same namespace**
kubectl create -f deploy/<BUNDLE.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-csi-79df798bdc-m79dc        6/6     Running   0          150d
trident-csi-xrst8                   2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv   1/1     Running   0          1m30s
```

4. 创建 `TridentOrchestrator` 安装Astra Trident的CR。

```
#Create a tridentOrchestrator to initate a Trident install
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

#Confirm Trident was upgraded to the desired version
kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v22.10.0
```

现有后端和 PVC 会自动可用。

使用 tridentctl 进行升级

您可以使用轻松升级现有的Astra Trident安装 `tridentctl`。

升级前的注意事项

升级到最新版本的 Astra Trident 时，请考虑以下事项：

- 从 Trident 20.01 开始，仅限测试版 "卷快照" 受支持。Kubernetes 管理员应注意安全地备份或将 alpha Snapshot 对象转换为测试版，以保留原有的 alpha Snapshot 。
- 测试版的卷快照引入了一组经过修改的 CRD 和一个快照控制器，这两个控制器都应在安装 Astra Trident 之前进行设置。 "本博客" 讨论将 alpha 卷快照迁移到测试版格式所涉及的步骤。
- 卸载并重新安装 Astra Trident 可作为升级。卸载 Trident 时，不会删除 Astra Trident 部署所使用的永久性卷声明（ PVC ）和永久性卷（ PV ）。在 Astra Trident 脱机期间，已配置的 PV 仍可用，而 Astra Trident 将在恢复联机后为在此期间创建的任何 PVC 配置卷。

> 升级 Astra Trident 时，请勿中断升级过程。确保安装程序运行完毕。

升级后的后续步骤

要利用较新的Trident版本中提供的丰富功能(例如按需卷快照)、您可以使用升级卷 `tridentctl upgrade` 命令：

如果存在旧卷，则应将其从 NFS/iSCSI 类型升级到 CSI 类型，以便能够使用 Astra Trident 中的一整套新功能。Trident 配置的原有 PV 支持传统功能集。

决定将卷升级到 CSI 类型时，请考虑以下事项：

- 您可能不需要升级所有卷。以前创建的卷将继续可访问并正常运行。
- 升级时， PV 可以作为部署 / 状态集的一部分挂载。不需要关闭部署 / 状态集。
- 升级时，您 * 无法 * 将 PV 连接到独立 POD 。在升级卷之前，您应关闭 POD 。
- 您只能升级绑定到 PVC 的卷。升级前，应删除和导入未绑定到 PVC 的卷。

卷升级示例

以下示例显示了如何执行卷升级。

1. 运行 `kubectl get pv` 列出PV。

```
kubectl get pv
NAME                        CAPACITY     ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                   STORAGECLASS    REASON   AGE
default-pvc-1-a8475         1073741824   RWO            Delete
Bound    default/pvc-1          standard                19h
default-pvc-2-a8486         1073741824   RWO            Delete
Bound    default/pvc-2          standard                19h
default-pvc-3-a849e         1073741824   RWO            Delete
Bound    default/pvc-3          standard                19h
default-pvc-4-a84de         1073741824   RWO            Delete
Bound    default/pvc-4          standard                19h
trident                     2Gi          RWO            Retain
Bound    trident/trident                                19h
```

目前、Trident 20.07使用创建了四个PV `netapp.io/trident` 配置程序。

2. 运行 `kubectl describe pv` 以获取PV的详细信息。

```
kubectl describe pv default-pvc-2-a8486

Name:            default-pvc-2-a8486
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: netapp.io/trident
                 volume.beta.kubernetes.io/storage-class: standard
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Bound
Claim:           default/pvc-2
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        1073741824
Node Affinity:   <none>
Message:
Source:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    10.xx.xx.xx
    Path:      /trid_1907_alpha_default_pvc_2_a8486
    ReadOnly:  false
```

PV是使用创建的 `netapp.io/trident` 配置程序和类型为NFS。要支持 Astra Trident 提供的所有新功能，应将此 PV 升级到 CSI 类型。

3. 运行 `tridenctl upgrade volume <name-of-trident-volume>` 用于将原有Astra Trident卷升级到CSI规范的命令。

```
./tridentctl get volumes -n trident
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
|            NAME     |  SIZE   | STORAGE CLASS | PROTOCOL |
BACKEND UUID             | STATE  | MANAGED |
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-3-a849e | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-1-a8475 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-4-a84de | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+

./tridentctl upgrade volume default-pvc-2-a8486 -n trident
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
|            NAME     |  SIZE   | STORAGE CLASS | PROTOCOL |
BACKEND UUID             | STATE  | MANAGED |
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
+---------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
```

4. 运行 `kubectl describe pv` 验证此卷是否为CSI卷。

```
kubectl describe pv default-pvc-2-a8486
Name:            default-pvc-2-a8486
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: csi.trident.netapp.io
                 volume.beta.kubernetes.io/storage-class: standard
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Bound
Claim:           default/pvc-2
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        1073741824
Node Affinity:   <none>
Message:
Source:
    Type:              CSI (a Container Storage Interface (CSI) volume
source)
    Driver:            csi.trident.netapp.io
    VolumeHandle:      default-pvc-2-a8486
    ReadOnly:          false
    VolumeAttributes:      backendUUID=c5a6f6a4-b052-423b-80d4-
8fb491a14a22

internalName=trid_1907_alpha_default_pvc_2_a8486
                           name=default-pvc-2-a8486
                           protocol=file
Events:                <none>
```

通过这种方式，您可以将由 Astra Trident 创建的 NFS/iSCSI 类型的卷逐个升级到 CSI 类型。

卸载 Astra Trident

根据 Astra Trident 的安装方式，有多种卸载方法。

使用 Helm 卸载

如果您使用Helm安装了Astra Trident、则可以使用将其卸载 `helm uninstall`。

```
#List the Helm release corresponding to the Astra 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
```

使用 **Trident** 操作符卸载

如果您使用操作符安装了 Astra Trident ，则可以通过执行以下操作之一卸载它：

- *编辑 `TridentOrchestrator` 要设置卸载标志：*您可以编辑 `TridentOrchestrator` 并设置 `spec.uninstall=true`。编辑 `TridentOrchestrator` CR并设置 `uninstall` 标记如下所示：

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

  当 `uninstall` 标志设置为 `true`、Trident操作符将卸载Trident、但不会删除Trident Orchestrator本身。如果要重新安装 Trident ，应清理 Trident Orchestrator 并创建新的 Trident 。

- *删除 `TridentOrchestrator`：通过删除 `TridentOrchestrator` cr用于部署Astra Trident、此时您需要指示操作员卸载Trident。操作员将处理的删除操作 `TridentOrchestrator` 然后继续删除Astra Trident部署和取消定义、并删除在安装过程中创建的Trident Pod。要完全删除Astra Trident (包括其创建的CRD)并有效地擦除板、您可以进行编辑 `TridentOrchestrator` 以传递 `wipeout` 选项请参见以下示例：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

这将完全卸载 Astra Trident 并清除与后端及其管理的卷相关的所有元数据。后续安装将视为全新安装。

只有在执行完全卸载时，才应考虑擦除 CRD 。此操作无法撤消。* 除非您希望重新启动并创建全新的 Astra Trident 安装，否则请勿擦除 CRD 。

使用卸载 `tridentctl`

运行 `uninstall` 命令输入 `tridentctl` 如下所示、删除除CRD和相关对象之外与Astra Trident关联的所有资源、从而可以轻松地重新运行安装程序以更新到最新版本。

```
./tridentctl uninstall -n <namespace>
```

要完全删除 Astra Trident ，您应删除由 Astra Trident 创建的 CRD 的最终结果并删除这些 CRD 。

降级 Astra Trident

了解降级到早期版本的 Astra Trident 所涉及的步骤。

何时降级

您可能会考虑降级的各种原因，例如：

- 应急规划
- 立即修复因升级而发现的错误
- 依赖关系问题，升级失败和不完整

迁移到使用 CRD 的 Astra Trident 版本时，应考虑降级。由于Astra Trident使用CRD来保持状态、因此创建的所有存储实体(后端、存储类、PV和卷快照)都具有关联的CRD对象、而不是写入到中的数据 `trident` PV (由早期安装的Astra Trident版本使用)。新创建的 PV ，后端和存储类均作为 CRD 对象进行维护。

请仅尝试对使用CRD运行的Astra Trident版本(19.07及更高版本)进行降级。这样可以确保在降级后可以看到对当前Astra Trident版本执行的操作。

何时不降级

您不应降级到使用的Trident版本 `etcd` 以保持状态(19.04及更早版本)。降级后，使用当前 Astra Trident 版本执行的所有操作都不会反映出来。在回滚到早期版本时，新创建的 PV 不可用。在迁移回早期版本时，对后端，PV ，存储类和卷快照（已创建 / 更新 / 删除）等对象所做的更改对 Astra Trident 不可见。恢复到早期版本不会中断对已使用早期版本创建的 PV 的访问，除非已对其进行升级。

使用操作员安装 **Astra Trident** 时的降级过程

对于使用Trident操作员完成的安装、降级过程有所不同、不需要使用 `tridentctl`。

对于使用 Trident 操作符完成的安装， Astra Trident 可以降级为以下任一项：

- 使用命名空间范围的运算符（ 20.07 - 20.10 ）安装的版本。
- 使用集群范围运算符（ 21.01 及更高版本）安装的版本。

降级为集群范围的运算符

要将 Astra Trident 降级为使用集群范围运算符的版本，请执行以下步骤。

步骤

1. "卸载 Astra Trident"。*除非要完全删除现有安装、否则请勿删除这些CRD
2. 可以使用与您的Trident版本关联的操作员清单来删除Trident运算符。例如：

`https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/bundle.yaml` 其中：*vXX.XX* 是版本号(例如 `v22.10`)和 *bundle.yaml* 是捆绑包YAML文件名。

3. 通过安装所需版本的 Astra Trident 继续降级。按照所需版本的文档进行操作。

降级到命名空间范围的运算符

本节总结了降级到 20.07 到 20.10 范围内的 Astra Trident 版本所涉及的步骤，该版本将使用命名空间范围的运算符进行安装。

步骤

1. "卸载 Astra Trident"。*请勿删除CRD、除非您要完全删除现有安装。*请确保 `tridentorchestrator` 已删除。

```
#Check to see if there are any tridentorchestrators present
kubectl get torc
NAME        AGE
trident     20h

#Looks like there is a tridentorchestrator that needs deleting
kubectl delete torc trident
tridentorchestrator.trident.netapp.io "trident" deleted
```

2. 可以使用与您的Trident版本关联的操作员清单来删除Trident运算符。例如：`https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/bundle.yaml` 其中：*vXX.XX* 是版本号(例如 `v22.10`)和 *bundle.yaml* 是捆绑包YAML文件名。

3. 删除 `tridentorchestrator` CRD。

```
#Check to see if ``tridentorchestrators.trident.netapp.io`` CRD is
present and delete it.

kubectl get crd tridentorchestrators.trident.netapp.io

NAME                                        CREATED AT
tridentorchestrators.trident.netapp.io   2021-01-21T21:11:37Z

kubectl delete crd tridentorchestrators.trident.netapp.io

customresourcedefinition.apiextensions.k8s.io
"tridentorchestrators.trident.netapp.io" deleted
```

已卸载 Astra Trident 。

4. 通过安装所需版本继续降级。按照所需版本的文档进行操作。

使用 **Helm** 降级

要降级、请使用 `helm rollback` 命令：请参见以下示例：

```
helm rollback trident [revision #]
```

使用安装**Astra Trident**时的降级过程 `tridentctl`

如果您使用安装了Astra Trident `tridentctl`、降级过程包括以下步骤。此顺序将指导您完成从 Astra Trident 21.07 迁移到 20.07 的降级过程。

> 在开始降级之前、您应创建Kubernetes集群的快照 `etcd`。这样，您就可以备份 Astra Trident 的 CRD 的当前状态了。

步骤

1. 确保使用安装Trident `tridentctl`。如果您不确定如何安装 Astra Trident ，请运行以下简单测试：
   a. 列出 Trident 命名空间中的 Pod 。
   b. 确定集群中运行的 Astra Trident 的版本。您可以使用 `tridentctl` 或者查看Trident Pod中使用的图像。
   c. 如果您*未看到* A `tridentOrchestrator`、(或) A `tridentprovisioner`、(或)名为的Pod `trident-operator-xxxxxxxxxx-xxxxx`、Astra Trident 已安装 `tridentctl`。
2. 使用现有卸载Astra Trident `tridentctl` 二进制文件。在这种情况下，您将使用 21.07 二进制文件卸载。

```
tridentctl version -n trident
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.07.0        | 21.07.0        |
+----------------+----------------+

tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted Trident daemonset.
INFO Deleted Trident service.
INFO Deleted Trident secret.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Deleted pod security policy.
podSecurityPolicy=tridentpods
INFO The uninstaller did not delete Trident's namespace in case it is
going to be reused.
INFO Trident uninstallation succeeded.
```

3. 完成此操作后，获取所需版本的 Trident 二进制文件（在此示例中为 20.07 ），并使用它安装 Astra Trident 。您可以为生成自定义 YAML "自定义安装" 如果需要，

```
cd 20.07/trident-installer/
./tridentctl install -n trident-ns
INFO Created installer service account.
serviceaccount=trident-installer
INFO Created installer cluster role.                clusterrole=trident-
installer
INFO Created installer cluster role binding.
clusterrolebinding=trident-installer
INFO Created installer configmap.                   configmap=trident-
installer
...
...
INFO Deleted installer cluster role binding.
INFO Deleted installer cluster role.
INFO Deleted installer service account.
```

降级过程已完成。

使用 Astra Trident

准备工作节点

Kubernetes 集群中的所有工作节点都需要能够挂载为 Pod 配置的卷。如果您使用的是 `ontap-nas`， `ontap-nas-economy`或 `ontap-nas-flexgroup` 驱动程序对于您的一个后端、您的工作节点需要NFS工具。否则，它们需要使用 iSCSI 工具。

默认情况下，最新版本的 RedHat CoreOS 同时安装了 NFS 和 iSCSI 。

安装 NFS 或 iSCSI 工具后，您应始终重新启动工作节点，否则将卷连接到容器可能会失败。

节点服务发现

从22.07开始、Astra Trident会尝试自动检测节点是否能够运行iSCSI或NFS服务。Astra Trident会为节点创建事件以标识发现的服务。您可以使用命令查看这些事件：

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

Trident还标识为Trident节点CR上的每个节点启用的服务。要查看发现的服务、请使用命令：

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

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。

NFS volumes

协议	操作系统	命令
NFS	RHEL/CentOS 7.	`sudo yum install -y nfs-utils`
NFS	Ubuntu	`sudo apt-get install -y nfs-common`

您应确保 NFS 服务在启动期间启动。

iSCSI 卷

使用 iSCSI 卷时，请考虑以下事项：

- Kubernetes 集群中的每个节点都必须具有唯一的 IQN 。* 这是必要的前提条件 * 。
- 如果使用RHCOS 4.5或更高版本或其他与RHEL兼容的Linux分发版、请与结合使用 `solidfire-san` 驱动程序和Element OS 12：5或更早版本、请确保中的CHAP身份验证算法设置为MD5 `/etc/iscsi/iscsid.conf`。Element 12.7提供了符合FIPS的安全CHAP算法SHA1、SHA-256和SHA3-256。

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

- 在使用运行RHEL/RedHat CoreOS和iSCSI PV的工作节点时、请务必指定 `discard` StorageClass中的mountOption、用于执行实时空间回收。请参见 "RedHat 的文档"。

协议	操作系统	命令

iSCSI	RHEL/CentOS	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` 下 `defaults`。
		
		5. 请确保 `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`

协议	操作系统	命令

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` 下 `defaults`。
		
		5. 请确保 `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`

对于Ubuntu 18.04、您必须使用发现目标端口 `iscsiadm` 启动前 `open-iscsi` 以启动iSCSI守护进程。您也可以修改 `iscsi` 要启动的服务 `iscsid` 自动。

```
sudo systemctl status open-iscsi
```

配置后端

后端定义了 Astra Trident 与存储系统之间的关系。它告诉 Astra Trident 如何与该存储系统进行通信，以及 Astra Trident 如何从该存储系统配置卷。Astra Trident 将自动从后端提供符合存储类定义的要求的存储池。了解有关根据您拥有的存储系统类型配置后端的更多信息。

- "配置 Azure NetApp Files 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 Astra Trident 与适用于 NetApp ONTAP 的 Amazon FSX 结合使用"

配置 **Azure NetApp Files** 后端

您可以将Azure NetApp Files (ANF)配置为Astra Trident的后端。您可以使用ANF后端连接NAS和SMB卷。

- "准备"
- "配置选项和示例"

注意事项

- Azure NetApp Files 服务不支持小于100 GB的卷。如果请求的卷较小，则 Astra Trident 会自动创建 100 GB 的卷。
- Astra Trident仅支持将SMB卷挂载到Windows节点上运行的Pod。
- Astra Trident不支持Windows ARM 架构。

准备配置**Azure NetApp Files** 后端

在配置ANF后端之前、您需要确保满足以下要求。

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。

- 要设置Azure NetApp Files 并创建NFS卷、请参见 "Azure：设置Azure NetApp Files 并创建NFS卷"。
- 要配置Azure NetApp Files 并添加SMB卷、请参见： "Azure：为Azure NetApp Files 创建SMB卷"。

要求

配置和使用 "Azure NetApp Files" 后端，您需要满足以下要求：

- `subscriptionID` 从启用了Azure NetApp Files 的Azure订阅。
- `tenantID`， `clientID`，和 `clientSecret` 从 "应用程序注册" 在 Azure Active Directory 中，具有足够的

Azure NetApp Files 服务权限。应用程序注册应使用以下任一项：

- 所有者或贡献者角色 "由Azure预定义"
- 答 "自定义贡献者角色" 订阅级别 (`assignableScopes`)、并具有以下权限、这些权限仅限于Astra Trident所需的权限。创建自定义角色后、 "使用Azure门户分配角色"。

```
{
    "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/writ
e",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/dele
te",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/rea
d",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/wri
te",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/del
```

```
ete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/Get
Metadata/action",

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

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/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": []
            }
        ]
    }
}
```

- Azure `location` 至少包含一个 "委派子网"。自Trident 22.01日开始 `location` 参数是后端配置文件顶层的必填字段。在虚拟池中指定的位置值将被忽略。

SMB卷的其他要求

- 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。Astra Trident仅支持将SMB卷挂载到Windows节点上运行的Pod。
- 至少有一个包含Active Directory凭据的Astra Trident密钥、以便ANF可以向Active Directory进行身份验证。以生成密钥 `smbcreds`：

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

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

Azure NetApp Files 后端配置选项和示例

了解ANF的NFS和SMB后端配置选项、并查看配置示例。

Astra Trident会使用后端配置(子网、虚拟网络、服务级别和位置)在请求的位置提供的容量池上创建ANF卷、并与请求的服务级别和子网匹配。

Astra Trident 不支持手动 QoS 容量池。

后端配置选项

ANF后端提供了这些配置选项。

参数	Description	Default
`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`	Azure 订阅中的订阅 ID	
`tenantID`	应用程序注册中的租户 ID	
`clientID`	应用程序注册中的客户端 ID	
`clientSecret`	应用程序注册中的客户端密钥	
`serviceLevel`	其中一个 `Standard`， `Premium`` 或 `Ultra`	"" （随机）
`location`	要创建新卷的 Azure 位置的名称	
`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	Description	Default
networkFeatures	一个卷的一组vNet功能可能是 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。指定 networkFeatures 如果未启用此功能、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、请包括 nfsvers=4 在逗号分隔的挂载选项列表中选择NFS v4.1。存储类定义中设置的挂载选项会覆盖后端配置中设置的挂载选项。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小超过此值，则配置失败	"" （默认情况下不强制实施）
debugTraceFlags	故障排除时要使用的调试标志。示例、 \{"api": false, "method": true, "discovery": true}。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	空
nasType	配置NFS或SMB卷创建。选项包括 nfs， smb 或为空。默认情况下、将设置为空会将NFS卷设置为空。	nfs

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

所需权限和资源

如果在创建PVC时收到"未找到容量池"错误、则您的应用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、则Astra Trident将记录创建后端时发现的Azure资源。验证是否正在使用适当的角色。

的值 resourceGroups， netappAccounts， capacityPools， virtualNetwork，和 subnet 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 值是指筛选器、用于将发现的一组资源限制为此存储后端可用的资源、并且可以以任意组合方式指定。完全限定名称采用以下格式：

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

卷配置

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

参数	Description	Default
`exportRule`	新卷的导出规则。`exportRule` 必须是以CIDR表示法表示的任意IPv4地址或IPv4子网组合的逗号分隔列表。SMB卷已忽略。	"0.0.0.0/0
`snapshotDir`	控制 .snapshot 目录的可见性	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）

对于在ANF后端创建的所有卷、Astra Trident会在配置存储池时将存储池上的标签复制到该存储卷。存储管理员可以为每个存储池定义标签，并对存储池中创建的所有卷进行分组。这是一种根据后端配置中提供的一组可自定义标签区分卷的便捷方式。

示例配置

示例 **1**： 最低配置

这是绝对的最低后端配置。使用此配置， Astra Trident 会发现在已配置位置委派给 ANF 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。因为 `nasType` 省略 `nfs` 默认情况下适用、后端将为NFS卷配置。

当您刚开始使用 ANF 并尝试执行相关操作时，此配置是理想的选择，但实际上，您希望为所配置的卷提供更多范围界定。

```
{
    "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"
}
```

示例 **2**： 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中 `eastus` 位置 `Ultra` 容量池。Astra Trident 会自动发现该位置委派给 ANF 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"
],
    }
```

示例 **3**： 高级配置

此后端配置进一步将卷放置范围缩小为一个子网，并修改了某些卷配置默认值。

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

示例 **4**： 虚拟存储池配置

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。

```
{
    "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"
    },
    "location": "eastus",

    "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"]
        }
    ]
}
```

以下内容 `StorageClass` 定义是指上述存储池。

使用的示例定义 `parameter.selector` 字段

使用 `parameter.selector` 您可以为每个指定 `StorageClass` 用于托管卷的虚拟池。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。

示例**1**：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"
```

示例**2**：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}
```

示例**3**：每个卷使用不同的密钥

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

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

创建后端

创建后端配置文件后，运行以下命令：

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

如果后端创建失败，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。

为 GCP 后端配置 CVS

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

了解适用于GCP的CVS的Astra Trident支持

Astra Trident支持默认CVS服务类型为on的卷 "GCP"。无论CVS服务类型允许的最小值如何、Astra Trident都不支持小于100 GiB的CVS卷。因此、如果请求的卷小于最小大小、Trident会自动创建100 GiB卷。

您需要的内容

以配置和使用 "适用于 Google Cloud 的 Cloud Volumes Service" 后端，您需要满足以下要求：

- 配置了 NetApp CVS 的 Google Cloud 帐户
- Google Cloud 帐户的项目编号
- Google Cloud服务帐户 `netappcloudvolumes.admin` role
- CVS 服务帐户的 API 密钥文件

后端配置选项

有关后端配置选项，请参见下表：

参数	Description	Default
`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

参数	Description	Default
`storageClass`	存储类型选择选项 `hardware` (性能优化)或 `software` (CVS服务类型)	
`projectNumber`	Google Cloud 帐户项目编号。该值可在 Google Cloud 门户的主页页面上找到。	
`apiRegion`	CVS 帐户区域。后端将在该区域配置卷。	
`apiKey`	Google Cloud服务帐户的API密钥 `netappcloudvolumes.admin` 角色。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 服务级别。这些值包括 "standard" ， "premer" 和 "Extreme " 。	标准
`network`	用于CVS卷的GCP网络	default
`debugTraceFlags`	故障排除时要使用的调试标志。示例、 `\{"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

如果使用共享VPC网络、则同时使用这两者 `projectNumber` 和 `hostProjectNumber` 必须指定。在这种情况下、 `projectNumber` 是服务项目、和 `hostProjectNumber` 是主机项目。

。 `apiRegion` 表示Astra Trident创建CVS卷的GCP区域。创建跨区域Kubernetes集群时、在中创建的CVS卷 `apiRegion` 可用于在多个GCP区域的节点上计划的工作负载。请注意，跨区域流量会产生额外成本。

- 要启用跨区域访问、请定义StorageClass `allowedTopologies` 必须包括所有地区。例如：

```
- key: topology.kubernetes.io/region
  values:
  - us-east1
  - europe-west1
```

- `storageClass` 是一个可选参数、可用于选择所需的 "CVS 服务类型"。您可以从基本CVS服务类型中进行选择 (`storageClass=software`)或CVS-Performance服务类型 (`storageClass=hardware`)、Trident默认使用此选项。请确保指定 `apiRegion` 提供相应的CVS `storageClass` 在后端定义中。

Astra Trident 与 Google Cloud 上的基本 CVS 服务类型集成是一项 * 测试版功能 * ，不适用于生产工作负载。在 CVS-Performance 服务类型中， Trident 是 "* 完全支持 "* ，默认情况下会使用它。

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。

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配置每个卷的方式。请参见以下配置示例。

参数	Description	Default
`exportRule`	新卷的导出规则	"0.0.0.0/0
`snapshotDir`	访问 `.snapshot` 目录	false
`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	"100Gi"

。 `exportRule` 值必须是以CIDR表示法表示的IPv4地址或IPv4子网任意组合的逗号分隔列表。

对于在 CVS Google Cloud 后端创建的所有卷， Trident 会在配置存储池时将其上的所有标签复制到该存储卷。存储管理员可以为每个存储池定义标签，并对存储池中创建的所有卷进行分组。这样，您就可以根据后端配置中提供的一组可自定义标签来方便地区分卷了。

示例 **1** ： 最低配置

这是绝对的最低后端配置。

```
{
    "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-----\n",
        "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：基本 CVS 服务类型配置

此示例显示了使用基本 CVS 服务类型的后端定义，该服务类型适用于通用工作负载，可提供轻 / 中性能以及高区域可用性。

```
{
    "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-----\n",
        "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 **3**： 单服务级别配置

此示例显示了一个后端文件，该文件对 Google Cloud us-west2 区域中由 Astra Trident 创建的所有存储应用相同的方面。此示例还显示了的使用情况 `proxyURL` 在后端配置文件中。

```
{
    "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-----\n",
        "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 **4**： 虚拟存储池配置

此示例显示了使用虚拟存储池和配置的后端定义文件 `StorageClasses` 这是指它们。

在下面所示的示例后端定义文件中、为所有存储池设置了特定的默认值、这些存储池设置了 `snapshotReserve` 5%和 `exportRule` 到0.0.0.0/0。虚拟存储池在中进行定义 `storage` 部分。在此示例中、每个存储池都设置了自己的存储池 `serviceLevel`、并且某些池会覆盖默认值。

```
{
```

```
    "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-----\n",
        "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"
        }
    ]
}
```

以下 StorageClass 定义引用了上述存储池。使用 `parameters.selector` 字段中、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。

第一个StorageClass (`cvs-extreme-extra-protection`)映射到第一个虚拟存储池。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。Astra Trident 决定选择哪个虚拟存储池，并确保满足快照预留要求。

```
apiVersion: storage.k8s.io/v1
```

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

```
metadata:
  name: cvs-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "protection=extra"
allowVolumeExpansion: true
```

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 Astra Trident 安装中创建和使用 Element 后端。

您需要的内容

- 运行 Element 软件的受支持存储系统。
- NetApp HCI/SolidFire 集群管理员或租户用户的凭据，可用于管理卷。
- 所有 Kubernetes 工作节点都应安装适当的 iSCSI 工具。请参见 "工作节点准备信息"。

您需要了解的信息

。 `solidfire-san` 存储驱动程序支持两种卷模式：文件和块。。 `Filesystem` volumemode、Astra Trident 会创建卷并创建文件系统。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	无文件系统。原始块设备。
`solidfire-san`	iSCSI	块	rwo ， rox ， rwx	无文件系统。原始块设备。
`solidfire-san`	iSCSI	文件系统	工单， ROX	`xfs， ext3， ext4`
`solidfire-san`	iSCSI	文件系统	工单， ROX	`xfs， ext3， ext4`

Astra Trident 在用作增强型 CSI 配置程序时使用 CHAP 。如果您使用的是 CHAP （这是 CSI 的默认设置），则无需进行进一步准备。建议显式设置 `UseCHAP` 可选择对非CSI Trident使用CHAP。否则，请参见 "此处"。

只有适用于 Astra Trident 的传统非 CSI 框架才支持卷访问组。如果配置为在 CSI 模式下运行，则 Astra Trident 将使用 CHAP 。

如果两者都不是 `AccessGroups` 或 `UseCHAP` 设置后、将应用以下规则之一：

- 如果为默认值 `trident` 检测到访问组、使用访问组。
- 如果未检测到访问组，并且 Kubernetes 版本为 1.7 或更高版本，则会使用 CHAP 。

后端配置选项

有关后端配置选项，请参见下表：

参数	Description	Default
`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	
`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	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

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

对于创建的所有卷， Astra Trident 会在配置存储池时将存储池上的所有标签复制到备用存储 LUN 。存储管理员可以为每个存储池定义标签，并对存储池中创建的所有卷进行分组。这样，您就可以根据后端配置中提供的一组可自定义标签来方便地区分卷了。

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

此示例显示了一个后端文件，该文件使用 CHAP 身份验证并使用特定 QoS 保证对三种卷类型进行建模。然后、您很可能会使用定义存储类来使用其中的每一种 `IOPS` storage class参数。

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

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

此示例显示了使用虚拟存储池配置的后端定义文件以及引用这些池的 StorageClasses 。

在下面所示的示例后端定义文件中、为所有存储池设置了特定的默认值、这些存储池设置了 `type` 在Silver。虚拟存储池在中进行定义 `storage` 部分。在此示例中，某些存储池设置了自己的类型，而某些池将覆盖上述默认值。

```
{
    "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存储池。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。卷将在选定虚拟池中定义各个方面。

第一个StorageClass (`solidfire-gold-four`)将映射到第一个虚拟存储池。这是唯一一个可通过提供金牌性

能的池 `Volume Type QoS` 金牌。最后一个StorageClass (`solidfire-silver`)调用提供银牌性能的任何存储池。Astra Trident 将决定选择哪个虚拟存储池，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"

使用 ONTAP SAN 驱动程序配置后端

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

- "准备"
- "配置和示例"

用户权限

Astra Trident应以ONTAP 或SVM管理员身份运行、通常使用 `admin` 集群用户或 `vsadmin` SVM用户或具有相同角色的其他名称的用户。对于适用于NetApp ONTAP 的Amazon FSX部署、Astra Trident应使用集群以ONTAP 或SVM管理员身份运行 `fsxadmin` 用户或 `vsadmin` SVM用户或具有相同角色的其他名称的用户。。 `fsxadmin` 用户是集群管理员用户的有限替代用户。

如果您使用 `limitAggregateUsage` 参数、需要集群管理员权限。在将适用于NetApp ONTAP 的Amazon FSx与Astra Trident结合使用时、会显示 `limitAggregateUsage` 参数不适用于 `vsadmin` 和 `fsxadmin` 用户帐户。如果指定此参数，配置操作将失败。

虽然可以在 ONTAP 中创建 Trident 驱动程序可以使用的限制性更强的角色，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 SAN**驱动程序配置后端

了解如何准备使用 ONTAP SAN 驱动程序配置 ONTAP 后端。对于所有 ONTAP 后端， Astra Trident 需要至少为 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、您可以配置 `san-dev` 使用的类 `ontap-san` 驱动程序和A `san-default` 使用的类 `ontap-san-economy` 一个。

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

身份验证

Astra 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： Astra 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

Astra 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、例如 `admin` 或 `vsadmin`。这样可以确保与未来的 ONTAP 版本向前兼容，这些版本可能会使功能 API 公开供未来的 Astra Trident 版本使用。可以创建自定义安全登录角色并将其用于 Astra Trident ，但不建议使用。

后端定义示例如下所示：

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
}
```

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=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
```

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",
"dataLIF": "1.2.3.8",
"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、使用更新后的backend.json文件、该文件包含要执行的所需参数 `tridentctl backend update`。

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "secret",
"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。成功的后端更新表明， Astra Trident 可以与 ONTAP 后端进行通信并处理未来的卷操作。

指定 **igroup**

Astra Trident 使用 igroup 来控制对其配置的卷（ LUN ）的访问。在为后端指定 igroup 时，管理员有两种选择：

- Astra Trident 可以自动为每个后端创建和管理 igroup 。条件 `igroupName` 不包含在后端定义中、Astra Trident将创建一个名为的igroup `trident-<backend-UUID>` 在 SVM 上。这将确保每个后端都有一个专用的 igroup ，并处理 Kubernetes 节点 IQN 的自动添加 / 删除。
- 或者，也可以在后端定义中提供预先创建的 igroup 。可以使用完成此操作 `igroupName` config参数。Astra Trident 会将 Kubernetes 节点 IQN 添加 / 删除到已有的 igroup 中。

用于具有的后端 `igroupName` 定义的、 `igroupName` 可以使用删除 `tridentctl backend update` 使用Astra Trident自动处理igroup。这样不会中断对已连接到工作负载的卷的访问。未来的连接将使用创建的

igroup Astra Trident 进行处理。

> 为 Astra Trident 的每个唯一实例指定一个 igroup 是一个最佳实践，对 Kubernetes 管理员和存储管理员都很有用。CSI Trident 可自动向 igroup 添加和删除集群节点 IQN ，从而极大地简化了其管理。在 Kubernetes 环境（以及 Astra Trident 安装）中使用相同的 SVM 时，使用专用的 igroup 可确保对一个 Kubernetes 集群所做的更改不会影响与另一个 Kubernetes 集群关联的 igroup 。此外，还必须确保 Kubernetes 集群中的每个节点都具有唯一的 IQN 。如上所述，Astra Trident 会自动处理 IQN 的添加和删除。在多个主机之间重复使用 IQN 可能会导致出现主机相互错误并拒绝访问 LUN 的不希望出现的情况。

如果将 Astra Trident 配置为充当 CSI 配置程序，则 Kubernetes 节点 IQN 会自动添加到 igroup 中或从 igroup 中删除。将节点添加到Kubernetes集群后、 `trident-csi` DemonSet部署POD (`trident-csi-xxxxx`)、并注册可将卷连接到的新节点。节点 IQN 也会添加到后端的 igroup 中。在对节点进行隔离，清空并从 Kubernetes 中删除时，可以执行一组类似的步骤来删除 IQN 。

如果 Astra Trident 未作为 CSI 配置程序运行，则必须手动更新 igroup ，以包含 Kubernetes 集群中每个工作节点的 iSCSI IQN 。需要将加入 Kubernetes 集群的节点的 IQN 添加到 igroup 中。同样，必须从 igroup 中删除从 Kubernetes 集群中删除的节点的 IQN 。

使用双向 **CHAP** 对连接进行身份验证

Astra Trident可以使用双向CHAP对iSCSI会话进行身份验证 `ontap-san` 和 `ontap-san-economy` 驱动程序。这需要启用 `useCHAP` 选项。设置为时 `true`、Astra Trident会将SVM的默认启动程序安全性配置为双向CHAP、并从后端文件设置用户名和密码。NetApp 建议使用双向 CHAP 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": "FaKePaSsWoRd",
    "igroupName": "trident",
    "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 。

此外 `useCHAP=true`， `chapInitiatorSecret`， `chapTargetInitiatorSecret`， `chapTargetUsername`，和 `chapUsername` 后端定义中必须包含字段。在创建后端后、可以运行来更改这些密码 `tridentctl update`。

工作原理

通过设置 `useCHAP` 为true、存储管理员指示Astra 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*卷中没有已存在的LUN、则Astra Trident会将默认安全类型设置为 `CHAP` 然后继续配置CHAP启动程序以及目标用户名和密码。
  - 如果 SVM 包含 LUN ，则 Astra Trident 不会在 SVM 上启用 CHAP 。这样可以确保对 SVM 上已存在的 LUN 的访问不受限制。
- 配置 CHAP 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）。
- 管理向添加启动程序的操作 `igroupName` 在后端提供。如果未指定、则默认为 `trident`。

创建后端后、Astra Trident将创建相应的 `tridentbackend` CRD并将CHAP密钥和用户名存储为Kubernetes密钥。此后端由 Astra Trident 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

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

> 更新后端的CHAP密码时、必须使用 `tridentctl` 更新后端。请勿通过 CLI/ONTAP UI 更新存储集群上的凭据，因为 Astra Trident 将无法选取这些更改。

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

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

现有连接将不受影响；如果凭据由 SVM 上的 Astra Trident 更新，则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

了解如何在您的 Astra Trident 安装中创建和使用 ONTAP SAN 驱动程序。本节提供了后端配置示例以及有关如何将后端映射到 StorageClasses 的详细信息。

后端配置选项

有关后端配置选项，请参见下表：

参数	Description	Default
`version`		始终为 1

参数	Description	Default
`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地址要进行无缝MetroCluster 切换、必须指定SVM管理LIF。	"10.0.0.1 "， "2001： 1234：abcd：：： fefe] "
`dataLIF`	协议 LIF 的 IP 地址。对于 IPv6，请使用方括号。设置后无法更新	由 SVM 派生，除非另有说明
`useCHAP`	使用 CHAP 对 iSCSI 的 ONTAP SAN 驱动程序进行身份验证 [布尔值]	false
`chapInitiatorSecret`	CHAP 启动程序密钥。如果为、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。如果为、则为必需项 `useCHAP=true`	"
`chapUsername`	入站用户名。如果为、则为必需项 `useCHAP=true`	"
`chapTargetUsername`	目标用户名。如果为、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	"
`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	"
`svm`	要使用的 Storage Virtual Machine	如果是SVM、则派生 `managementLIF` 已指定
`igroupName`	要使用的 SAN 卷的 igroup 的名称	"trident — < 后端 UUID >"
`storagePrefix`	在 SVM 中配置新卷时使用的前缀。设置后无法更新	Trident
`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	Description	Default
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）
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。示例 ｛"api"： false ， "method "： true ｝	空
useREST	用于使用 ONTAP REST API 的布尔参数。MetroCluster 不支持*技术预览*。	false

<code>useREST</code>注意事项

- useREST 作为一个*技术预览版提供、建议用于测试环境、而不是生产工作负载。设置为时 true、Astra Trident将使用ONTAP REST API与后端进行通信。此功能需要使用ONTAP 9.10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。这一点可通过预定义来满足 vsadmin 和 cluster-admin 角色。
- useREST MetroCluster 不支持。

要与 ONTAP 集群通信，您应提供身份验证参数。这可以是安全登录的用户名 / 密码，也可以是已安装的证书。

如果您使用适用于NetApp ONTAP 后端的Amazon FSX、请勿指定 limitAggregateUsage 参数。。 fsxadmin 和 vsadmin Amazon FSX for NetApp ONTAP 提供的角色不包含检索聚合使用情况并通过Astra Trident限制聚合使用情况所需的访问权限。

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

。 ontap-san 驱动程序、默认情况下使用SVM中的所有数据LIF IP并使用iSCSI多路径。为的数据LIF指定IP地址 ontap-san 驱动程序会强制它们禁用多路径并仅使用指定的地址。

创建后端时、请记住这一点 dataLIF 和 storagePrefix 创建后无法修改。要更新这些参数，您需要创建一个新的后端。

igroupName 可以设置为已在ONTAP 集群上创建的igroup。如果未指定，则 Astra Trident 会自动创建一个名为 trident -<backender-UUUUUID> 的 igroup 。如果要在环境之间共享 SVM ，则如果提供预定义的 igroupName ， NetApp 建议为每个 Kubernetes 集群使用一个 igroup 。这对于 Astra Trident 自动保持 IQN 添加 / 删除是必需的。

后端也可以在创建后更新 igroup ：

- 可以更新 igroupName 以指向在 Astra Trident 之外的 SVM 上创建和管理的新 igroup 。
- 可以省略 igroupName 。在这种情况下， Astra Trident 将自动创建和管理 trident -<backend-UUUUID> igroup 。

在这两种情况下，仍可访问卷附件。未来的卷附件将使用更新后的 igroup 。此更新不会中断对后端卷的访问。

可以为指定完全限定域名(FQDN) managementLIF 选项

`managementLIF` 对于所有ONTAP 驱动程序、也可以设置为IPv6地址。确保将Trident与一起安装 `--use-ipv6` 标志。必须谨慎定义 `managementLIF` 方括号内的IPv6地址。

使用IPv6地址时、请确保 managementLIF 和 dataLIF (如果包含在后端定义中)在方括号内进行定义、例如、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]。条件 dataLIF 如果未提供、则Astra Trident将从SVM提取IPv6数据LIF。

要使ontap-san驱动程序能够使用CHAP、请设置 useCHAP 参数设置为 true 在后端定义中。然后， Astra Trident 将配置双向 CHAP 并将其用作后端给定 SVM 的默认身份验证。请参见 "此处" 了解其工作原理。

。 ontap-san-economy 驱动程序、 limitVolumeSize 选项还会限制它所管理的qtree和LUN卷的最大大小。

Astra Trident会在使用创建的所有卷的"Comments"字段中设置配置标签 ontap-san 驱动程序。对于创建的每个卷， FlexVol 上的 "Comments" 字段将使用其所在存储池上的所有标签填充。存储管理员可以为每个存储池定义标签，并对存储池中创建的所有卷进行分组。这样，您就可以根据后端配置中提供的一组可自定义标签来方便地区分卷了。

用于配置卷的后端配置选项

您可以在配置的特殊部分中使用这些选项来控制默认配置每个卷的方式。有关示例，请参见以下配置示例。

参数	Description	Default
spaceAllocation	LUN 的空间分配	true
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	"
snapshotReserve	为快照预留的卷百分比为 "0"	条件 snapshotPolicy 为"无"、否则为""
splitOnClone	创建克隆时，从其父级拆分该克隆	false
splitOnClone	创建克隆时，从其父级拆分该克隆	false

参数	Description	Default
`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Astra Trident中配置的任何卷都将启用NAE。有关详细信息、请参见： "Astra Trident如何与NVE和NAE配合使用"。	false
`luksEncryption`	启用LUKS加密。请参见 "使用Linux统一密钥设置(LUKS)"。	""
`securityStyle`	新卷的安全模式	"unix"
`tieringPolicy`	使用 " 无 " 的分层策略	适用于 ONTAP 9.5 SVM-DR 之前的配置的 " 仅快照 "

在 Astra Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。建议使用非共享 QoS 策略组，并确保策略组分别应用于每个成分卷。共享 QoS 策略组将对所有工作负载的总吞吐量实施上限。

下面是定义了默认值的示例：

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "managementLIF": "10.0.0.1",
 "dataLIF": "10.0.0.2",
 "svm": "trident_svm",
 "username": "admin",
 "password": "password",
 "labels": {"k8scluster": "dev2", "backend": "dev2-sanbackend"},
 "storagePrefix": "alternate-trident",
 "igroupName": "custom",
 "debugTraceFlags": {"api":false, "method":true},
 "defaults": {
     "spaceReserve": "volume",
     "qosPolicy": "standard",
     "spaceAllocation": "false",
     "snapshotPolicy": "default",
     "snapshotReserve": "10"
 }
}
```

用于使用创建的所有卷 `ontap-san` 驱动程序、Astra 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。Astra Trident 将 FlexVol 增加 10% （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.

用于定义的后端 `snapshotReserve`、Astra Trident将按如下所示计算卷大小：

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

1.1 是 Astra Trident 向 FlexVol 额外添加 10% 以容纳 LUN 元数据。适用于 `snapshotReserve` = 5%、PVC请求= 5GiB、卷总大小为5.79GiB、可用大小为5.5GiB。。 `volume show` 命令应显示与以下示例类似的结果：

目前，调整大小是对现有卷使用新计算的唯一方法。

最低配置示例

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

如果您正在将 NetApp ONTAP 上的 Amazon FSx 与 Astra Trident 结合使用，建议为 LIF 指定 DNS 名称，而不是 IP 地址。

`ontap-san` 具有基于证书的身份验证的驱动程序

这是一个最低后端配置示例。 `clientCertificate`， `clientPrivateKey`，和 `trustedCACertificate` (如果使用可信CA、则可选)将填充 `backend.json` 和分别采用客户端证书、专用密钥和可信CA证书的base64编码值。

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "DefaultSANBackend",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
    "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
    "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

`ontap-san` 具有双向**CHAP**的驱动程序

这是一个最低后端配置示例。此基本配置将创建 `ontap-san` 后端 `useCHAP` 设置为 `true`。

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "labels": {"k8scluster": "test-cluster-1", "backend": "testcluster1-
sanbackend"},
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

`ontap-san-economy` 驱动程序

```
{
    "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",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

虚拟存储池后端示例

在下面所示的示例后端定义文件中、会为所有存储池设置特定的默认值、例如 `spaceReserve` 无、`spaceAllocation` 为false、和 `encryption` 为false。虚拟存储池在存储部分中进行定义。

在此示例中、某些存储池会设置自己的存储池 `spaceReserve`， `spaceAllocation`，和 `encryption` 值、而某些池会覆盖上述设置的默认值。

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret",

    "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"
            }
        }
    ]
}
```

以下是的iSCSI示例 `ontap-san-economy` 驱动程序：

```
{
    "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",
    "igroupName": "trident",
    "username": "vsadmin",
```

```
    "password": "secret",

    "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"
            }
        }
    ]
}
```

将后端映射到 **StorageClasses**

以下 StorageClass 定义引用了上述虚拟存储池。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。卷将在选定虚拟池中定义各个方面。

- 第一个StorageClass (`protection-gold`)将映射到中的第一个、第二个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第一个虚拟存储池 `ontap-san` 后端。这是唯一一个提供黄金级保护的池。
- 第二个StorageClass (`protection-not-gold`)将映射到中的第三个、第四个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第二个、第三个虚拟存储池 `ontap-san` 后端。这些池是唯一提供黄金级以外保护级别的池。

- 第三个StorageClass (`app-mysqldb`)将映射到中的第四个虚拟存储池 `ontap-nas` 中的后端和第三个虚拟存储池 `ontap-san-economy` 后端。这些池是唯一为 mysqldb 类型的应用程序提供存储池配置的池。
- 第四个StorageClass (`protection-silver-creditpoints-20k`)将映射到中的第三个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第二个虚拟存储池 `ontap-san` 后端。这些池是唯一以 20000 个信用点提供黄金级保护的池。
- 第五个StorageClass (`creditpoints-5k`)将映射到中的第二个虚拟存储池 `ontap-nas-economy` 中的后端和第三个虚拟存储池 `ontap-san` 后端。这些是唯一一款具有 5000 个信用点的池产品。

Astra Trident 将决定选择哪个虚拟存储池，并确保满足存储要求。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

配置ONTAP NAS后端

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

- "准备"
- "配置和示例"

> 客户必须使用 `ontap-nas` 适用于需要数据保护、灾难恢复和移动性的生产工作负载的驱动程序。Astra Control可为使用创建的卷提供无缝保护、灾难恢复和移动性 `ontap-nas` 驱动程序。。 `ontap-nas-economy` 驱动程序仅适用于预期卷使用量远高于ONTAP 支持的有限使用情形、并且没有预期的数据保护、灾难恢复或移动性(在Kubernetes集群之间移动卷)要求。

用户权限

Astra Trident应以ONTAP 或SVM管理员身份运行、通常使用 `admin` 集群用户或 `vsadmin` SVM用户或具有相同角色的其他名称的用户。对于适用于NetApp ONTAP 的Amazon FSX部署、Astra Trident应使用集群以ONTAP 或SVM管理员身份运行 `fsxadmin` 用户或 `vsadmin` SVM用户或具有相同角色的其他名称的用户。。 `fsxadmin` 用户是集群管理员用户的有限替代用户。

> 如果您使用 `limitAggregateUsage` 参数、需要集群管理员权限。在将适用于NetApp ONTAP 的Amazon FSx与Astra Trident结合使用时、会显示 `limitAggregateUsage` 参数不适用于 `vsadmin` 和 `fsxadmin` 用户帐户。如果指定此参数，配置操作将失败。

虽然可以在 ONTAP 中创建 Trident 驱动程序可以使用的限制性更强的角色，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

了解如何准备使用 ONTAP NAS 驱动程序配置 ONTAP 后端。对于所有 ONTAP 后端， Astra Trident 需要至少为 SVM 分配一个聚合。

对于所有 ONTAP 后端， Astra Trident 需要至少为 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、您可以配置一个使用的黄金类 `ontap-nas` 驱动程序和使用的铜牌类 `ontap-nas-economy` 一个。

所有Kubernetes工作节点都必须安装适当的NFS工具。请参见 "此处" 有关详细信息：

身份验证

Astra 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： Astra 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

Astra 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、例如 `admin` 或 `vsadmin`。这样可以确保与未来的 ONTAP 版本向前兼容，这些版本可能会使功能 API 公开供未来的 Astra Trident 版本使用。可以创建自定义安全登录角色并将其用于 Astra Trident ，但不建议使用。

后端定义示例如下所示：

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret"
}
```

请注意，后端定义是凭据以纯文本格式存储的唯一位置。创建后端后，用户名 / 密码将使用 Base64 进行编码并存储为 Kubernetes 密钥。创建 / 更新后端是唯一需要了解凭据的步骤。因此，这是一项仅由管理员执行的操作，由 Kubernetes 或存储管理员执行。

启用基于证书的身份验证

新的和现有的后端可以使用证书并与 ONTAP 后端进行通信。后端定义需要三个参数。

- clientCertificate ： 客户端证书的 Base64 编码值。
- clientPrivateKey ： 关联私钥的 Base64 编码值。
- trustedCACertifate ： 受信任 CA 证书的 Base64 编码值。如果使用可信 CA ，则必须提供此参数。如果不使用可信 CA ，则可以忽略此设置。

典型的工作流包括以下步骤。

步骤

1. 生成客户端证书和密钥。生成时，将公用名（ Common Name ， CN ）设置为要作为身份验证的 ONTAP 用户。

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

2. 将可信 CA 证书添加到 ONTAP 集群。此问题可能已由存储管理员处理。如果未使用可信 CA ，则忽略。

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

3. 在 ONTAP 集群上安装客户端证书和密钥（从步骤 1 开始）。

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

4. 确认ONTAP 安全登录角色支持 `cert` 身份验证方法。

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

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

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

6. 使用 Base64 对证书，密钥和可信 CA 证书进行编码。

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

7. 使用从上一步获得的值创建后端。

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

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

更新身份验证方法或轮换凭据

您可以更新现有后端以使用其他身份验证方法或轮换其凭据。这两种方式都适用：使用用户名 / 密码的后端可以更新为使用证书；使用证书的后端可以更新为基于用户名 / 密码的后端。为此、您必须删除现有身份验证方法并添加新的身份验证方法。然后、使用更新后的backend.json文件、该文件包含要执行的所需参数 `tridentctl backend update`。

```
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": "secret",
"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。成功的后端更新表明， Astra Trident 可以与 ONTAP 后端进行通信并处理未来的卷操作。

管理 NFS 导出策略

Astra Trident 使用 NFS 导出策略来控制对其配置的卷的访问。

使用导出策略时， Astra Trident 提供了两个选项：

- Astra Trident 可以动态管理导出策略本身；在此操作模式下，存储管理员会指定一个表示可接受 IP 地址的 CIDR 块列表。Astra Trident 会自动将属于这些范围的节点 IP 添加到导出策略中。或者，如果未指定任何 CIDR ，则在节点上找到的任何全局范围的单播 IP 都将添加到导出策略中。
- 存储管理员可以手动创建导出策略和添加规则。除非在配置中指定了不同的导出策略名称，否则 Astra Trident 将使用默认导出策略。

动态管理导出策略

CSI Trident 20.04 版可以动态管理 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，这有助于将对存储集群的访问限制为仅允许 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。

只有 CSI Trident 才支持动态管理导出策略。请务必确保工作节点未被 NAT 处理。

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",
    "backendName": "ontap_nas_auto_export,
    "managementLIF": "192.168.0.135",
    "svm": "svm1",
    "username": "vsadmin",
    "password": "FaKePaSsWoRd",
    "autoExportCIDRs": ["192.168.0.0/24"],
    "autoExportPolicy": true
}
```

使用此功能时，您必须确保 SVM 中的根接合具有预先创建的导出策略，并具有允许节点 CIDR 块的导出规则（例如默认导出策略）。请始终遵循 NetApp 建议的最佳实践，为 Astra Trident 专用 SVM 。

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Astra Trident将为创建导出策略 `svm1` SVM并使用处理规则的添加和删除 `autoExportCIDRs` 地址块。例如、UUID为403b5326-8482-40db-96d0-d83fb3f4daec和的后端 `autoExportPolicy` 设置为 `true` 创建名为的导出策略 `trident-403b5326-8482-40db-96d0-d83fb3f4daec` 在 SVM 上。
- `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 Astra Trident 会添加在工作节点上找到的所有全局范围的单播地址。

在此示例中、将显示 `192.168.0.0/24` 提供了地址空间。这表示此地址范围内的 Kubernetes 节点 IP 将添加到 Astra Trident 创建的导出策略中。当Astra Trident注册其运行所在的节点时、它会检索该节点的IP地址并根据中提供的地址块对其进行检查 `autoExportCIDRs`。筛选 IP 后， Astra Trident 会为其发现的客户端 IP 创建导出策略规则，并为其标识的每个节点创建一个规则。

您可以更新 `autoExportPolicy` 和 `autoExportCIDRs` 用于后端。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。您也可以选择禁用 `autoExportPolicy` 用于后端、并回退到手动创建的导出策略。这需要设置 `exportPolicy` 参数。

在Astra Trident创建或更新后端之后、您可以使用检查后端 `tridentctl` 或相应的 `tridentbackend` CRD：

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

当节点添加到Kubernetes集群并注册到Astra Trident控制器时、现有后端的导出策略会进行更新(前提是它们位于中指定的地址范围内 `autoExportCIDRs` 后端)。

删除节点后， Astra Trident 会检查所有联机后端，以删除该节点的访问规则。通过从受管后端的导出策略中删除此节点 IP ， Astra Trident 可防止恶意挂载，除非此 IP 可由集群中的新节点重复使用。

对于以前存在的后端、请使用更新后端 `tridentctl update backend` 将确保Astra Trident自动管理导出策略。这将创建一个以后端 UUID 命名的新导出策略，后端上存在的卷将在重新挂载时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 Astra Trident Pod 。然后， Astra Trident 将更新其管理的后端的导出策略，以反映此 IP 更改。

ONTAP NAS配置选项和示例

了解如何在您的 Astra Trident 安装中创建和使用 ONTAP NAS 驱动程序。本节提供了后端配置示例以及有关如何将后端映射到 StorageClasses 的详细信息。

后端配置选项

有关后端配置选项，请参见下表：

参数	Description	Default
`version`		始终为 1

参数	Description	Default
`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地址要进行无缝MetroCluster 切换、必须指定SVM管理LIF。	"10.0.0.1 "， "2001： 1234：abcd：：： fefe] "
`dataLIF`	协议 LIF 的 IP 地址。对于 IPv6，请使用方括号。设置后无法更新	由 SVM 派生，除非另有说明
`autoExportPolicy`	启用自动创建和更新导出策略 [布尔值]	false
`autoExportCIDRs`	用于筛选Kubernetes节点IP的CIDR列表 `autoExportPolicy` 已启用	["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	
`svm`	要使用的 Storage Virtual Machine	如果是SVM、则派生 `managementLIF` 已指定
`igroupName`	要使用的 SAN 卷的 igroup 的名称	"trident — < 后端 UUID >"
`storagePrefix`	在 SVM 中配置新卷时使用的前缀。设置后无法更新	Trident
`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）
`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）
`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。示例 ｛"api"： false， "method "：true｝	空

参数	Description	Default
nfsMountOptions	NFS 挂载选项的逗号分隔列表	"
qtreesPerFlexvol	每个 FlexVol 的最大 qtree 数，必须在 50 ， 300 范围内	200
useREST	用于使用 ONTAP REST API 的布尔参数。MetroCluster 不支持*技术预览*。	false

<code>useREST</code>注意事项

- useREST 作为一个*技术预览版提供、建议用于测试环境、而不是生产工作负载。设置为时 true、Astra Trident将使用ONTAP REST API与后端进行通信。此功能需要使用ONTAP 9.10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。这一点可通过预定义来满足 vsadmin 和 cluster-admin 角色。
- useREST MetroCluster 不支持。

要与 ONTAP 集群通信，您应提供身份验证参数。这可以是安全登录的用户名 / 密码，也可以是已安装的证书。

如果您使用适用于NetApp ONTAP 后端的Amazon FSX、请勿指定 limitAggregateUsage 参数。。 fsxadmin 和 vsadmin Amazon FSX for NetApp ONTAP 提供的角色不包含检索聚合使用情况并通过Astra Trident限制聚合使用情况所需的访问权限。

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

创建后端时、请记住 dataLIF 和 storagePrefix 创建后无法修改。要更新这些参数，您需要创建一个新的后端。

可以为指定完全限定域名(FQDN) managementLIF 选项也可以为指定FQDN dataLIF 选项、在这种情况下、FQDN将用于NFS挂载操作。这样，您就可以创建循环 DNS ，以便在多个数据 LIF 之间实现负载平衡。

`managementLIF` 对于所有ONTAP 驱动程序、也可以设置为IPv6地址。确保将Astra Trident安装到 `--use-ipv6` 标志。必须小心定义 `managementLIF` 方括号内的IPv6地址。

使用IPv6地址时、请确保 managementLIF 和 dataLIF (如果包含在后端定义中)在方括号内进行定义、例如、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]。条件 dataLIF 如果未提供、则Astra Trident将从SVM提取IPv6数据LIF。

使用 autoExportPolicy 和 autoExportCIDRs 选项、CSI Trident可以自动管理导出策略。所有 ontap-nas-* 驱动程序均支持此功能。

。 ontap-nas-economy 驱动程序、 limitVolumeSize 选项还会限制它为qtree和LUN以及管理的卷的最大大小 qtreesPerFlexvol 选项用于自定义每个FlexVol 的最大qtree数。

。 nfsMountOptions 参数可用于指定挂载选项。Kubernetes 永久性卷的挂载选项通常在存储类中指定，但如果在存储类中未指定挂载选项，则 Astra Trident 将回退为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 Astra Trident 不会在关联的永久性卷上设置任何挂载选项。

> Astra Trident会在使用创建的所有卷的"Comments"字段中设置配置标签(`ontap-nas` 和(`ontap-nas-flexgroup`。根据所使用的驱动程序、注释将在FlexVol 上进行设置 (`ontap-nas`) 或FlexGroup (`ontap-nas-flexgroup`）。Astra Trident 会在配置存储池时将存储池上的所有标签复制到该存储卷。存储管理员可以为每个存储池定义标签，并对存储池中创建的所有卷进行分组。这样，您就可以根据后端配置中提供的一组可自定义标签来方便地区分卷了。

用于配置卷的后端配置选项

您可以在配置的特殊部分中使用这些选项来控制默认配置每个卷的方式。有关示例，请参见以下配置示例。

参数	Description	Default
`spaceAllocation`	LUN 的空间分配	true
`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为 "0"	条件 `snapshotPolicy` 为"无"、否则为""
`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Astra Trident中配置的任何卷都将启用NAE。有关详细信息、请参见： "Astra Trident如何与NVE和NAE配合使用"。	false
`securityStyle`	新卷的安全模式	"unix"
`tieringPolicy`	使用 " 无 " 的分层策略	适用于 ONTAP 9.5 SVM-DR 之前的配置的 " 仅快照 "
unixPermissions	新卷的模式	777.
snapshotDir	控制的可见性 `.snapshot` 目录	false
导出策略	要使用的导出策略	default
securityStyle	新卷的安全模式	"unix"

在 Astra Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。建议使用非共享 QoS 策略组，并确保策略组分别应用于每个成分卷。共享 QoS 策略组将对所有工作负载的总吞吐量实施上限。

下面是定义了默认值的示例：

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

适用于 ontap-nas 和 ontap-nas-flexgroups`现在、Astra Trident会使用新的计算方法来确保FlexVol 的大小与snapshotReserve百分比和PVC相同。当用户请求 PVC 时， Astra Trident 会使用新计算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和 `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、Astra Trident定义了 snapshotReserve 数字表示整个卷的百分比。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。。 volume show 命令应显示与以下示例类似的结果：

在升级 Astra Trident 时，先前安装的现有后端将按照上述说明配置卷。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、具有的2 GiB PVC `snapshotReserve=50` 之前的结果是、卷可提供1 GiB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` 具有基于证书的身份验证的驱动程序

这是一个最低后端配置示例。 `clientCertificate`， `clientPrivateKey`，和 `trustedCACertificate` (如果使用可信CA、则可选)将填充 `backend.json` 和分别采用客户端证书、专用密钥和可信CA证书的base64编码值。

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

`ontap-nas` 具有自动导出策略的驱动程序

此示例显示了如何指示 Astra Trident 使用动态导出策略自动创建和管理导出策略。此操作对于也是如此 `ontap-nas-economy` 和 `ontap-nas-flexgroup` 驱动程序。

```
{
    "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": "secret",
    "nfsMountOptions": "nfsvers=4",
}
```

`ontap-nas-flexgroup` 驱动程序

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "labels": {"k8scluster": "test-cluster-east-1b", "backend": "test1-
ontap-cluster"},
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
}
```

`ontap-nas` 支持**IPv6**的驱动程序

```
{
 "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": "netapp123"
}
```

ontap-nas-economy 驱动程序

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret"
}
```

虚拟存储池后端示例

在下面所示的示例后端定义文件中、会为所有存储池设置特定的默认值、例如 spaceReserve 无、spaceAllocation 为false、和 encryption 为false。虚拟存储池在存储部分中进行定义。

在此示例中、某些存储池会设置自己的存储池 spaceReserve， spaceAllocation，和 encryption 值、而某些池会覆盖上述设置的默认值。

ontap-nas 驱动程序

```
{
    {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "admin",
    "password": "secret",
    "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":{"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` 驱动程序

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

```
    "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-economy 驱动程序

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

    "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":{"department":"legal", "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 定义引用了上述虚拟存储池。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。卷将在选定虚拟池中定义各个方面。

- 第一个StorageClass (`protection-gold`)将映射到中的第一个、第二个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第一个虚拟存储池 `ontap-san` 后端。这是唯一一个提供黄金级保护的池。
- 第二个StorageClass (`protection-not-gold`)将映射到中的第三个、第四个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第二个、第三个虚拟存储池 `ontap-san` 后端。这些池是唯一提供黄金级以外保护级别的池。
- 第三个StorageClass (`app-mysqldb`)将映射到中的第四个虚拟存储池 `ontap-nas` 中的后端和第三个虚拟存储池 `ontap-san-economy` 后端。这些池是唯一为 mysqldb 类型的应用程序提供存储池配置的池。
- 第四个StorageClass (`protection-silver-creditpoints-20k`)将映射到中的第三个虚拟存储池 `ontap-nas-flexgroup` 中的后端和第二个虚拟存储池 `ontap-san` 后端。这些池是唯一以 20000 个信用点提供黄金级保护的池。
- 第五个StorageClass (`creditpoints-5k`)将映射到中的第二个虚拟存储池 `ontap-nas-economy` 中的后端和第三个虚拟存储池 `ontap-san` 后端。这些是唯一一款具有 5000 个信用点的池产品。

Astra Trident 将决定选择哪个虚拟存储池，并确保满足存储要求。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

将 **Astra Trident** 与适用于 **NetApp ONTAP** 的 **Amazon FSX** 结合使用

"适用于 NetApp ONTAP 的 Amazon FSX"是一种完全受管的 AWS 服务，支持客户启动和运行由 NetApp 的 ONTAP 存储操作系统提供支持的文件系统。Amazon FSX for NetApp ONTAP 支持您利用您熟悉的 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 支持 ONTAP 的许多文件系统功能和管理 API 。

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 * 。

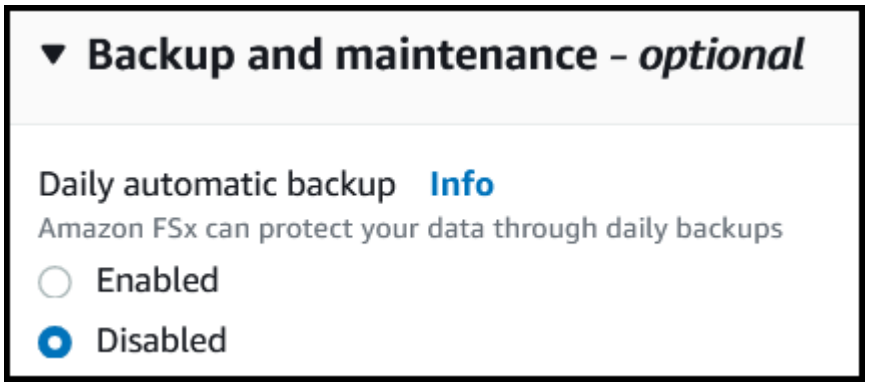
通过将 Astra Trident 与适用于 NetApp ONTAP 的 Amazon FSx 结合使用，您可以确保在 Amazon Elastic Kubernetes Service （ EKS ）中运行的 Kubernetes 集群可以配置由 ONTAP 备份的块和文件永久性卷。

创建适用于 **ONTAP** 的 **Amazon FSX** 文件系统

Trident 无法删除在启用了自动备份的 Amazon FSx 文件系统上创建的卷。要删除 PVC ，您需要手动删除 PV 和 ONTAP 的 FSX 卷。

要防止此问题描述 ，请执行以下操作：

- 请勿使用 "* 快速创建 "* 来创建适用于 ONTAP 的 FSX 文件系统。快速创建工作流可启用自动备份，但不提供选择退出选项。
- 使用 "** 标准创建 " 时，禁用自动备份。禁用自动备份可以使 Trident 成功删除卷，而无需进一步手动干预。



了解 **Astra Trident**

如果您是 Astra Trident 的新用户，请使用下面提供的链接进行熟悉：

- "常见问题解答"
- "使用 Astra Trident 的要求"
- "部署 Astra Trident"
- "配置适用于 NetApp ONTAP 的 ONTAP ， Cloud Volumes ONTAP 和 Amazon FSX 的最佳实践"
- "集成 Astra Trident"
- "ONTAP SAN 后端配置"
- "ONTAP NAS 后端配置"

详细了解驱动程序功能 "此处"。

适用于 NetApp ONTAP 的 Amazon FSX 使用 "FabricPool" 以管理存储层。通过它，您可以根据数据是否经常访问来将数据存储在层中。

Astra Trident应作为运行 `vsadmin` SVM用户或具有相同角色的其他名称的用户。适用于NetApp ONTAP的Amazon FSX具有 `fsxadmin` 有限更换ONTAP 的用户 `admin` 集群用户。不建议使用 `fsxadmin` 用户、使用Trident、作为 `vsadmin` SVM用户可以访问更多Astra Trident功能。

驱动程序

您可以使用以下驱动程序将 Astra Trident 与适用于 NetApp ONTAP 的 Amazon FSx 集成：

- `ontap-san`：配置的每个PV都是其自己的Amazon FSX for NetApp ONTAP 卷中的一个LUN。
- `ontap-san-economy`：配置的每个PV都是一个LUN、对于NetApp ONTAP 卷、每个Amazon FSX的LUN数量是可配置的。
- `ontap-nas`：配置的每个PV都是一个适用于NetApp ONTAP 的完整Amazon FSX卷。
- `ontap-nas-economy`：配置的每个PV都是一个qtree、对于NetApp ONTAP 卷、每个Amazon FSX的qtree数量是可配置的。
- `ontap-nas-flexgroup`：配置的每个PV都是一个适用于NetApp ONTAP FlexGroup 的完整Amazon FSX卷。

身份验证

Astra Trident 提供两种身份验证模式：

- 基于证书： Astra Trident 将使用 SVM 上安装的证书与 FSX 文件系统上的 SVM 进行通信。
- 基于凭据：您可以使用 `fsxadmin` 文件系统或的用户 `vsadmin` 为SVM配置的用户。

> 我们强烈建议使用 `vsadmin` 用户、而不是 `fsxadmin` 配置后端。Astra Trident 将使用此用户名和密码与 FSX 文件系统进行通信。

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。

要了解有关身份验证的详细信息，请参见以下链接：

- "ONTAP NAS"
- "ONTAP SAN"

使用适用于 **NetApp ONTAP** 的 **Amazon FSX** 在 **EKS** 上部署和配置 **Astra Trident**

您需要的内容

- 具有的现有Amazon EKS集群或自管理Kubernetes集群 `kubectl` 已安装。

- 可从集群的工作节点访问的适用于 NetApp ONTAP 文件系统和 Storage Virtual Machine （ SVM ）的现有 Amazon FSX 。
- 为准备工作的工作节点 "NFS 和 / 或 iSCSI"。

> 确保按照 Amazon Linux 和 Ubuntu 所需的节点准备步骤进行操作 "Amazon Machine 映像"（ AMIS ），具体取决于您的 EKS AMI 类型。

有关其他 Astra Trident 要求，请参见 "此处"。

步骤

1. 使用其中一种部署Astra Trident "部署方法"。
2. 按照以下步骤配置 Astra Trident ：
   a. 收集 SVM 的管理 LIF DNS 名称。例如、使用AWS命令行界面查找 `DNSName` 下的条目 `Endpoints → Management` 运行以下命令后：

```
aws fsx describe-storage-virtual-machines --region <file system region>
```

3. 创建并安装用于身份验证的证书。如果您使用的是 `ontap-san` 后端、请参见 "此处"。如果您使用的是 `ontap-nas` 后端、请参见 "此处"。

> 您可以从可以访问文件系统的任何位置使用 SSH 登录到文件系统（例如，安装证书）。使用 `fsxadmin` 用户、创建文件系统时配置的密码以及中的管理DNS名称 `aws fsx describe-file-systems`。

4. 使用您的证书和管理 LIF 的 DNS 名称创建后端文件，如以下示例所示：

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "customBackendName",
  "managementLIF": "svm-XXXXXXXXXXXXXXXXX.fs-XXXXXXXXXXXXXXXXX.fsx.us-east-2.aws.internal",
  "svm": "svm01",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz",
 }
```

有关创建后端的信息，请参见以下链接：

- "使用 ONTAP NAS 驱动程序配置后端"
- "使用 ONTAP SAN 驱动程序配置后端"

请勿指定 `dataLIF` 。 `ontap-san` 和 `ontap-san-economy` 支持Astra Trident使用多路径的驱动程序。

。 `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。

部署完成后，执行以下步骤以创建 "存储类，配置卷以及将卷挂载到 Pod 中"。

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"

使用 kubectl 创建后端

后端定义了 Astra Trident 与存储系统之间的关系。它告诉 Astra Trident 如何与该存储系统进行通信，以及 Astra Trident 如何从该存储系统配置卷。安装 Astra 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Kubernetes界面创建和管理Trident后端。您可以使用执行此操作 `kubectl` 或与Kubernetes分发版等效的CLI工具。

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`， `tbconfig`， `tbackendconfig`)是一个命名为节奏的前端CRD、可用于管理Astra Trident后端 `kubectl`。现在、Kubernetes和存储管理员可以直接通过Kubernetes命令行界面创建和管理后端、而无需专用命令行实用程序 (`tridentctl`）。

创建时 `TridentBackendConfig` 对象、将发生以下情况：

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

每个 `TridentBackendConfig` 使用维护一对一映射 `TridentBackend`。前者是为用户提供的用于设计和配置后端的接口；后者是 Trident 表示实际后端对象的方式。

`TridentBackend` CRS由Astra Trident自动创建。您 * 不应 * 修改它们。如果要更新后端、请通过修改来执行此操作 `TridentBackendConfig` 对象。

有关的格式、请参见以下示例 `TridentBackendConfig` CR：

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

您还可以查看中的示例 "Trident 安装程序" 所需存储平台 / 服务的示例配置目录。

。 `spec` 获取后端特定的配置参数。在此示例中、后端使用 `ontap-san` 存储驱动程序、并使用此处所示的配置参数。有关所需存储驱动程序的配置选项列表，请参见 "存储驱动程序的后端配置信息"。

。 `spec` 第节还包括 `credentials` 和 `deletionPolicy` 字段、这些字段是在中新增的 `TridentBackendConfig` CR：

- `credentials`：此参数是必填字段、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`：此字段定义了在使用时应执行的操作 `TridentBackendConfig` 已删除。它可以采用以下两种可能值之一：
  - `delete`：这将删除这两者 `TridentBackendConfig` CR以及关联的后端。这是默认值。
  - `retain`：当出现时 `TridentBackendConfig` CR已删除、后端定义仍存在、可使用进行管理 `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。此字段的值可以在之后更新 `TridentBackendConfig` 已创建。

> 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的名称 `TridentBackendConfig` 对象(metadata.name)。建议使用显式设置后端名称 `spec.backendName`。

> 使用创建的后端 `tridentctl` 没有关联的 `TridentBackendConfig` 对象。您可以选择使用管理此类后端 `kubectl` 通过创建 `TridentBackendConfig` CR.必须小心指定相同的配置参数(例如 `spec.backendName`， `spec.storagePrefix`， `spec.storageDriverName`等）。Astra Trident将自动绑定新创建的 `TridentBackendConfig` 使用预先存在的后端。

步骤概述

以使用创建新后端 `kubectl`、您应执行以下操作：

1. 创建 "Kubernetes 机密"。此密钥包含 Astra 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
适用于 GCP 的 Cloud Volumes Service	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
适用于 GCP 的 Cloud Volumes Service	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	username	用于连接到集群 /SVM 的用户名。用于基于凭据的身份验证
ONTAP	password	连接到集群 /SVM 的密码。用于基于凭据的身份验证
ONTAP	客户端权限密钥	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证

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。适用于 `ontap-san` 和 `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP 启动程序密钥。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy`
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP 目标启动程序密钥。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy`

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

第2步：创建 `TridentBackendConfig` CR

现在、您可以创建了 `TridentBackendConfig` CR.在此示例中、是使用的后端 `ontap-san` 驱动程序是使用创建的 `TridentBackendConfig` 对象如下所示：

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

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

第3步：验证的状态 `TridentBackendConfig` CR

现在、您创建了 `TridentBackendConfig` cr、您可以验证状态。请参见以下示例：

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

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

阶段可以采用以下值之一：

- `Bound`： `TridentBackendConfig` CR与后端关联、后端包含 `configRef` 设置为 `TridentBackendConfig` CR的UID。
- `Unbound`：表示使用 ""。。 `TridentBackendConfig` 对象未绑定到后端。所有新创建的 `TridentBackendConfig` 默认情况下、CRS处于此阶段。此阶段发生更改后，它将无法再次还原为 "Unbound （已取消绑定） "。
- `Deleting`： `TridentBackendConfig` CR `deletionPolicy` 已设置为delete。当 `TridentBackendConfig` CR将被删除、它将过渡到Deleting状态。
  - 如果后端不存在永久性卷请求(PVC)、请删除 `TridentBackendConfig` 将导致Astra Trident删除后端以及 `TridentBackendConfig` CR.
  - 如果后端存在一个或多个 PVC ，则会进入删除状态。。 `TridentBackendConfig` CR随后也进入删除阶段。后端和 `TridentBackendConfig` 只有在删除所有PVC后才会删除。
- `Lost`：与关联的后端 `TridentBackendConfig` 意外或故意删除了CR和 `TridentBackendConfig` CR仍引用已删除的后端。。 `TridentBackendConfig` 无论使用什么、仍可删除CR `deletionPolicy` 价值。
- `Unknown`：Astra Trident无法确定与关联的后端的状态或是否存在 `TridentBackendConfig` CR.例如、如果API服务器未响应或 `tridentbackends.trident.netapp.io` 缺少CRD。这可能需要用户干预。

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` 和 `backendUUID` 为响应创建的后端的 `TridentBackendConfig` CR.。`lastOperationStatus` 字段表示上次操作的状态 `TridentBackendConfig` cr、可以由用户触发(例如、用户在中更改了某些内容 `spec`)或由Astra Trident触发(例如、在Astra Trident重新启动期间)。可以是成功，也可以是失败。 `phase` 表示之间关系的状态 `TridentBackendConfig` CR和后端。在上面的示例中、 `phase` 已绑定值、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。

> 您不能更新或删除包含关联的后端 `TridentBackendConfig` 对象使用 `tridentctl`。了解切换所涉及的步骤 `tridentctl` 和 `TridentBackendConfig`， "请参见此处"。

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

删除 `TridentBackendConfig`、您可以指示Astra Trident删除/保留后端(基于 `deletionPolicy`）。要删除后端、请确保 `deletionPolicy` 设置为delete。仅删除 `TridentBackendConfig`、请确保 `deletionPolicy` 设置为保留。这样可以确保后端仍然存在、并可使用进行管理 `tridentctl`。

运行以下命令：

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

Astra Trident不会删除正在使用的Kubernetes机密 `TridentBackendConfig`。Kubernetes 用户负责清理密钥。删除机密时必须小心。只有在后端未使用机密时，才应将其删除。

查看现有后端

运行以下命令：

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

您也可以运行 `tridentctl get backend -n trident` 或 `tridentctl get backend -o yaml -n trident` 获取所有后端的列表。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、请使用中使用的Kubernetes Secret `TridentBackendConfig` 必须更新对象。Astra 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）。在这种情况下、 `TridentBackendConfig` 可以直接通过Kubernetes更新对象。

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

或者、也可以对现有进行更改 `TridentBackendConfig` cr运行以下命令：

```
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。您可以通过运行来查看日志以确定发生原因 `kubectl get tbc <tbc-name> -o yaml -n trident` 或 `kubectl describe tbc <tbc-name> -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"下，运行以下命令：

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

如果后端创建失败，则后端配置出现问题。您可以运行以下命令来查看日志以确定发生原因：

```
tridentctl logs -n trident
```

确定并更正配置文件中的问题后、您只需运行即可 `create` 命令。

删除后端

要从 Astra 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 Astra 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中的问题后、您只需运行即可 `update` 命令。

确定使用后端的存储类

这是一个示例、说明您可以通过问题解答 与JSON一起提出的问题 `tridentctl` 后端对象的输出。此操作将使用 `jq` 实用程序、您需要安装该实用程序。

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

这也适用于使用创建的后端 `TridentBackendConfig`。

在后端管理选项之间移动

了解如何在 Astra Trident 中管理后端。随附 `TridentBackendConfig`现在、管理员可以通过两种独特的方式管理

后端。这会提出以下问题：

- 可以使用创建后端 `tridentctl` 通过进行管理 `TridentBackendConfig`?
- 可以使用创建后端 `TridentBackendConfig` 可使用进行管理 `tridentctl`?

管理 `tridentctl` 后端使用 `TridentBackendConfig`

本节介绍管理使用创建的后端所需的步骤 `tridentctl` 通过创建直接通过Kubernetes界面 `TridentBackendConfig` 对象。

这适用于以下情形：

- 预先存在的后端、没有 `TridentBackendConfig` 因为它们是使用创建的 `tridentctl`。
- 使用创建的新后端 `tridentctl`而其他 `TridentBackendConfig` 对象存在。

在这两种情况下，后端仍会存在，其中 Astra Trident 会计划卷并对其进行操作。管理员可以选择以下两种方式之一：

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

使用管理已有后端 `kubectl`、您需要创建 `TridentBackendConfig` 绑定到现有后端。下面简要介绍了它的工作原理：

1. 创建 Kubernetes 机密。此密钥包含 Astra 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。必须小心指定相同的配置参数(例如 `spec.backendName`， `spec.storagePrefix`，`spec.storageDriverName`等)。 `spec.backendName` 必须设置为现有后端的名称。

第 0 步：确定后端

以创建 `TridentBackendConfig` 如果绑定到现有后端、则需要获取后端的配置。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。
- 虚拟存储池（如果存在）必须与原始后端保持相同的顺序。
- 凭据通过 Kubernetes Secret 提供，而不是以纯文本形式提供。

在这种情况下、将显示 `TridentBackendConfig` 将如下所示：

```
cat backend-tbc-ontap-nas.yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'

kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

第3步：验证的状态 `TridentBackendConfig` CR

在之后 `TridentBackendConfig` 已创建、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 -n trident 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、后端将使用进行完全管理 `tbc-ontap-nas-backend` `TridentBackendConfig` 对象。

管理 `TridentBackendConfig` 后端使用 `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完全管理此类后端 `tridentctl` 删除 `TridentBackendConfig` 并确保 `spec.deletionPolicy` 设置为 `retain`。

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

从输出中可以看出这一点 `TridentBackendConfig` 已成功创建并绑定到后端[观察后端的UUUUID]。

第1步：确认 `deletionPolicy` 设置为 `retain`

让我们来了解一下的价值 `deletionPolicy`。此值需要设置为 `retain`。这样可以确保在出现时 `TridentBackendConfig` CR已删除、后端定义仍存在、可使用进行管理 `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
```

请勿继续执行下一步、除非 `deletionPolicy` 设置为 `retain`。

第2步：删除 `TridentBackendConfig` CR

最后一步是删除 `TridentBackendConfig` CR.确认后 `deletionPolicy` 设置为 `retain`、您可以继续执行删除操作：

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

删除时 `TridentBackendConfig` 对象、Astra Trident只需将其删除、而不实际删除后端本身。

管理存储类

查找有关创建存储类，删除存储类以及查看现有存储类的信息。

设计存储类

请参见 "此处" 有关什么是存储类以及如何配置这些类的详细信息，请参见。

创建存储类。

创建存储类文件后，运行以下命令：

```
kubectl create -f <storage-class-file>
```

`<storage-class-file>` 应替换为存储类文件名。

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， Astra Trident 将继续对其进行管理。

> Astra Trident强制使用空 `fsType` 创建的卷。对于iSCSI后端、建议强制实施 `parameters.fsType` 在StorageClass中。您应删除使用创建的StorageClasses并重新创建它们 `parameters.fsType` 已指定。

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 Astra Trident 的同步存储类，请运行以下命令：

```
tridentctl get storageclass
```

- 要查看 Astra Trident 的同步存储类详细信息，请运行以下命令：

```
tridentctl get storageclass <storage-class> -o json
```

设置默认存储类

Kubernetes 1.6 增加了设置默认存储类的功能。如果用户未在永久性卷声明（ PVC ）中指定永久性卷，则此存储类将用于配置永久性卷。

- 通过设置标注来定义默认存储类 `storageclass.kubernetes.io/is-default-class` 在存储类定义中为true。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。

> 在任何给定时间，集群中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

这是一个示例、说明您可以通过问题解答 与JSON一起提出的问题 `tridentctl` Astra Trident后端对象的输出。此操作将使用 `jq` 实用程序、您可能需要先安装此实用程序。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

执行卷操作

了解 Astra Trident 为管理卷提供的功能。

- "使用 CSI 拓扑"
- "使用快照"
- "展开卷"
- "导入卷"

使用 **CSI** 拓扑

Astra Trident 可以通过使用有选择地创建卷并将其附加到 Kubernetes 集群中的节点 "CSI 拓扑功能"。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， Astra Trident 使用了 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：

- 使用 `VolumeBindingMode` 设置为 `Immediate`、Astra Trident将创建卷、而不会感知任何拓扑。创建 PVC 时会处理卷绑定和动态配置。这是默认值 `VolumeBindingMode` 和适用于不强制实施拓扑限制的集群。创建永久性卷时，不会依赖于请求的 Pod 的计划要求。
- 使用 `VolumeBindingMode` 设置为 `WaitForFirstConsumer`、在计划和创建使用PVC的Pod之前、将延迟为PVC创建和绑定永久性卷。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Kubernetes集群 "支持的Kubernetes版本"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 集群中的节点应具有可引入拓扑感知的标签 (`topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone`）。在安装 Astra Trident 之前，集群中的节点上应存在这些标签 * ，以使 Astra 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

可以设计 Astra Trident 存储后端，以便根据可用性区域有选择地配置卷。每个后端都可以具有一个可选的 `supportedTopologies` 表示必须支持的分区和区域列表的块。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。

下面是后端定义示例：

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "xxxxxxxxxxxx",
 "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 ，Astra Trident 将在后端创建卷。

您可以定义 `supportedTopologies` 也是每个存储池的一个。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": "Netapp123",
"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"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
         "region": "Iowa-DC",
         "zone": "Iowa-DC-B",
         "supportedTopologies": [
             {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
         ]
     }
]
}
```

在此示例中、将显示 `region` 和 `zone` 标签表示存储池的位置。 `topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone` 指定存储池的使用位置。

第 **2** 步：定义可识别拓扑的 **StorageClasses**

根据为集群中的节点提供的拓扑标签，可以将 StorageClasses 定义为包含拓扑信息。这将确定用作 PVC 请求候选对象的存储池，以及可使用 Trident 配置的卷的节点子集。

请参见以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
- matchLabelExpressions:
- key: topology.kubernetes.io/zone
  values:
  - us-east1-a
  - us-east1-b
- key: topology.kubernetes.io/region
  values:
  - us-east1
parameters:
  fsType: "ext4"
```

在上述StorageClass定义中、 `volumeBindingMode` 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` 提供要使用的分区和区域。。 `netapp-san-us-east1` StorageClass将在上创建PVC `san-backend-us-east1` 上述定义的后端。

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 。

请参见示例 `spec` 以下：

```
---
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Kubernetes在中的节点上计划Pod `us-east1` 区域、然后从中的任何节点中进行选择 `us-east1-a` 或 `us-east1-b` 分区。

请参见以下输出：

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

更新后端以包括 `supportedTopologies`

可以更新已有后端以包括列表 `supportedTopologies` 使用 `tridentctl backend update`。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

您可以创建永久性卷(PV)的Kubernetes VolumeSnapshot (卷快照)、以维护Astra Trident卷的时间点副本。此外、您还可以从现有卷快照创建一个新卷、也称为_clone_。支持卷快照 `ontap-nas`， `ontap-san`， `ontap-san-economy`， `solidfire-san`， `gcp-cvs`，和 `azure-netapp-files` 驱动程序。

开始之前

您必须具有外部快照控制器和自定义资源定义(CRD)。这是Kubernetes流程编排程序(例如：Kubeadm、GKE、OpenShift)的职责。

如果您的Kubernetes分发版不包含快照控制器和CRD、请参见 [部署卷快照控制器]。

> 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。

第**1**步：创建 `VolumeSnapshotClass`

此示例将创建一个卷快照类。

```
cat snap-sc.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 `driver` 指向Astra Trident的CSI驱动程序。 `deletionPolicy` 可以是 `Delete` 或 `Retain`。设置为时 `Retain`、存储集群上的底层物理快照会保留、即使在使用时也是如此 `VolumeSnapshot` 对象已删除。

有关详细信息、请参见链接：./trident引用/objects.html#Kubernetes -volumesnapshotclass-objects[`VolumeSnapshotClass`]。

第 **2** 步：创建现有 **PVC** 的快照

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
```

在此示例中、为名为的PVC创建快照 `pvc1` 快照的名称设置为 `pvc1-snap`。

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

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

这就创建了 `VolumeSnapshot` 对象。VolumeSnapshot类似于PVC、并与关联 `VolumeSnapshotContent` 表示实际快照的对象。

可以标识 `VolumeSnapshotContent` 的对象 `pvc1-snap` VolumeSnapshot的说明。

```
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
.
.
```

。 `Snapshot Content Name` 标识提供此快照的VolumeSnapshotContent对象。。 `Ready To Use` 参数表示可使用Snapshot创建新的PVC。

第 3 步：从 VolumeSnapshots 创建 PVC

以下示例将使用快照创建PVC：

```
cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

`dataSource` 显示必须使用名为的VolumeSnapshot创建PVC `pvc1-snap` 作为数据源。此操作将指示 Astra Trident 从快照创建 PVC 。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。

> 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。要删除 Astra Trident 卷，应删除该卷的快照。

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. 在所需命名空间中创建Snapshot控制器。编辑以下 YAML 清单以修改命名空间。

```
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
```

相关链接

- "卷快照"
- "VolumeSnapshotClass"

展开卷

通过 Astra Trident ， Kubernetes 用户可以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 ）。

> 支持iSCSI卷扩展 `ontap-san`， `ontap-san-economy`， `solidfire-san` 驱动程序并需要Kubernetes 1.16及更高版本。

概述

扩展 iSCSI PV 包括以下步骤：

- 编辑StorageClass定义以设置 `allowVolumeExpansion` 字段设置为 `true`。
- 编辑PVC定义并更新 `spec.resources.requests.storage` 以反映新需要的大小、该大小必须大于原始大小。
- 要调整 PV 大小，必须将 PV 连接到 Pod 。调整 iSCSI PV 大小时，有两种情况：
  - 如果 PV 连接到 Pod ，则 Astra Trident 会扩展存储后端的卷，重新扫描设备并调整文件系统大小。
  - 尝试调整未连接 PV 的大小时， Astra 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。

以下示例显示了扩展 iSCSI PV 的工作原理。

第 **1** 步：配置 **StorageClass** 以支持卷扩展

```
cat storageclass-ontapsan.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的StorageClass、请对其进行编辑以包括 `allowVolumeExpansion` 参数。

第 **2** 步：使用您创建的 **StorageClass** 创建 **PVC**

```
cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Astra Trident 会创建一个永久性卷（ PV ）并将其与此永久性卷声明（ PVC ）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

在此示例中、创建了一个使用的POD `san-pvc`。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-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:    centos-pod
```

第 4 步：展开 PV

要将已创建的PV从1Gi调整为2Gi、请编辑PVC定义并更新 `spec.resources.requests.storage` 至2Gi。

```
kubectl edit pvc san-pvc
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 ...
```

第 **5** 步：验证扩展

您可以通过检查 PVC ， PV 和 Astra Trident 卷的大小来验证扩展是否正常运行：

```
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

Astra Trident支持对上配置的NFS PV进行卷扩展 `ontap-nas`， `ontap-nas-economy`， `ontap-nas-flexgroup`， `gcp-cvs`，和 `azure-netapp-files` 后端。

第 1 步：配置 StorageClass 以支持卷扩展

要调整NFS PV的大小、管理员首先需要通过设置来配置存储类以允许卷扩展 `allowVolumeExpansion` 字段设置为 `true`：

```
cat storageclass-ontapnas.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

如果您已创建没有此选项的存储类、则只需使用编辑现有存储类即可 `kubectl edit storageclass` 以允许卷扩展。

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
```

Astra Trident 应为此 PVC 创建一个 20 MiB NFS PV：

```
kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

第 3 步：展开 PV

要将新创建的20MiB PV调整为1GiB、请编辑PVC并进行设置 `spec.resources.requests.storage` 到1 GB：

```
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 Astra Trident 卷的大小来验证调整大小是否正常工作：

```
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

您可以使用将现有存储卷作为Kubernetes PV导入 `tridentctl import`。

支持卷导入的驱动程序

下表介绍了支持导入卷的驱动程序及其引入的版本。

驱动程序	版本。
`ontap-nas`	19.04
`ontap-nas-flexgroup`	19.04
`solidfire-san`	19.04
`azure-netapp-files`	19.04
`gcp-cvs`	19.04

驱动程序	版本。
ontap-san	19.04

为什么应导入卷?

将卷导入到 Trident 的使用情形有多种：

- 对应用程序进行容器化并重复使用其现有数据集
- 为临时应用程序使用数据集的克隆
- 重建发生故障的 Kubernetes 集群
- 在灾难恢复期间迁移应用程序数据

导入的工作原理是什么?

卷导入过程使用永久性卷声明（ PVC ）文件创建 PVC 。PVC 文件应至少包含 name ， namespace ，accessModes 和 storageClassName 字段，如以下示例所示。

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

。 `tridentctl` 客户端用于导入现有存储卷。Trident 通过保留卷元数据并创建 PVC 和 PV 来导入卷。

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

要导入存储卷，请指定包含该卷的 Astra Trident 后端的名称以及用于唯一标识存储上的卷的名称（例如：ONTAP FlexVol ， Element Volume ， CVS 卷路径）。存储卷必须允许读 / 写访问，并可由指定的 Astra Trident 后端访问。。 `-f` 字符串参数为必填项、用于指定YAML或JSON PVC文件的路径。

当 Astra Trident 收到导入卷请求时，系统会在 PVC 中确定并设置现有卷大小。存储驱动程序导入卷后，系统将创建 PV ，并为其创建一个 Claims Ref 。回收策略最初设置为 `retain` 在PV中。Kubernetes 成功绑定 PVC 和 PV 后，将更新回收策略以匹配存储类的回收策略。存储类的回收策略为时 `delete`、删除PV时、存储卷将被删除。

使用导入卷时 `--no-manage` 参数中、Trident不会在对象的生命周期内对PVC或PV执行任何其他操作。因为Trident会忽略的PV和PVC事件 `--no-manage` 对象、删除PV时不会删除存储卷。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。

Trident 19.07 及更高版本可处理 PV 的连接，并在导入卷时挂载该卷。对于使用早期版本的 Astra Trident 进行的导入，数据路径中不会执行任何操作，卷导入将不会验证是否可以挂载卷。如果卷导入出错(例如、StorageClass不正确)、您可以通过将PV上的回收策略更改为来恢复 `retain`、删除PVC和PV、然后重试volume import命令。

`ontap-nas` 和 `ontap-nas-flexgroup` 导入

使用创建的每个卷 `ontap-nas` 驱动程序是ONTAP 集群上的FlexVol。使用导入FlexVol `ontap-nas` 驱动程序的工作原理相同。ONTAP 集群上已存在的FlexVol 可以作为导入 `ontap-nas` PVC。同样、FlexGroup vols也可以作为导入 `ontap-nas-flexgroup` PVC。

> 要由 Trident 导入 ONTAP 卷，必须为 rw 类型。如果卷的类型为 DP ，则为 SnapMirror 目标卷；应先中断镜像关系，然后再将卷导入到 Trident 中。

> 。 `ontap-nas` 驱动程序无法导入和管理qtree。。 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名称。

例如、导入名为的卷 `managed_volume` 位于名为的后端 `ontap_nas`、请使用以下命令：

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

导入名为的卷 `unmanaged_volume` (在上 `ontap_nas backend`)、而Trident不会管理此项、请使用以下命令：

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

使用时 `--no-manage` 参数、Trident不会重命名卷或验证卷是否已挂载。如果卷未手动挂载，则卷导入操作将失败。

> 先前存在的使用自定义 UnixPermissions 导入卷的错误已得到修复。您可以在 PVC 定义或后端配置中指定 unixPermissions ，并指示 Astra Trident 相应地导入卷。

`ontap-san` 导入

Astra Trident 还可以导入包含单个 LUN 的 ONTAP SAN FlexVol 。这与一致 `ontap-san` 驱动程序、用于为FlexVol 中的每个PVC和LUN创建FlexVol。您可以使用 `tridentctl import` 命令的方式与其他情况相同：

- 包括的名称 `ontap-san` 后端。
- 提供需要导入的 FlexVol 的名称。请记住，此 FlexVol 仅包含一个必须导入的 LUN 。
- 提供必须与结合使用的PVC定义路径 `-f` 标志。
- 可以选择对 PVC 进行管理，也可以选择不对其进行管理。默认情况下， Trident 将管理 PVC 并重命名后端的 FlexVol 和 LUN 。要作为非受管卷导入、请传递 `--no-manage` 标志。

> 导入非受管时 `ontap-san` 卷中的LUN、您应确保FlexVol 中的LUN名为 `lun0` 和映射到具有所需启动程序的igroup。Astra Trident 会自动为受管导入处理此问题。

然后， Astra Trident 将导入 FlexVol 并将其与 PVC 定义关联。Astra Trident还会将FlexVol 重命名为 `pvc-<uuid>` 将FlexVol 中的LUN格式化为 `lun0`。

> 建议导入没有活动连接的卷。如果要导入当前使用的卷，请先克隆该卷，然后再执行导入。

示例

以导入 `ontap-san-managed` 上存在的FlexVol `ontap_san_default` 后端、运行 `tridentctl import` 命令为：

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

ONTAP 卷的类型必须为 RW ，才能由 Astra Trident 导入。如果卷的类型为 DP ，则为 SnapMirror 目标卷；在将卷导入到 Astra Trident 之前，应中断镜像关系。

`element` 导入

您可以使用 Trident 将 NetApp Element 软件 /NetApp HCI 卷导入到 Kubernetes 集群中。您需要提供Astra Trident后端的名称、以及作为参数的卷和PVC文件的唯一名称 `tridentctl import` 命令：

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

Element 驱动程序支持重复的卷名称。如果卷名称重复，则 Trident 的卷导入过程将返回错误。作为临时解决策，克隆卷并提供唯一的卷名称。然后导入克隆的卷。

`gcp-cvs` 导入

要在 GCP 中导入由 NetApp Cloud Volumes Service 支持的卷，请按卷路径而非名称来标识该卷。

导入 gcp-cvs 后端上的卷称为 gcpcvs_YEppr 卷路径 adroit-jolly-swift、请使用以下命令：

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

卷路径是卷导出路径中： / 之后的部分。例如、如果导出路径为 10.0.0.1:/adroit-jolly-swift、卷路径为 adroit-jolly-swift。

azure-netapp-files 导入

导入 azure-netapp-files 后端上的卷称为 azurenetappfiles_40517 卷路径 `importvol1`下，运行以下命令：

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

ANF 卷的卷路径位于： / 之后的挂载路径中。例如、如果挂载路径为 10.0.0.2:/importvol1、卷路径为 importvol1。

在命名空间之间共享NFS卷

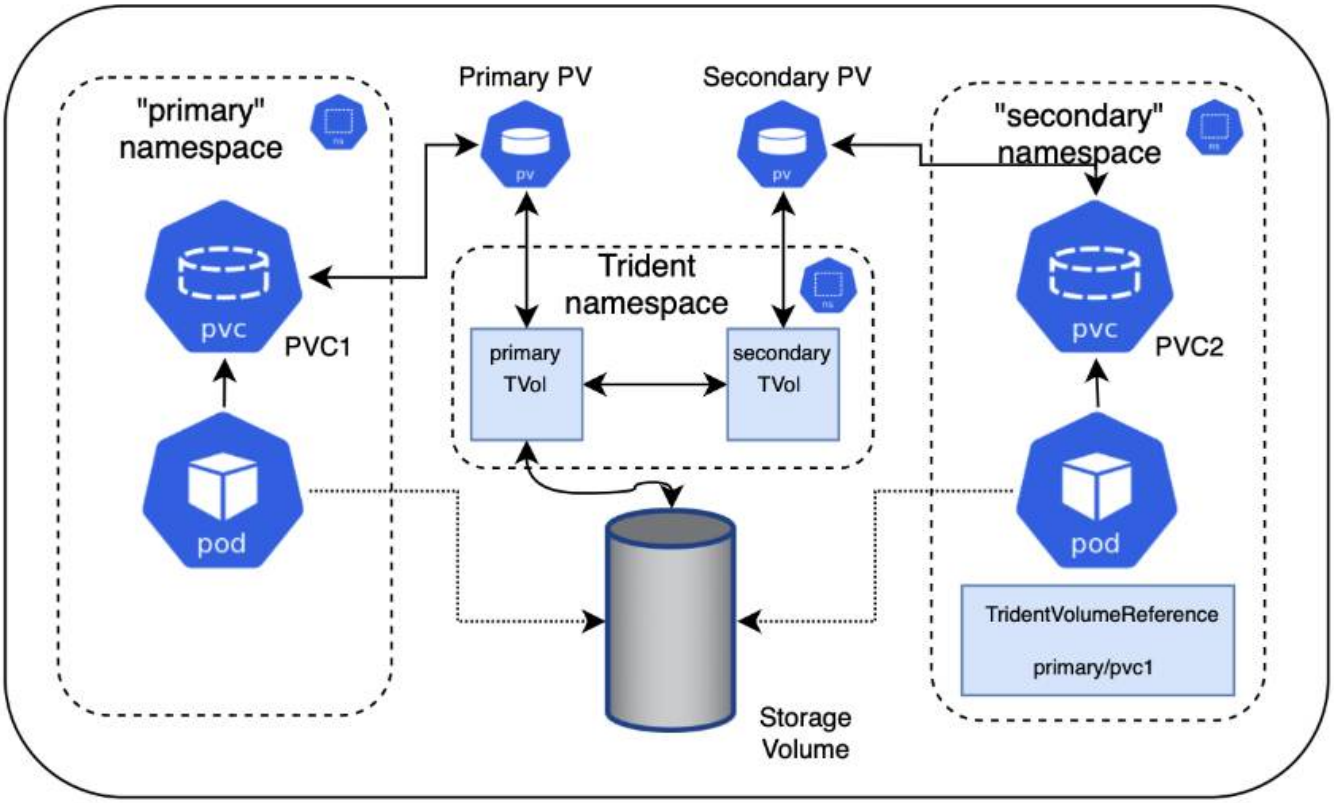
使用Astra Trident、您可以在主命名空间中创建卷、并在一个或多个二级命名空间中共享该卷。

功能

使用Astra TridentVolumeReference CR、您可以在一个或多个Kubernetes命名空间之间安全地共享ReadWriteMany (rwx) NFS卷。此Kubernetes本机解决方案 具有以下优势：

- 可通过多个级别的访问控制来确保安全性
- 适用于所有Trident NFS卷驱动程序
- 不依赖于tridentctl或任何其他非本机Kubernetes功能

此图显示了两个Kubernetes命名空间之间的NFS卷共享。



快速入门

只需几个步骤即可设置NFS卷共享。

1. 配置源**PVC**以共享卷

源命名空间所有者授予访问源PVC中数据的权限。

2 授予在目标命名空间中创建**CR**的权限

集群管理员向目标命名空间的所有者授予创建TridentVolumeReference CR的权限。

3 在目标命名空间中创建**TridentVolumeReference**

目标命名空间的所有者将创建TridentVolumeReference CR以引用源PVC。

4 在目标命名空间中创建从属**PVC**

目标命名空间的所有者创建从属PVC以使用源PVC中的数据源。

配置源和目标命名空间

为了确保安全性、跨命名空间共享需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。每个步骤都会指定用户角色。

步骤

1. *源命名空间所有者：*创建PVC (`pvc1`)、以授予与目标命名空间共享的权限 (`namespace2`) `shareToNamespace` 标注。

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

Astra Trident会创建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命名空间 `*`。例如：`trident.netapp.io/shareToNamespace: *`
- 您可以更新PVC以包括 `shareToNamespace` 随时添加标注。

2. *集群管理员：*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TridentVolumeReference CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. *目标命名空间所有者：*创建PVC (`pvc2`) (`namespace2`) `shareFromPVC` 用于指定源PVC的标注。

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

目标PVC的大小必须小于或等于源PVC。

结果

Astra Trident读取 `shareFromPVC` 在目标PVC上添加标注、并将目标PV创建为一个从属卷、而其自身没有指向源PV的存储资源、并共享源PV存储资源。目标PVC和PV显示为正常绑定。

删除共享卷

您可以删除跨多个命名空间共享的卷。Astra Trident将删除对源命名空间上卷的访问、并保持对共享该卷的其他命名空间的访问。删除引用卷的所有命名空间后、Astra Trident将删除该卷。

使用 ... `tridentctl get` 查询从属卷

使用[`tridentctl` 实用程序中、您可以运行 `get` 用于获取从属卷的命令。有关详细信息、请参见链接：./trident reference/tridentctl.html[`tridentctl` 命令和选项]。

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

flags

- `-h, --help`：卷帮助。
- `--parentOfSubordinate string`：将查询限制为从源卷。
- `--subordinateOf string`：将查询限制为卷的下属。

限制

- Astra Trident无法阻止目标命名空间写入共享卷。您应使用文件锁定或其他进程来防止覆盖共享卷数据。
- 您不能通过删除来撤消对源PVC的访问 `shareToNamespace` 或 `shareFromNamepace` 标注或删除 `TridentVolumeReference` CR.要撤消访问、必须删除从属PVC。
- 无法在从属卷上执行快照、克隆和镜像。

有关详细信息 ...

要了解有关跨命名空间卷访问的详细信息、请执行以下操作：

- 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"。

监控 Astra Trident

Astra Trident 提供了一组 Prometheus 指标端点，您可以使用这些端点监控 Astra Trident 的性能。

通过 Astra Trident 提供的指标，您可以执行以下操作：

- 保留有关 Astra Trident 运行状况和配置的选项卡。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。您可以了解 Astra Trident 与后端通信并执行操作所需的时间。

默认情况下、Trident的指标会显示在目标端口上 `8001` 在上 `/metrics` 端点。安装 Trident 时，这些指标默认为 * 已启用 * 。

您需要的内容

- 安装了 Astra Trident 的 Kubernetes 集群。
- 一个 Prometheus 实例。可以是 "容器化 Prometheus 部署" 或者，您也可以选择将 Prometheus 作为运行 "原生应用程序"。

第 1 步：定义 Prometheus 目标

您应定义一个 Prometheus 目标以收集指标并获取有关后端 Astra Trident 管理的信息，它创建的卷等。这 "博客" 介绍如何将 Prometheus 和 Grafana 与 Astra Trident 结合使用来检索指标。博客介绍了如何在 Kubernetes 集群中以操作员身份运行 Prometheus ，以及如何创建 ServiceMonitor 来获取 Astra Trident 的指标。

第 2 步：创建 Prometheus ServiceMonitor

要使用Trident指标、您应创建一个监控的Prometheus ServiceMonitor `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定义将检索返回的指标 `trident-csi` 服务、并专门查找 `metrics` 服务的端点。因此，Prometheus 现在已配置为了解 Astra Trident 的指标。

除了直接从Astra Trident获得的指标之外、kubelet还公开了许多指标 `kubelet_volume_*` 通过自己的指标端点查看指标。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。请参见 "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 Astra 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 Astra 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 Astra Trident 执行的操作的平均持续时间（毫秒） *

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

获取 **Astra Trident** 使用信息

- 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
```

了解有关 **Astra Trident AutoSupport** 遥测的信息

默认情况下， Astra Trident 会按每日节奏向 NetApp 发送 Prometheus 指标和基本后端信息。

- 要阻止Astra Trident向NetApp发送Prometheus指标和基本后端信息、请传递 `--silence-autosupport` 在Astra Trident安装期间标记。
- Astra Trident还可以根据需要将容器日志发送到NetApp支持部门 `tridentctl send autosupport`。您需要触发 Astra Trident 以上传其日志。在提交日志之前，您应接受 NetApp 的https://www.netapp.com/company/legal/privacy-policy/["隐私政策"^]。
- 除非另有说明，否则 Astra Trident 会从过去 24 小时提取日志。
- 您可以使用指定日志保留时间范围 `--since` 标志。例如： `tridentctl send autosupport --since=1h`。此信息通过收集和发送 `trident-autosupport` 与Astra Trident一起安装的容器。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"。

Astra Trident 发送的有效负载示例如下：

```
{
  "items": [
    {
      "backendUUID": "ff3852e1-18a5-4df4-b2d3-f59f829627ed",
      "protocol": "file",
      "config": {
        "version": 1,
        "storageDriverName": "ontap-nas",
        "debug": false,
        "debugTraceFlags": null,
        "disableDelete": false,
        "serialNumbers": [
          "nwkvzfanek_SN"
        ],
        "limitVolumeSize": ""
      },
      "state": "online",
      "online": true
    }
  ]
}
```

- AutoSupport 消息将发送到 NetApp 的 AutoSupport 端点。如果您使用私有注册表存储容器映像、则可以使用 `--image-registry` 标志。
- 您也可以通过生成安装 YAML 文件来配置代理 URL 。可以使用完成此操作 `tridentctl install --generate-custom-yaml` 创建YAML文件并添加 `--proxy-url` 的参数 `trident-autosupport` 容器 `trident-deployment.yaml`。

禁用 **Astra Trident** 指标

要*禁止报告指标、应使用生成自定义YAML `--generate-custom-yaml` 标志)并对其进行编辑以删除 `--metrics` 用于调用的标志 `trident-main`容器。

适用于 Docker 的 Astra Trident

部署的前提条件

在部署 Astra Trident 之前，您必须在主机上安装和配置必要的协议前提条件。

验证要求

- 验证您的部署是否满足所有要求 "要求"。
- 验证您是否安装了受支持的 Docker 版本。如果您的 Docker 版本已过时， "安装或更新它"。

```
docker --version
```

- 验证是否已在主机上安装和配置协议前提条件：

协议	操作系统	命令
NFS	RHEL/CentOS 7.	`sudo yum install -y nfs-utils`
NFS	Ubuntu	`sudo apt-get install -y nfs-common`

协议	操作系统	命令

iSCSI	RHEL/CentOS 7.	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` 下 `defaults`。
		
		5. 请确保 `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`

协议	操作系统	命令

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` 下 `defaults`。
		
		5. 请确保 `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`

部署 Astra Trident

```
sudo systemctl status
snap.docker.*
```

适用于 Docker 的 Astra Trident 可直接与适用于 NetApp 存储平台的 Docker 生态系统集成。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。

Astra Trident 的多个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"。确保满足前提条件后，即可部署 Astra Trident 。

Docker 托管插件方法（ 1.13/17.03 及更高版本）

开始之前

如果您在传统守护进程方法中使用了 Astra Trident 之前的 Docker 1.13/ 17.03 ，请确保在使用受管插件方法之前停止 Astra Trident 进程并重新启动 Docker 守护进程。

1. 停止所有正在运行的实例：

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. 重新启动 Docker 。

```
systemctl restart docker
```

3. 确保已安装 Docker 引擎 17.03 （新版本 1.13 ）或更高版本。

```
docker --version
```

如果您的版本已过期， "安装或更新安装"。

步骤

1. 创建配置文件并按如下所示指定选项：
   - `config`：默认文件名为 `config.json`但是、您可以通过指定来使用所选的任何名称 `config` 选项和文件名。配置文件必须位于中 `/etc/netappdvp` 主机系统上的目录。
   - `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": "secret",
    "aggregate": "aggr1"
}
EOF
```

2. 使用受管插件系统启动 Astra Trident 。替换 `<version>` 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 Astra Trident 消耗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/v22.10.0/trident-
installer-22.10.0.tar.gz
tar zxf trident-installer-22.10.0.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": "secret",
    "aggregate": "aggr1"
}
EOF
```

2. 放置二进制文件并创建配置文件后，使用所需的配置文件启动 Trident 守护进程。

```
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 **Astra Trident**

有关基于系统d的系统的示例单元文件、请参见 `contrib/trident.service.example` 在Git repo.要在CentOS 7/RHEL中使用此文件、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。

   此名称因您在中命名文件而异 `/usr/lib/systemd/system` 目录。

```
systemctl enable trident
```

5. 启动服务。

```
systemctl start trident
```

6. 查看状态。

```
systemctl status trident
```

每当您修改单元文件时、请运行 `systemctl daemon-reload` 命令以使其能够识别所做的更改。

升级或卸载 Astra Trident

您可以安全地升级适用于 Docker 的 Astra Trident ，而不会对正在使用的卷产生任何影响。在升级过程中、会有一段短暂的时间 `docker volume` 定向到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

执行以下步骤以升级适用于 Docker 的 Astra 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
```

> Astra Trident 18.01 版取代了 nDVP 。您应直接从升级 `netapp/ndvp-plugin` 以图像形式显示到 `netapp/trident-plugin` 图像。

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
```

如果要从旧版本的 Astra Trident （ 20.10 之前的版本）升级到 Astra Trident 20.10 或更高版本，则可能会遇到错误。有关详细信息，请参见 "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、再通过传递一个额外的配置参数来安装所需的Astra 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 Astra 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

您可以使用标准轻松创建、克隆和删除卷 `docker volume` 根据需要指定了Astra Trident驱动程序名称的命令。

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
```

- 使用特定的 Astra Trident 实例创建卷：

```
docker volume create -d ntap_bronze --name bronzeVolume
```

如果未指定任何 "选项"，将使用驱动程序的默认值。

- 覆盖默认卷大小。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
```

使用时 `solidfire-san` 驱动程序中、上述示例将删除并清除卷。

执行以下步骤以升级适用于 Docker 的 Astra Trident 。

克隆卷

使用时 `ontap-nas`， `ontap-san`， `solidfire-san`，和 `gcp-cvs storage drivers`、Astra Trident可以克隆卷。使用时 `ontap-nas-flexgroup` 或 `ontap-nas-economy` 驱动程序、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Astra Trident支持其文件系统(例如：an)、则可以使用Trident * only *通过容器访问外部创建的块设备(或其克隆) `ext4`格式化 `/dev/sdc1` 无法通过Astra Trident访问)。

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。使用提供选项和值 `-o` 在命令行界面操作期间执行此操作。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	Description
`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`、这意味着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。通过浏览到、可以恢复快照中保留的数据 `.snapshot` 卷中任意目录中的目录。
`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，但节省的存储很少，因此最好立即拆分克隆。
`encryption`	在新卷上启用NetApp卷加密(NVE)；默认为 `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Astra Trident中配置的任何卷都将启用NAE。 有关详细信息、请参见： "Astra 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	Description
unixPermissions	此选项用于控制为卷本身设置的权限。默认情况下、权限将设置为 `---rwxr-xr-x`或以数字表示法0755、和 `root 将成为所有者。文本或数字格式均可使用。
snapshotDir	将其设置为 true 将创建 .snapshot 访问卷的客户端可以看到的目录。默认值为 false、表示的可见性 .snapshot 默认情况下、目录处于禁用状态。某些映像(例如官方MySQL映像)在运行时无法按预期运行 .snapshot 目录可见。
exportPolicy	设置要用于卷的导出策略。默认值为 default。
securityStyle	设置用于访问卷的安全模式。默认值为 unix。有效值为 unix 和 mixed。

以下附加选项适用于 iSCSI * 仅 * ：

选项	Description
fileSystemType	设置用于格式化 iSCSI 卷的文件系统。默认值为 ext4。有效值为 ext3， ext4，和 xfs。
spaceAllocation	将其设置为 false 将关闭LUN的空间分配功能。默认值为 true、表示当卷空间用尽且卷中的LUN无法接受写入时、ONTAP 会向主机发出通知。此选项还允许 ONTAP 在主机删除数据时自动回收空间。

示例

请参见以下示例：

- 创建 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`、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、将使用指定与其关联的QoS策略 `-o type=service_level` 术语。

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

其他 Element 软件卷创建选项包括：

选项	**Description**
`size`	卷的大小，默认为 1GiB 或配置条目 … " 默认值 " ： { "size" ： "5c" } 。

选项	Description
`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
```

GCP 上的 **CVS** 卷选项

基于 GCP 的 CVS 驱动程序的卷创建选项包括以下内容：

选项	Description
`size`	卷的大小默认为100 GiB。
`serviceLevel`	卷的 CVS 服务级别默认为标准。有效值包括标准，高级和极高。
`snapshotReserve`	此操作会将快照预留设置为所需百分比。默认值为 no 值，表示 CVS 将选择快照预留（通常为 0% ）。

示例

- 创建 2 TiB 卷：

```
docker volume create -d netapp --name demo -o size=2T
```

- 创建 5 TiB 高级卷：

```
docker volume create -d netapp --name demo -o size=5T -o
serviceLevel=premium
```

最小卷大小为100 GiB。

Azure NetApp Files 卷选项

Azure NetApp Files 驱动程序的卷创建选项包括：

选项	Description
`size`	卷的大小默认为 100 GB 。

示例

- 创建 200 GiB 卷：

```
docker volume create -d netapp --name demo -o size=200G
```

最小卷大小为 100 GB 。

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. 如果您使用建议的托管插件方法(例如、使用 `docker plugin` 命令)、请按如下所示查看它们：

```
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、则主机中会提供日志 `/var/log/netappdvp` 目录。要启用调试日志记录、请指定 `-debug` 运行插件时。

一般故障排除提示

- 新用户遇到的最常见问题是配置不当，导致插件无法初始化。如果发生这种情况，在尝试安装或启用插件时，您可能会看到如下消息：

```
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、请确保这样 `rpcbind` 已安装且正在运行。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。您可以通过运行来检查rpcbind服务的状态 `systemctl status rpcbind` 或其等效项。

管理多个 Astra 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。多个实例的关键是使用为其提供不同的名称 `--alias` 选项、或者 `--volume-driver` 在主机上实例化Trident时的选项。

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 Astra Trident 配置的配置选项。

全局配置选项

这些配置选项适用于所有 Astra Trident 配置，而不考虑所使用的存储平台。

选项	**Description**	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`， `azure-netapp-files`或 `gcp-cvs`
`storagePrefix`	卷名称的可选前缀。默认值："netappdvp_ "。	暂存 _
`limitVolumeSize`	卷大小的可选限制。默认值： "" （未强制实施）	10 G

请勿使用 `storagePrefix` 元素后端的(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` 选项可用于所有控制器类型。有关如何设置默认卷大小的示例，请参见 ONTAP 配置一节。

选项	Description	示例
`size`	新卷的可选默认大小。默认值："1G"	10 G

ONTAP 配置

除了上述全局配置值之外，在使用 ONTAP 时，还可以使用以下顶级选项。

选项	Description	示例
`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；如果未指定，则派生此地址。。 `ontap-nas` 驱动程序*仅*。您可以指定FQDN、在这种情况下、FQDN将用于NFS挂载操作。。 `ontap-san` 驱动程序、默认情况下使用SVM中的所有数据LIF IP并使用iSCSI多路径。指定的IP地址 `dataLIF` 。 `ontap-san` 驱动程序强制驱动程序禁用多路径、并且仅使用指定的地址。	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	机密
`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 ）。。 `ontap-nas-flexgroup` 驱动程序、此选项将被忽略。分配给 SVM 的所有聚合都用于配置 FlexGroup 卷。	aggr1.

选项	Description	示例
`limitAggregateUsage`	可选，如果使用量超过此百分比，则配置失败	75%
`nfsMountOptions`	对 NFS 挂载选项进行精细控制；默认为 -o 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"。	-o nfsver=4
`igroupName`	插件使用的 igroup ；默认为 "netappdvp " 。* 仅适用于 `ontap-san`driver* 。	myigroup
`limitVolumeSize`	可请求的最大卷大小和 qtree 父卷大小。用于 **`ontap-nas-economy`** 驱动程序、此选项还会限制其创建的**FlexVol**的大小。	300 克
`qtreesPerFlexvol`	每个 FlexVol 的最大 qtree 数必须在 50 ， 300 范围内，默认值为 200 。*用于 `ontap-nas-economy` 驱动程序、此选项允许自定义每个qtree*的最大FlexVol 数。	300

您可以使用默认选项来避免在创建的每个卷上指定这些选项：

选项	Description	示例
`spaceReserve`	空间预留模式； " 无 " （精简配置）或 " 卷 " （厚）	无
`snapshotPolicy`	要使用的 Snapshot 策略，默认值为 " 无 "	无
`snapshotReserve`	Snapshot 预留百分比，默认值为 " " 以接受 ONTAP 的默认值	10
`splitOnClone`	创建克隆时将其从父级拆分，默认为 "false"	false

选项	Description	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Astra Trident中配置的任何卷都将启用NAE。 有关详细信息、请参见： "Astra Trident如何与NVE和NAE配合使用"。	true
unixPermissions	已配置 NFS 卷的 NAS 选项，默认为 "777"	777.
snapshotDir	用于访问的NAS选项 .snapshot 目录、默认为"false"	true
exportPolicy	要使用的 NFS 导出策略的 NAS 选项，默认为 " 默认 "	default
securityStyle	用于访问已配置 NFS 卷的 NAS 选项，默认为 "UNIX"	混合
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 SVM-DR 之前的配置，默认为 " 无 " ； " 仅快照 "	无

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、则会显示 ontap-nas 由于FlexVol提供了其他功能、例如Docker卷粒度快照和克隆、因此驱动程序是首选NAS解决方案。

如果所需的Docker卷数超过FlexVol 限制所能容纳的数量、请选择 ontap-nas-economy 或 ontap-san-economy 驱动程序。

。 ontap-nas-economy 驱动程序会在一个自动管理的ONTAP 卷池中将Docker卷创建为FlexVol qtree。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。。 ontap-nas-economy 驱动程序不支持Docker卷粒度快照或克隆。

。 `ontap-nas-economy` 目前、Docker Swarm不支持驱动程序、因为Swarm不会跨多个节点编排卷创建。

。 `ontap-san-economy` 驱动程序会在一个由自动管理的FlexVol构成的共享池中将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` 驱动程序、用于将并行性提高到单个卷、该卷可以扩展到包含数十亿个文件的PB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 大小为 100 GB 。
- 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、其中一个使用 `ontap-nas` 另一种方法是使用 `ontap-nas-economy`。

ONTAP 配置文件示例

的* NFS示例 `ontap-nas` 驱动程序*

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "defaults": {
      "size": "10G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

的* NFS示例 `ontap-nas-flexgroup` 驱动程序*

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "defaults": {
      "size": "100G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

的* NFS示例 `ontap-nas-economy` 驱动程序*

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1"
}
```

的* iSCSI示例 `ontap-san` 驱动程序*

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "igroupName": "myigroup"
}
```

的* NFS示例 `ontap-san-economy` 驱动程序*

```
{
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi_eco",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "igroupName": "myigroup"
}
```

Element 软件配置

除了全局配置值之外，在使用 Element 软件（ NetApp HCI/SolidFire ）时，还可以使用这些选项。

选项	Description	示例
`Endpoint`	<a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">https://<login>:<password>@<mvip>/json-rpc/<element-version>;	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IP 地址和端口	10.0.0.7 ： 3260
`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 Trident 版本并对现有卷执行升级，则需要设置此值才能访问通过 volume-name 方法映射的旧卷。	"netappdvp- "

。 `solidfire-san` 驱动程序不支持Docker Swarm。

Element 软件配置文件示例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

GCP 配置上的 Cloud Volumes Service （ CVS ）

Astra Trident支持默认CVS服务类型为on的卷 "GCP"。无论CVS服务类型允许的最小值如何、Astra Trident都不支持小于100 GiB的CVS卷。因此、如果请求的卷小于最小大小、Trident会自动创建100 GiB卷。

除了全局配置值之外，在 GCP 上使用 CVS 时，还可以使用这些选项。

选项	Description	示例
apiRegion	CVS 帐户区域（必需）。是此后端将配置卷的 GCP 区域。	"us-west2"
projectNumber	GCP 项目编号（必需）。可以在 GCP Web 门户的主屏幕中找到。	"123456789012"
hostProjectNumber	GCP 共享 VPC 主机项目编号（如果使用共享 VPC ，则为必填项）	"098765432109"
apiKey	具有 CVS 管理员角色的 GCP 服务帐户的 API 密钥（必需）。是 GCP 服务帐户专用密钥文件的 JSON 格式内容（逐字复制到后端配置文件）。服务帐户必须具有 netappcloudvolumes.admin 角色。	（私钥文件的内容）
secretKey	CVS 帐户密钥（必需）。可以在 CVS Web 门户中的 "Account settings" >"API access" 中找到。	default
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，系统会跳过证书验证，以允许在代理服务器中使用自签名证书。* 不支持已启用身份验证的代理服务器 * 。	http://proxy-server-hostname/”
nfsMountOptions	NFS 挂载选项；默认为 -o nfsver=3	"nfsver=3 ， proto=tcp ，timeo=600"
serviceLevel	性能级别（标准，高级，极高），默认为 " 标准 "	高级版
network	用于 CVS 卷的 GCP 网络，默认为 " 默认 "	default

如果使用共享VPC网络、则应同时指定这两者 `projectNumber` 和 `hostProjectNumber`。在这种情况下、 `projectNumber` 是服务项目和 `hostProjectNumber` 是主机项目。

在 GCP 上使用 CVS 时，可以使用这些默认卷选项设置。

选项	Description	示例
exportRule	NFS 访问列表（地址和 / 或 CIDR 子网），默认为 "0.0.0.0/0 "	"10.0.1.0/24 10.0.0.2.100"
snapshotDir	控制的可见性 .snapshot 目录	false
snapshotReserve	Snapshot 预留百分比，默认值为 " " 以接受 CVS 默认值 0	10
size	卷大小，默认为 "100GiB"	"10T"

GCP 配置文件上的 **CVS** 示例

```
{
    "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-----\n",
        "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/"
}
```

Azure NetApp Files 配置

配置和使用 "Azure NetApp Files" 后端，您需要满足以下要求：

- `subscriptionID` 从启用了Azure NetApp Files 的Azure订阅
- `tenantID`， `clientID`，和 `clientSecret` 从 "应用程序注册" 在 Azure Active Directory 中，具有足够的 Azure NetApp Files 服务权限
- 至少包含一个的 Azure 位置 "委派子网"

如果您是首次使用 Azure NetApp Files 或在新位置使用，则需要对进行一些初始配置 "《快速入门指南》" 将引导您完成操作。

Astra Trident 21.04.0 及更早版本不支持手动 QoS 容量池。

选项	Description	Default
`version`	始终为 1	
`storageDriverName`	"azure-netapp-files"	
`backendName`	存储后端的自定义名称	驱动程序名称 + "_" + 随机字符
`subscriptionID`	Azure 订阅中的订阅 ID	
`tenantID`	应用程序注册中的租户 ID	
`clientID`	应用程序注册中的客户端 ID	
`clientSecret`	应用程序注册中的客户端密钥	
`serviceLevel`	" 标准 " ， " 高级 " 或 " 超 " 之一	" （随机）
`location`	将在中创建新卷的 Azure 位置名称	" （随机）
`virtualNetwork`	具有委派子网的虚拟网络的名称	" （随机）
`subnet`	委派给的子网的名称 `Microsoft.Netapp/volumes`	" （随机）
`nfsMountOptions`	精细控制 NFS 挂载选项	-o nfsver=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）

Azure NetApp Files 服务不支持小于 100 GB 的卷。为了便于部署应用程序，如果请求的卷较小， Trident 会自动创建 100 GB 的卷。

您可以在配置的特殊部分中使用这些选项来控制默认配置每个卷的方式。

选项	Description	Default
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	控制的可见性 .snapshot 目录	false
size	新卷的默认大小	"100G"

Azure NetApp Files 配置示例

- 示例 1： azure-netapp-files* 的最小后端配置

这是绝对的最低后端配置。使用此配置， Trident 将发现全球每个位置委派给 ANF 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上。

当您刚开始使用 ANF 并尝试执行以下操作时，此配置非常有用： 但实际上，您将需要为您配置的卷提供额外的范围界定，以确保这些卷具有所需的特征，并最终位于一个靠近使用该卷的计算的网络上。有关详细信息，请参见后续示例。

```
{
    "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"
}
```

- 示例 2： azure-netapp-files* 的单一位置和特定服务级别

此后端配置会将卷放置在 Azure 的 " 东向 " 位置的 " 高级 " 容量池中。Trident 会自动发现该位置委派给 ANF 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": "Premium"
}
```

- 示例 3： azure-netapp-files* 的高级配置

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": "Premium",
    "virtualNetwork": "my-virtual-network",
    "subnet": "my-subnet",
    "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
    "limitVolumeSize": "500Gi",
    "defaults": {
        "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
        "size": "200Gi"
    }
}
```

- 示例 4： 使用 azure-netapp-files* 的虚拟存储池

此后端配置定义了多个 "存储池" 在单个文件中。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。

这只是擦除虚拟存储池及其标签的强大功能表面。

```
{
    "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",
    "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
    "labels": {
        "cloud": "azure"
    },
    "location": "eastus",

    "storage": [
        {
            "labels": {
                "performance": "gold"
            },
            "serviceLevel": "Ultra"
        },
        {
            "labels": {
                "performance": "silver"
            },
            "serviceLevel": "Premium"
        },
        {
            "labels": {
                "performance": "bronze"
            },
            "serviceLevel": "Standard",
        }
    ]
}
```

已知问题和限制

查找有关将 Astra Trident 与 Docker 结合使用时的已知问题和限制的信息。

将 **Trident Docker** 卷插件从旧版本升级到 **20.10** 及更高版本会导致升级失败，并且不会显示此类文件或目录错误。

临时解决策

1. 禁用插件。

```
docker plugin disable -f netapp:latest
```

2. 删除此插件。

```
docker plugin rm -f netapp:latest
```

3. 通过提供额外的插件重新安装插件 `config` 参数。

```
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名称解释为 Windows 路径。 "请参见错误 25773"。

Docker Swarm 的某些行为会阻止 **Astra Trident** 在每个存储和驱动程序组合中为其提供支持。

- 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 Astra Trident ）必须在 Swarm 集群中的每个节点上独立运行。由于ONTAP 的工作方式和的方式 `ontap-nas` 和 `ontap-san` 驱动程序正常运行、但它们恰好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** 。

常见问题解答

查找有关 Astra Trident 的安装，配置，升级和故障排除的常见问题解答。

一般问题

Astra Trident 的发布频率如何?

Astra Trident 每三个月发布一次： 1 月， 4 月， 7 月和 10 月。这是 Kubernetes 发布后一个月。

Astra Trident 是否支持特定版本的 Kubernetes 中发布的所有功能?

Astra Trident 通常不支持 Kubernetes 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。

Astra Trident 的运行是否依赖于其他 NetApp 产品?

Astra Trident 与其他 NetApp 软件产品没有任何依赖关系，它可以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 Astra Trident 配置详细信息?

使用 `tridentctl get` 命令以获取有关Astra Trident配置的详细信息。

我能否获取有关 Astra Trident 如何配置存储的指标?

是的。Trident 20.01 引入了 Prometheus 端点，可用于收集有关 Astra Trident 操作的信息，例如托管的后端数量，配置的卷数量，占用的字节数等。您还可以使用 Cloud Insights 进行监控和分析。

使用 Astra Trident 作为 CSI 配置程序时，用户体验是否会发生变化?

否在用户体验和功能方面没有变化。使用的配置程序名称是 `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 Astra Trident 。

在 Kubernetes 集群上安装和使用 Astra Trident

支持的版本是什么 `etcd`?

Astra Trident不再需要 `etcd`。它使用 CRD 来保持状态。

Astra Trident 是否支持从专用注册表脱机安装?

可以， Astra Trident 可以脱机安装。请参见 "此处"。

是否可以远程安装 **Astra Trident**？

是的。Astra Trident 18.10及更高版本支持从具有的任何计算机远程安装 `kubectl` 对集群的访问。之后 `kubectl` 访问已验证(例如、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 **Astra Trident** 配置高可用性?

Astra Trident 以 Kubernetes 部署（ ReplicaSet ）的形式安装有一个实例，因此它具有内置的 HA 。您不应增加部署中的副本数量。如果安装了 Astra Trident 的节点丢失或 POD 无法访问， Kubernetes 会自动将 POD 重新部署到集群中运行正常的节点。Astra Trident 仅支持控制平面，因此，如果重新部署 Astra Trident ，当前安装的 Pod 不会受到影响。

Astra Trident 是否需要访问 **Kube-system** 命名空间?

Astra Trident 从 Kubernetes 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 。

Astra Trident 使用哪些角色和特权?

Trident 安装程序会创建一个 Kubernetes ClusterRole ，该 ClusterRole 可对集群的 PersistentVolume ， PersistentVolumeClaim ， StorageClass 和 Kubernetes 集群的 Secret 资源进行特定访问。请参见 "此处"。

是否可以在本地生成 **Astra Trident** 用于安装的准确清单文件?

如果需要，您可以在本地生成和修改 Astra Trident 用于安装的确切清单文件。请参见 "此处"。

是否可以为两个单独的 **Kubernetes** 集群的两个单独的 **Astra Trident** 实例共享同一个 **ONTAP** 后端 **SVM**？

尽管不建议这样做，但您可以对两个 Astra Trident 实例使用同一个后端 SVM 。在安装期间为每个实例指定唯一的卷名称和/或指定唯一的卷名称 `StoragePrefix` 中的参数 `setup/backend.json` 文件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 **ContainerLinux** （以前称为 **CoreOS** ）下安装 **Astra Trident**？

Astra Trident 只是 Kubernetes Pod ，可安装在 Kubernetes 运行的任何位置。

是否可以将 **Astra Trident** 与 **NetApp Cloud Volumes ONTAP** 结合使用?

是的， AWS ， Google Cloud 和 Azure 支持 Astra Trident 。

Astra Trident 是否支持 **Cloud Volumes Services**？

是的， Astra 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 Astra Trident ？

虽然 Astra Trident 是开源且免费提供的，但只要您的 NetApp 后端受支持， NetApp 就会完全支持它。

如何提出支持案例?

要提交支持案例，请执行以下操作之一：

1. 请联系您的支持客户经理并获得帮助以提交服务单。
2. 联系以提出支持案例 "NetApp 支持"。

如何生成支持日志包?

您可以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?

在上创建问题描述 "Astra Trident Github" 并在问题描述的主题和问题描述中提及 * RFE* 。

我应在何处提出缺陷?

在上创建问题描述 "Astra Trident Github"。请务必包含与问题描述相关的所有必要信息和日志。

如果我有有关 Astra Trident 的快速问题需要澄清，会发生什么情况？是否有社区或论坛？

如有任何问题、问题或请求、请通过我们的Astra联系我们 "渠道不和" 或GitHub。

我的存储系统密码已更改， Astra Trident 不再工作，如何恢复?

使用更新后端的密码 `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。替换 `myBackend` 在示例中、使用后端名称、和 `/path/to_new_backend.json` 路径正确 `backend.json` 文件

Astra Trident 找不到我的 Kubernetes 节点。如何修复此问题?

Astra Trident 无法找到 Kubernetes 节点的原因可能有两种。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 Astra Trident 后发生网络更改，则只有在添加到集群中的新 Kubernetes 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。Trident 的元数据存储在 CRD 对象中。已由 Trident 配置的所有 PV 都将正常运行。

升级 Astra 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 Astra Trident 从一个主要版本升级到下一个直接主要版本。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?

如果要降级，需要评估许多因素。请参见 "有关降级的章节"。

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 ？

NetApp 建议在后端定义文件中同时包含这两者。但是，只有管理 LIF 是必需的。

Astra Trident 是否可以为 ONTAP 后端配置 CHAP ？

是的。从 20.04 开始， Astra Trident 支持 ONTAP 后端的双向 CHAP 。这需要设置 `useCHAP=true` 在后端配置中。

如何使用 Astra Trident 管理导出策略?

Astra Trident 可以从 20.04 版开始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， Astra Trident 会自动管理为给定 CIDR 中具有 IP 的节点添加和删除规则的操作。此功能需要 CSI Trident 。

是否可以在 DataLIF 中指定端口?

Astra Trident 19.01 及更高版本支持在 DataLIF 中指定端口。在中进行配置 `backend.json` 文件为 `"managementLIF": <ip address>:<port>"`。例如、如果管理LIF的IP地址为192.0.2.1、端口为1000、请配置 `"managementLIF": "192.0.2.1:1000"`。

IPv6 地址是否可用于管理和数据 LIF ？

是的。Astra Trident 20.01 支持为管理 LIF 定义 IPv6 地址，并为 ONTAP 后端定义 dataLIF 参数。您应确保地址遵循IPv6语义、并且管理LIF在方括号内进行定义(例如、`[ec0d:6504:a9c1:ae67:53d1:4bdf:ab32:e233]`）。此外、您还应确保使用安装了Astra Trident `--use-ipv6` 用于通过IPv6正常运行的标志。

是否可以在后端更新管理 LIF ？

可以、可以使用更新后端管理LIF `tridentctl update backend` 命令：

是否可以更新后端的数据 LIF ？

不可以，无法更新后端的数据 LIF 。

是否可以在适用于 **Kubernetes** 的 **Astra Trident** 中创建多个后端?

Astra 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。

Astra Trident 如何存储后端凭据?

Astra Trident 会将后端凭据存储为 Kubernetes Secretes 。

Astra Trident 如何选择特定后端?

如果无法使用后端属性自动为某个类选择合适的池、则会显示 `storagePools` 和 `additionalStoragePools` 参数用于选择一组特定的池。

如何确保 **Astra Trident** 不会从特定后端配置?

。 `excludeStoragePools` 参数用于筛选Astra Trident要用于配置的一组池、并将删除匹配的任何池。

如果存在多个相同类型的后端，则 **Astra Trident** 如何选择要使用的后端?

如果配置了多个相同类型的后端、则Astra Trident会根据中的参数选择相应的后端 `StorageClass` 和 `PersistentVolumeClaim`。例如、如果存在多个ontap-NAS驱动程序后端、则Astra Trident会尝试匹配中的参数 `StorageClass` 和 `PersistentVolumeClaim` 组合并匹配后端、可满足中列出的要求 `StorageClass` 和 `PersistentVolumeClaim`。如果有多个后端与请求匹配，则 Astra Trident 会随机从其中一个后端中进行选择。

Astra Trident 是否支持 **Element** 或 **SolidFire** 的双向 **CHAP** ？

是的。

Astra Trident 如何在 **ONTAP** 卷上部署 **qtree** ？ 一个卷可以部署多少个 **qtree** ？

。 `ontap-nas-economy` 驱动程序可在同一个FlexVol 中创建多达200个qtree (可配置为50到300)、每个集群节点创建100、000个qtree、每个集群创建2.4 M个qtree。输入新的 `PersistentVolumeClaim` 这是由经济型驱动程序提供服务的、该驱动程序会查看是否已存在可为新的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 Astra 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， Astra Trident 不会使用 Kubernetes 将挂载选项设置为任何值。要在 Kubernetes 存储类中指定挂载选项，请按照给定示例进行操作 "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 Astra 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： "Astra Trident 如何与NVE和NAE配合使用"

通过 Astra Trident 为 ONTAP 实施 QoS 的最佳方式是什么?

使用 ... `StorageClasses` 为ONTAP 实施QoS。

如何通过 Astra 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、 `StorageClass` 优先。为 ONTAP 配置以下内容：

1. 开启 `StorageClass`、设置 `provisioningType` 属性为thick。
2. 在后端定义文件中、通过设置启用厚卷 `backend spaceReserve parameter` 作为卷。

如何确保即使意外删除了 PVC 也不会删除所使用的卷?

从版本 1.10 开始， Kubernetes 会自动启用 PVC 保护。

是否可以扩展由 Astra Trident 创建的 NFS PVC ？

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

如果我的卷是在 Astra Trident 外部创建的，是否可以将其导入到 Astra Trident ？

从 19.04 开始，您可以使用卷导入功能将卷引入 Kubernetes 。

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.
```

是否可以扩展由 Astra Trident 创建的 iSCSI PVC ？

Trident 19.10 支持使用 CSI 配置程序扩展 iSCSI PV 。

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。当 NetApp 存储由于容量不足而无法支持 Kubernetes 配额设置时， Astra Trident 会尝试配置，但会出错。

是否可以使用 Astra Trident 创建卷快照?

是的。Astra Trident 支持从快照创建按需卷快照和永久性卷。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 Astra Trident 卷快照?

自目前起、我们为提供按需快照支持 `ontap-nas`， `ontap-nas-flexgroup`， `ontap-san`， `ontap-san-economy`， `solidfire-san`， `gcp-cvs`，和 `azure-netapp-files` 后端驱动程序。

如何为采用 ONTAP 的 Astra Trident 配置的卷创建快照备份?

此功能可从获得 `ontap-nas`， `ontap-san`，和 `ontap-nas-flexgroup` 驱动程序。您也可以指定 `snapshotPolicy`。 `ontap-san-economy` FlexVol 级别的驱动程序。

也可以在上查看此信息 `ontap-nas-economy` 驱动程序、但在FlexVol 级别粒度上、而不在qtree级别粒度上。要启用对Astra Trident配置的卷的快照功能、请设置backend参数选项 `snapshotPolicy` 到ONTAP 后端定义的所需快照策略。Astra Trident 无法识别存储控制器创建的任何快照。

是否可以为通过 Astra Trident 配置的卷设置快照预留百分比?

可以。您可以通过设置来预留特定百分比的磁盘空间、以便通过Astra Trident存储Snapshot副本 `snapshotReserve` 属性。如果已配置 `snapshotPolicy` 和 `snapshotReserve` 在后端定义文件中、快照预留百分比是根据设置的 `snapshotReserve` 后端文件中提及的百分比。如果 `snapshotReserve` 未提及百分比数、默认情况下、ONTAP 会将快照预留百分比设置为5。如果 `snapshotPolicy` 选项设置为none、快照预留百分比设置为0。

是否可以直接访问卷快照目录和复制文件?

可以、您可以通过设置来访问Trident配置的卷上的Snapshot目录 `snapshotDir` 后端定义文件中的参数。

是否可以通过 Astra 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 。

支持

Astra Trident 是一个官方支持的 NetApp 项目。您可以使用任何标准机制联系 NetApp ，并获得所需的企业级支持。

在我们的Astra上、还有一个由容器用户(包括Astra Trident开发人员)参与的活跃公有 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。

故障排除

使用此处提供的指针排除安装和使用 Astra Trident 时可能遇到的问题。

要获得有关Astra Trident的帮助、请使用创建支持包 `tridentctl logs -a -n trident` 并将其发送到 `NetApp Support <Getting Help>`。

有关故障排除文章的完整列表，请参见 "NetApp 知识库（需要登录）"。您还可以找到有关排除与 Astra 相关的问题的信息 "此处"。

常规故障排除

- 如果Trident Pod无法正常启动(例如、当Trident Pod卡在中时) `ContainerCreating` 阶段中的就绪容器少于两个)、正在运行 `kubectl -n trident describe deployment trident` 和 `kubectl -n trident describe pod trident--**` 可以提供更多见解。获取kubelet日志(例如、通过 `journalctl -xeu kubelet`)也很有用。
- 如果Trident日志中的信息不足、您可以尝试通过传递来为Trident启用调试模式 `-d` 根据您的安装选项标记为install参数。

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

  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"期限"列来检查此情况 `kubectl get pod -n trident`。

  对于Astra 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
  ```

- 您也可以通过包括来获取每个后端的调试日志 `debugTraceFlags` 在后端定义中。例如、包括 `debugTraceFlags: {“api”:true, “method”:true,}` 在Trident日志中获取API调用和方法遍历。现有后端可以具有 `debugTraceFlags` 已配置 `tridentctl backend update`。

- 使用RedHat CoreOS时、请确保执行此操作 `iscsid` 在工作节点上启用并默认启动。可以使用 OpenShift MachineConfigs 或修改点燃模板来完成此操作。
- 使用 Trident 时可能会遇到的一个常见问题 "Azure NetApp Files" 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" Configuration
- 如果在将PV挂载到容器时出现问题、请确保这样 `rpcbind` 已安装且正在运行。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。您可以检查的状态 `rpcbind` 通过运行提供服务 `systemctl status rpcbind` 或其等效项。
- Trident后端报告其位于中 `failed` State尽管之前已执行过操作、但可能是由于更改了与后端关联的SVM/admin凭据而导致的。使用更新后端信息 `tridentctl update backend` 或者、放弃Trident POD将修复此问题描述。
- 如果要升级 Kubernetes 集群和 / 或 Trident 以使用测试版卷快照，请确保已完全删除所有现有的 alpha Snapshot CRS 。然后、您可以使用 `tridentctl obliviate alpha-snapshot-crd` 用于删除alpha snapshot CRD的命令。请参见 "本博客" 了解迁移 alpha 快照所涉及的步骤。
- 如果在容器运行时安装Trident时遇到权限问题、请尝试使用安装Trident `--in cluster=false` 标志。这不会使用安装程序POD、并可避免因出现权限问题 `trident-installer` 用户。
- 使用 `uninstall parameter <Uninstalling Trident>` 用于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 `tridenctl uninstall` 用于删除Trident的命令。下载所需的 "Trident 版本" 并使用安装 `tridentctl install` 命令：只有在未创建新 PV 且未对现有 PV/ 后端 / 存储类进行更改的情况下，才考虑降级。由于Trident现在使用CRD来保持状态、因此创建的所有存储实体(后端、存储类、PV和卷快照)都具有 `associated CRD objects <Kubernetes CustomResourceDefinition Objects>` 而不是写入到早期安装的Trident版本所使用的PV中的数据。* 新创建的 PV 在移回早期版本时不可用。 * 降级后， Trident 将无法看到对后端， PV ，存储类和卷快照（已创建 / 更新 / 删除）等对象所做的更改 * 。先前安装的 Trident 版本所使用的 PV 仍可供 Trident 查看。如果返回到早期版本，则不会中断对已使用旧版本创建的 PV 的访问，除非已对其进行升级。
- 要完全删除Trident、请运行 `tridentctl obliviate crd` 命令：此操作将删除所有 CRD 对象并取消定义 CRD 。Trident 将不再管理其已配置的任何 PV 。

  之后，需要从头开始重新配置 Trident 。
- 成功安装后、如果PVC卡在中 `Pending` 阶段、运行 `kubectl describe pvc` 可以提供追加信息 、说明Trident为何无法为此PVC配置PV。

使用操作员对未成功的 Trident 部署进行故障排除

如果使用操作员部署Trident、则为的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态、并且操作员无法自行恢复、您应运行以下命令来检查操作员的日志：

```
tridentctl logs -l trident-operator
```

跟踪 trident 操作器容器的日志可能会指向问题所在。例如，其中一个问题描述可能是无法从运行良好的环境中的上游注册表中提取所需的容器映像。

要了解Trident安装失败的原因、您应查看 `TridentOrchestrator` 状态。

```
kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                          Message
  ----     ------  ----               ----                          -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

此错误表示已存在 `TridentOrchestrator`用于安装Trident。由于每个Kubernetes集群只能有一个Trident实例、因此操作员可确保在任何给定时间只有一个活动实例 `TridentOrchestrator` 它可以创建的内容。

此外，观察 Trident Pod 的状态通常可以指示情况是否不正确。

```
kubectl get pods -n trident

NAME                                READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                   1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw        4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                   1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                   1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv   1/1     Running            0
8m17s
```

您可以清楚地看到，由于未提取一个或多个容器映像， Pod 无法完全初始化。

要解决此问题、您应编辑 `TridentOrchestrator` CR.或者、您也可以删除 `TridentOrchestrator`、并使用修改后的准确定义创建一个新的。

使用对未成功的Trident部署进行故障排除 `tridentctl`

为了帮助确定出现了什么问题、您可以使用重新运行安装程序 `-d` 参数、用于打开调试模式并帮助您了解问题所在：

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

解决问题后、您可以按如下所示清理安装、然后运行 `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.
```

最佳实践和建议

部署

在部署 Astra Trident 时，请使用此处列出的建议。

部署到专用命名空间

"命名空间" 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。Astra Trident 为 Kubernetes 集群中的所有命名空间提供 PV 资源，从而利用具有提升权限的服务帐户。

此外，访问 Trident POD 可能会使用户能够访问存储系统凭据和其他敏感信息。请务必确保应用程序用户和管理应用程序无法访问 Trident 对象定义或 Pod 本身。

使用配额和范围限制来控制存储消耗

Kubernetes 具有两项功能，这些功能结合使用后，可提供一种功能强大的机制来限制应用程序的资源消耗。。 "存储配额机制" 使管理员能够在每个命名空间基础上实施全局容量和对象计数消耗限制以及特定于存储类的限制。此外，使用 "范围限制" 确保在将 PVC 请求转发给配置程序之前，该请求同时处于最小值和最大值范围内。

这些值是按命名空间定义的，这意味着每个命名空间都应定义符合其资源要求的值。有关信息，请参见此处 "如何利用配额"。

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，以实现最大的灵活性 "SVM-DR" 与 Trident 结合使用。

限制最大卷数

ONTAP 存储系统具有最大卷数，具体取决于软件版本和硬件平台。请参见 "NetApp Hardware Universe" 以 ONTAP 确定确切限制。当卷计数用尽时，配置操作不仅会对 Trident 失败，而且会对所有存储请求失败。

Trident的 `ontap-nas` 和 `ontap-san` 驱动程序为创建的每个Kubernetes永久性卷(PV)配置一个FlexVolume。。 `ontap-nas-economy` 驱动程序会为每200个PV创建大约一个FlexVolume (可在50到300之间配置)。。 `ontap-san-economy` 驱动程序会为每100个PV创建大约一个FlexVolume (可在50到200之间配置)。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` 根据您的环境特定的多个条件而有所不同：

- ONTAP 集群中现有卷的数量
- 希望在 Trident 之外为其他应用程序配置的卷数
- Kubernetes 应用程序预期占用的永久性卷数

。 `max-volumes` 值是在ONTAP 集群中的所有节点上配置的总卷数、而不是在单个ONTAP 节点上配置的总卷数。因此，在某些情况下， ONTAP 集群节点所配置的 Trident 卷可能远远多于或少于其他节点。

例如，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，如果只是 "聚合" 从一个节点分配给 SVM ，或者从一个节点分配的聚合无法配置（例如，由于容量），则另一个节点将成为所有 Trident 配置卷的目标。这意味着、可能会在之前达到该节点的卷限制 `max-volumes` 达到值后、会同时影响使用该节点的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Trident可以创建的卷配置最大大小、请使用 `limitVolumeSize` 中的参数 `backend.json` 定义。

除了控制存储阵列上的卷大小之外，您还应利用 Kubernetes 功能。

配置 **Trident** 以使用双向 **CHAP**

您可以在后端定义中指定 CHAP 启动程序以及目标用户名和密码，并在 SVM 上启用 Trident CHAP 。使用

`useCHAP` 参数在后端配置中、Trident使用CHAP对ONTAP 后端的iSCSI连接进行身份验证。Trident 20.04 及更高版本支持双向 CHAP 。

创建并使用 **SVM QoS** 策略

利用应用于 SVM 的 ONTAP QoS 策略，限制 Trident 配置的卷可使用的 IOPS 数量。这有助于实现 "防止抢占资源" 或控制不足的容器，以使其不会影响 Trident SVM 以外的工作负载。

您可以通过几个步骤为 SVM 创建 QoS 策略。有关最准确的信息，请参见适用于您的 ONTAP 版本的文档。以下示例将创建一个 QoS 策略，将 SVM 可用的总 IOPS 限制为 5000 。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

此外，如果您的 ONTAP 版本支持此功能，则可以考虑使用最低 QoS 来保证容器化工作负载的吞吐量。自适应 QoS 与 SVM 级别策略不兼容。

专用于容器化工作负载的 IOPS 数量取决于许多方面。其中包括：

- 使用存储阵列的其他工作负载。如果存在与 Kubernetes 部署无关的其他工作负载，则应注意利用存储资源，以确保这些工作负载不会意外受到不利影响。
- 容器中运行的预期工作负载。如果 IOPS 要求较高的工作负载将在容器中运行，则 QoS 策略较低会导致出现不良体验。

请务必记住，在 SVM 级别分配的 QoS 策略会导致配置到 SVM 的所有卷共享同一个 IOPS 池。如果一个或少量容器化应用程序的 IOPS 要求较高，则可能会成为其他容器化工作负载的抢占资源的应用程序。如果是这种情况，您可能需要考虑使用外部自动化来分配每个卷的 QoS 策略。

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：

- 您应设置 `qosPolicy` 输入 `defaults` 后端配置的块。请参见以下后端配置示例：

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "0.0.0.0",
  "dataLIF": "0.0.0.0",
  "svm": "svm0",
  "username": "user",
  "password": "pass",
  "defaults": {
    "qosPolicy": "standard-pg"
  },
  "storage": [
    {
      "labels": {"performance": "extreme"},
      "defaults": {
        "adaptiveQosPolicy": "extremely-adaptive-pg"
      }
    },
    {
      "labels": {"performance": "premium"},
      "defaults": {
        "qosPolicy": "premium-pg"
      }
    }
  ]
}
```

- 您应该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" 旨在防止这种类型的损害。但是，存在一个例外：特权容器。

有权限的容器是指运行时拥有比正常情况更多主机级别权限的容器。默认情况下，这些选项不会被拒绝，因此请确保使用禁用此功能 "POD 安全策略"。

对于需要从 Kubernetes 和外部主机访问的卷，应采用传统方式管理存储，并由管理员引入 PV ，而不是由

Trident 管理。这样可以确保只有在 Kubernetes 和外部主机断开连接且不再使用此卷时，才会销毁此存储卷。此外，还可以应用自定义导出策略，以便从 Kubernetes 集群节点和 Kubernetes 集群以外的目标服务器进行访问。

对于具有专用基础架构节点（例如 OpenShift ）或其他不能为用户应用程序计划的节点的部署，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。从 20.04 版开始， Trident 可以自动创建和管理导出策略。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` 用于创建导出策略的ONTAP 命令行界面命令。
- 使用向导出策略添加规则 `vserver export-policy rule create` ONTAP 命令行界面命令。

通过运行这些命令，您可以限制哪些 Kubernetes 节点可以访问数据。

禁用 `showmount` 用于应用程序**SVM**

。 `showmount` 通过功能、NFS客户端可以向SVM查询可用NFS导出列表。部署到Kubernetes集群的POD可以对进行问题描述 `showmount -e` 对数据LIF执行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。

您应禁用 `showmount` 使用SVM级别的ONTAP 命令行界面命令：

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire 最佳实践

了解为 Trident 配置 SolidFire 存储的最佳实践。

创建 **SolidFire** 帐户

每个 SolidFire 帐户都代表一个唯一的卷所有者，并接收自己的一组质询握手身份验证协议（ Challenge-Handshake Authentication Protocol ， CHAP ）凭据。您可以使用帐户名称和相对 CHAP 凭据或通过卷访问组访问分配给帐户的卷。一个帐户最多可以分配 2 ， 000 个卷，但一个卷只能属于一个帐户。

创建 **QoS** 策略

如果要创建并保存可应用于多个卷的标准化服务质量设置，请使用 SolidFire 服务质量（ QoS ）策略。

您可以按卷设置 QoS 参数。通过设置三个可配置的参数来定义 QoS ，可以确保每个卷的性能：最小 IOPS ，最大 IOPS 和突发 IOPS 。

以下是 4 KB 块大小的可能最小，最大和突发 IOPS 值。

IOPS 参数	定义	最小value	默认值	最大值（ 4 KB ）
最小 IOPS	卷的性能保障级别。	50	50	15000
最大 IOPS	性能不会超过此限制。	50	15000	200,000
突发 IOPS	在短时突发情形下允许的最大 IOPS 。	50	15000	200,000

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。请参见 "SolidFire 服务质量" 有关 QoS 和性能的详细信息。

SolidFire 身份验证

Element 支持两种身份验证方法： CHAP 和卷访问组（ VAG ）。CHAP 使用 CHAP 协议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。搜索 "NetApp 库" 对于最新版本。

- 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，与您的 NetApp 团队合作并使用非常重要 "NetApp 库" 以查找最新文档。

集成 Astra Trident

要集成 Astra 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，虚拟存储池设计，永久性卷声明（ PVC ）对存储配置的影响，卷操作以及使用 Astra Trident 部署 OpenShift 服务。

驱动程序选择和部署

为存储系统选择并部署后端驱动程序。

ONTAP 后端驱动程序

ONTAP 后端驱动程序可通过所使用的协议以及在存储系统上配置卷的方式来区分。因此、在确定要部署的驱动程序时、请仔细考虑。

更高级别的是，如果您的应用程序中的组件需要共享存储（多个 Pod 访问同一个 PVC ），则基于 NAS 的驱动程序将成为默认选项，而基于块的 iSCSI 驱动程序则可满足非共享存储的需求。根据应用程序要求以及存储和基础架构团队的舒适程度选择协议。一般来说，对于大多数应用程序来说，它们之间没有什么区别，因此通常是根据是否需要共享存储（多个 POD 需要同时访问）来决定的。

可用的ONTAP 后端驱动程序包括：

- `ontap-nas`：配置的每个PV都是一个完整的ONTAP FlexVolume。
- `ontap-nas-economy`：配置的每个PV都是一个qtree、每个FlexVolume具有可配置的qtree数量(默认值为200)。
- `ontap-nas-flexgroup`：使用配置为完整ONTAP FlexGroup 的每个PV以及分配给SVM的所有聚合。
- `ontap-san`：配置的每个PV都是其自身FlexVolume中的一个LUN。
- `ontap-san-economy`：配置的每个PV都是一个LUN、每个FlexVolume具有可配置的LUN数量(默认值为100)。

在三个 NAS 驱动程序之间进行选择会对应用程序可用的功能产生一些影响。

请注意，在下表中，并非所有功能都通过 Astra Trident 公开。如果需要某些功能，存储管理员必须在配置后应用这些功能。上标脚注区分了每个功能和驱动程序的功能。

ONTAP NAS 驱动程序	快照	克隆	动态导出策略	多连接	**QoS**	调整大小	**Replication**
`ontap-nas`	是的。	是的。	是脚注：5[]	是的。	是脚注：1[]	是的。	是脚注：1[]
`ontap-nas-economy`	是脚注：3[]	是脚注：3[]	是脚注：5[]	是的。	是脚注：3[]	是的。	是脚注：3[]
`ontap-nas-flexgroup`	是脚注：1[]	否	是脚注：5[]	是的。	是脚注：1[]	是的。	是脚注：1[]

Astra Trident 为 ONTAP 提供了 2 个 SAN 驱动程序，其功能如下所示。

ONTAP SAN 驱动程序	快照	克隆	多连接	双向 **CHAP**	**QoS**	调整大小	**Replication**
`ontap-san`	是的。	是的。	是脚注：4[]	是的。	是脚注：1[]	是的。	是脚注：1[]
`ontap-san-economy`	是的。	是的。	是脚注：4[]	是的。	是脚注：3[]	是的。	是脚注：3[]

> 上述表的脚注：是脚注： 1[]： 不受 Astra Trident 管理是脚注： 2[]： 由 Astra Trident 管理，但不是 PV 粒度 Yesfootnote： 3[]： 不受 Astra Trident 管理，也不是 PV 粒度 Yesnote： 4[]： 支持原始块卷是脚注： 5[]： CSI 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、之间的大部分功能 `ontap-nas` 和 `ontap-nas-economy` 相同。但是、因为 `ontap-nas-economy` 驱动程序限制了按PV粒度控制计划的能力、这尤其会影响灾难恢复和备份规划。对于希望在ONTAP存储上使用PVC克隆功能的开发团队、只有在使用时才可能实现这一点 `ontap-nas`， `ontap-san` 或 `ontap-san-economy` 驱动程序。

。 `solidfire-san` 驱动程序还可以克隆PVC。

Cloud Volumes ONTAP 后端驱动程序

Cloud Volumes ONTAP 可为各种使用情形提供数据控制以及企业级存储功能，包括文件共享和为 NAS 和 SAN 协议（ NFS， SMB/CIFS 和 iSCSI ）提供服务的块级存储。Cloud Volume ONTAP 的兼容驱动程序包括 `ontap-nas`， `ontap-nas-economy`， `ontap-san` 和 `ontap-san-economy`。它们适用于适用于 Azure 的 Cloud Volume ONTAP ，适用于 GCP 的 Cloud Volume ONTAP 。

适用于**ONTAP** 的**Amazon FSX**后端驱动程序

借助适用于 ONTAP 的 Amazon FSX ，客户可以利用他们熟悉的 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 的许多文件系统功能和管理 API 。Cloud Volume ONTAP 的兼容驱动程序包括 `ontap-nas`， `ontap-nas-economy`， `ontap-nas-flexgroup`， `ontap-san` 和 `ontap-san-economy`。

NetApp HCI/SolidFire后端驱动程序

。 `solidfire-san` 与NetApp HCI/SolidFire平台结合使用的驱动程序可帮助管理员根据QoS限制为Trident配置Element后端。如果您希望设计后端、以便为Trident配置的卷设置特定的QoS限制、请使用 `type` 参数。管理员还可以使用限制在存储上创建的卷大小 `limitVolumeSize` 参数。目前、不支持通过实现卷大小调整和卷复制等Element存储功能 `solidfire-san` 驱动程序。这些操作应通过 Element Software Web UI 手动完成。

SolidFire 驱动程序	快照	克隆	多连接	CHAP	QoS	调整大小	Replication
`solidfire-san`	是的。	是的。	是脚注：2[]	是的。	是的。	是的。	是脚注：1[]

> 脚注：是脚注： 1[]：不由 Astra Trident 管理是脚注： 2[]：支持原始块卷

Azure NetApp Files 后端驱动程序

Astra Trident使用 `azure-netapp-files` 用于管理的驱动程序 "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 "适用于 Azure NetApp Files 的 Astra Trident 后端配置"。

Azure NetApp Files 驱动程序	快照	克隆	多连接	QoS	展开	Replication
`azure-netapp-files`	是的。	是的。	是的。	是的。	是的。	是脚注： 1[]

> 脚注：是脚注： 1[]：不由 Astra Trident 管理

具有GCP后端驱动程序的Cloud Volumes Service

Astra Trident使用 `gcp-cvs` 用于链接到GCP后端Cloud Volumes Service 的驱动程序。要在Trident上配置GCP后端、需要指定 `projectNumber`， `apiRegion`，和 `apiKey` 在后端文件中。项目编号可在 GCP Web 门户中找到，而 API 密钥必须从您在 GCP 上为 Cloud Volumes 设置 API 访问时创建的服务帐户专用密钥文件中获取。Astra Trident 可以在两个卷中创建 CVS 卷之一 "服务类型"：

1. * CVS*：基本 CVS 服务类型，可提供较高的区域可用性，但性能级别有限 / 中等。
2. * CVS-Performance*：经过性能优化的服务类型最适合重视性能的生产工作负载。从三个独特的服务级别中进行选择 [`standard`， `premium`，和 `extreme`]。

CVS和CVS-Performance卷的最小大小为100 GiB。

适用于 GCP 的 CVS 驱动程序	快照	克隆	多连接	QoS	展开	Replication
`gcp-cvs`	是的。	是的。	是的。	是的。	是的。	是脚注： 1[]

> 脚注：是脚注： 1[]：不由 Astra Trident 管理

。 `gcp-cvs` 驱动程序使用虚拟存储池。虚拟存储池会对后端进行抽象化，从而使 Astra Trident 决定卷的放置。管理员在 backend.json 文件中定义虚拟存储池。存储类使用标签标识虚拟存储池。

存储类设计

要创建 Kubernetes 存储类对象，需要配置并应用各个存储类。本节讨论如何为您的应用程序设计存储类。

特定后端利用率

可以在特定存储类对象中使用筛选功能来确定要将哪个存储池或一组池与该特定存储类结合使用。可以在存储类中设置三组筛选器： `storagePools`， `additionalStoragePools`和/或 `excludeStoragePools`。

。 `storagePools` 参数有助于将存储限制为与任何指定属性匹配的一组池。。 `additionalStoragePools` 参数用于扩展Astra Trident用于配置的池集以及由属性和选择的池集 `storagePools` parameters您可以单独使用参数，也可以同时使用这两个参数，以确保选择适当的存储池集。

。 `excludeStoragePools` 参数用于明确排除列出的一组与属性匹配的池。

模拟**QoS**策略

如果要设计存储类以模拟服务质量策略、请使用创建存储类 `media` 属性为 `hdd` 或 `ssd`。基于 `media` 属性、Trident将选择提供服务的相应后端 `hdd` 或 `ssd` 聚合以匹配介质属性、然后将卷的配置定向到特定聚合。因此、我们可以创建存储类高级版 `media` 属性设置为 `ssd` 可归类为高级QoS策略。我们可以创建另一个存储类标准，该标准会将介质属性设置为 `HDD'，并可归类为标准 QoS 策略。我们还可以使用存储类中的 ``IOPS' 属性将配置重定向到可定义为 QoS 策略的 Element 设备。

根据特定功能使用后端

存储类可设计为在启用了精简和厚配置，快照，克隆和加密等功能的特定后端直接配置卷。要指定要使用的存储，请创建存储类，以指定启用了所需功能的相应后端。

虚拟存储池

所有 Astra Trident 后端均可使用虚拟存储池。您可以使用 Astra Trident 提供的任何驱动程序为任何后端定义虚拟存储池。

通过虚拟存储池，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 Astra Trident 会选择与所有选择器标签匹配的后端来放置卷。如果存储类选择器标签与多个存储池匹配，则 Astra Trident 将选择其中一个存储池来配置卷。

虚拟存储池设计

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。随着虚拟存储池的推出，此问题描述得以缓解。虚拟存储池是在后端和 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 Astra Trident 为所有受支持的 NetApp 后端定义虚拟存储池。该列表包括 SolidFire/NetApp HCI ， ONTAP ， GCP 上的 Cloud Volumes Service 以及 Azure NetApp Files 。

> 定义虚拟存储池时，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

可以设计虚拟存储池来模拟服务类。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。为 ANF 后端配置多个标签，以表示不同的性能级别。设置 `servicelevel` 添加适当的性能级别、并在每个标签下添加其他所需的方面。现在，创建可映射到不同虚拟存储池的不同 Kubernetes 存储类。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。

分配特定的方面

可以从一个存储后端设计具有一组特定方面的多个虚拟存储池。为此，请为后端配置多个标签，并在每个标签下设置所需的方面。现在、使用创建不同的Kubernetes存储类 `parameters.selector` 要映射到不同虚拟存储池的字段。在后端配置的卷将在选定的虚拟存储池中定义相关方面。

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 Astra Trident 的配置决策过程。

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。

Astra Trident 将尝试与根据下表指定的访问方法所使用的存储协议匹配。这独立于底层存储平台。

	ReadWriteOnce	ReadOnlyMany	读取写入任何
iSCSI	是的。	是的。	是（原始块）
NFS	是的。	是的。	是的。

如果在未配置 NFS 后端的情况下向 Trident 部署提交了 ReadWriteMany PVC 请求，则不会配置任何卷。因此，请求者应使用适合其应用程序的访问模式。

卷操作

修改永久性卷

除了两个例外，永久性卷是 Kubernetes 中不可变的对象。创建后，可以修改回收策略和大小。但是，这并不会阻止在 Kubernetes 外部修改卷的某些方面。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。Astra 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

Astra 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

Astra Trident 还支持从卷快照创建 PersistentVolumes 。为此、只需创建PersistentVolumeClaim并提及即可 `datasource` 作为需要从中创建卷的所需快照。Astra Trident 将通过创建包含快照上的数据的卷来处理此 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。此操作不会影响 Astra Trident 或 Kubernetes 集群，只要目标聚合是 Astra Trident 所使用的 SVM 有权访问的聚合即可。重要的是，如果已将聚合新添加到 SVM ，则需要通过将后端重新添加到 Astra Trident 来刷新后端。这将触发 Astra Trident 对 SVM 重新进行清单配置，以便识别新聚合。

但是， Astra 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 Astra 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 Astra Trident 中。

展开卷

Astra Trident 支持调整 NFS 和 iSCSI PV 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、请设置 `allowVolumeExpansion` to `true` 在与卷关联的StorageClass中。每当需要调整持久性卷的大小时、请编辑 `spec.resources.requests.storage` 在永久性卷声明中为所需的卷大小添加标注。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、支持此功能 `ontap-nas`， `ontap-nas-flexgroup`， `solidfire-san`， `azure-netapp-files`，和 `gcp-cvs` 驱动程序。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 和时 `solidfire-san` 驱动程序、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Astra Trident管理的Kubernetes中。导入卷命令中使用的 PVC YAML 或 JSON 文件指向将 Astra 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` 或 `gcp-cvs` 使用驱动程序时、请使用命令 `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` 将卷导入到要由Astra Trident管理的Kubernetes中。这样可以确保卷引用是唯一的。

执行上述命令后， Astra Trident 将在后端找到卷并读取其大小。它将自动添加（并在必要时覆盖）已配置的 PVC 卷大小。然后， Astra 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

有关为注册表部署和管理存储的文档，请参见 "netapp.io" 在中 "博客"。

日志记录服务

与其他 OpenShift 服务一样，日志记录服务也是使用清单文件（也称为）提供的配置参数 Ansible 部署的主机，提供给攻略手册。其中包括两种安装方法：在初始 OpenShift 安装期间部署日志记录以及在安装 OpenShift 之后部署日志记录。

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 的 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 。

如果选择将NFS与日志记录服务结合使用、则需要设置Ansible变量 `openshift_enable_unsupported_configurations` to `true` 以防止安装程序失败。

入门

可以选择为这两个应用程序以及 OpenShift 集群本身的核心操作部署日志记录服务。如果选择部署操作日志记录、请指定变量 `openshift_logging_use_ops` 作为 `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，必须根据部署方法配置 Ansible 变量。让我们来了解一下每种部署方法的选项。

下表仅包含与存储配置相关的变量，因为这些变量与日志记录服务相关。您可以在中找到其他选项 "RedHat OpenShift 日志记录文档" 应根据您的部署情况查看，配置和使用。

下表中的变量将导致 Ansible 攻略手册使用提供的详细信息为日志记录服务创建 PV 和 PVC 。与在 OpenShift 安装后使用组件安装攻略手册相比，此方法的灵活性明显降低，但是，如果您有可用的现有卷，则可以选择此方法。

变量	详细信息
`openshift_logging_storage_kind`	设置为 `nfs` 让安装程序为日志记录服务创建NFS PV。
`openshift_logging_storage_host`	NFS 主机的主机名或 IP 地址。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、如果卷接合为 `/openshift_logging`、您将使用该路径作为此变量。
`openshift_logging_storage_volume_name`	名称、例如 `pv_ose_logs`、要创建的PV。
`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` 为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。此过程可能会因 OpenShift 的不同版本而略有变化，因此请务必阅读并遵循 "RedHat OpenShift Container Platform 3.11 文档" 适用于您的版本。

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，可以在集群初始设置期间或使用组件安装方法运行之后部署指标服务。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` 让安装程序为日志记录服务创建NFS PV。
`openshift_metrics_storage_host`	NFS 主机的主机名或 IP 地址。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、如果卷接合为 `/openshift_metrics`、您将使用该路径作为此变量。
`openshift_metrics_storage_volume_name`	名称、例如 `pv_ose_metrics`、要创建的PV。
`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 。如果您使用组件攻略手册进行部署，则在 OpenShift 安装之后， Ansible 将创建所需的任何 PVC ，并在 Astra 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。确保您查看并遵循 "RedHat 的 OpenShift 部署指南" 为您的版本配置，以便为您的环境进行配置。

数据保护

了解NetApp存储平台提供的数据保护和可恢复性选项。Astra Trident 可以配置可利用其中某些功能的卷。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。

备份 `etcd` 集群数据

Astra Trident会将其元数据存储在Kubernetes集群中 `etcd` 数据库。定期备份 `etcd` 集群数据对于在发生灾难时恢复Kubernetes集群非常重要。

步骤

1. 。 `etcdctl snapshot save` 命令用于为创建的时间点快照 `etcd` 集群：

```
sudo docker run --rm -v /backup:/backup \
  --network host \
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \
  --env ETCDCTL_API=3 \
  registry.k8s.io/etcd-amd64:3.2.18 \
  etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  snapshot save /backup/etcd-snapshot.db
```

此命令可通过旋转etcd容器来创建etcd快照、并将其保存在中 `/backup` 目录。

2. 发生灾难时，您可以使用 etcd 快照启动 Kubernetes 集群。使用 `etcdctl snapshot restore` 命令以还

原创建到的特定快照 `/var/lib/etcd` 文件夹。还原后、确认是否存在 `/var/lib/etcd` 文件夹中已填充 `member` 文件夹。以下是的示例 `etcdctl snapshot restore` 命令：

```
etcdctl snapshot restore '/backup/etcd-snapshot-latest.db' ; mv
/default.etcd/member/ /var/lib/etcd/
```

3. 在初始化 Kubernetes 集群之前，请复制所有必要的证书。
4. 使用创建集群 `--ignore-preflight-errors=DirAvailable—var-lib-etcd` 标志。
5. 集群启动后，请确保 Kube-system Pod 已启动。
6. 使用 `kubectl get crd` 用于验证Trident创建的自定义资源是否存在的命令、并检索Trident对象以确保所有数据可用。

使用 ONTAP 快照恢复日期

快照通过为应用程序数据提供时间点恢复选项发挥着重要作用。但是，快照本身并不是备份，它们无法防止存储系统故障或其他灾难。但是，在大多数情况下，它们是一种方便，快速和轻松的数据恢复方式。了解如何使用 ONTAP 快照技术为卷创建备份以及如何还原这些备份。

- 如果未在后端定义快照策略、则默认使用 `none` 策略。这会导致 ONTAP 不会自动创建快照。但是，存储管理员可以通过 ONTAP 管理界面手动创建快照或更改快照策略。这不会影响 Trident 操作。
- 默认情况下， Snapshot 目录处于隐藏状态。这有助于最大程度地提高使用配置的卷的兼容性 `ontap-nas` 和 `ontap-nas-economy` 驱动程序。启用 `.snapshot` 目录 `ontap-nas` 和 `ontap-nas-economy` 支持应用程序直接从快照恢复数据的驱动程序。
- 使用将卷还原到先前快照中记录的状态 `volume snapshot restore` ONTAP 命令行界面命令。还原快照副本时，还原操作会覆盖现有卷配置。创建 Snapshot 副本后对卷中数据所做的任何更改都将丢失。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

使用 ONTAP 复制数据

复制数据在防止存储阵列故障导致的数据丢失方面发挥着重要作用。

要了解有关 ONTAP 复制技术的详细信息，请参见 "ONTAP 文档"。

SnapMirror Storage Virtual Machine （ SVM ）复制

您可以使用 "SnapMirror" 复制完整的 SVM ，其中包括其配置设置及其卷。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。

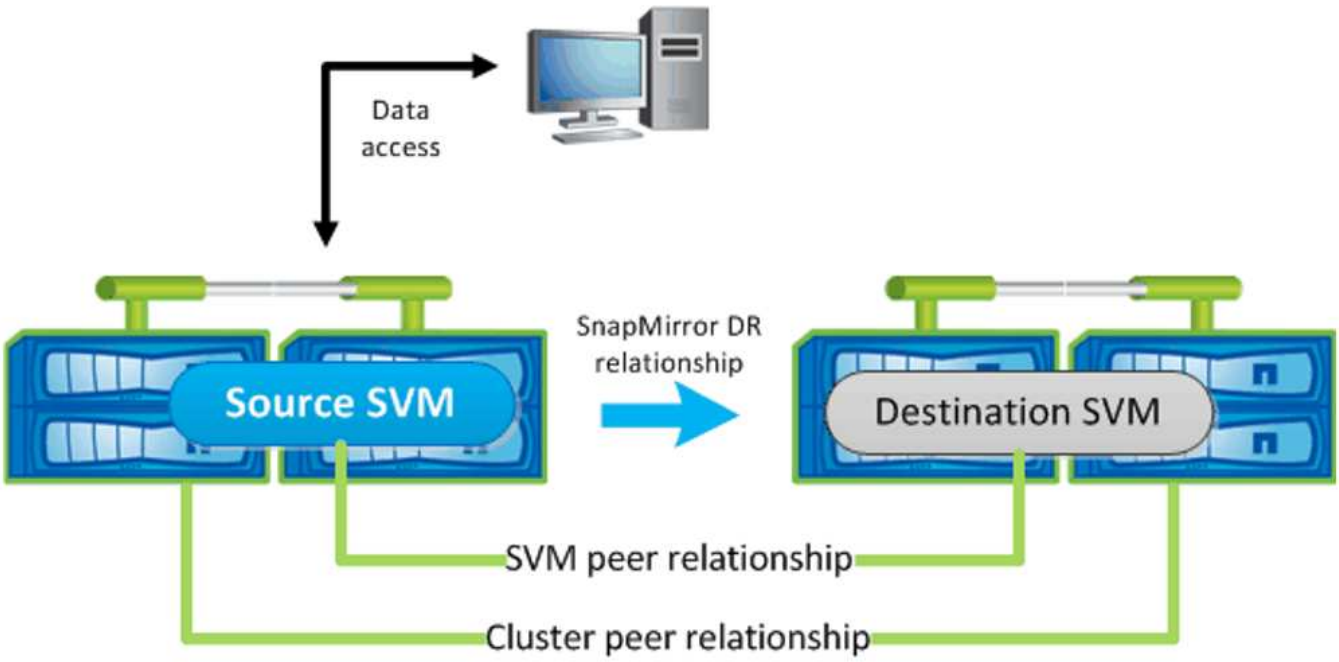
Astra Trident 无法自行配置复制关系，因此存储管理员可以使用 ONTAP 的 SnapMirror SVM 复制功能将卷自动复制到灾难恢复（ Disaster Recovery ， DR ）目标。

如果您计划使用 SnapMirror SVM 复制功能或当前正在使用此功能，请考虑以下事项：

- 您应为每个 SVM 创建一个单独的后端，此后端已启用 SVM-DR 。
- 您应配置存储类，以便在需要时不选择复制的后端。这一点对于避免将不需要复制关系保护的卷配置到支持 SVM-DR 的后端非常重要。
- 应用程序管理员应了解与复制数据相关的额外成本和复杂性，在利用数据复制之前，应确定恢复计划。
- 在激活 SnapMirror 目标 SVM 之前，请停止所有计划的 SnapMirror 传输，中止所有正在进行的 SnapMirror 传输，中断复制关系，停止源 SVM ，然后启动 SnapMirror 目标 SVM 。
- Astra Trident 不会自动检测 SVM 故障。因此、发生故障时、管理员应运行 `tridentctl backend update` 用于触发Trident故障转移到新后端的命令。

下面简要介绍了 SVM 设置步骤：

- 在源和目标集群以及 SVM 之间设置对等关系。
- 使用创建目标SVM `-subtype dp-destination` 选项
- 创建复制作业计划以确保按所需时间间隔进行复制。
- 使用创建从目标SVM到源SVM的SnapMirror复制 `-identity-preserve true` 选项、以确保源SVM配置和源SVM接口复制到目标。从目标 SVM 初始化 SnapMirror SVM 复制关系。



Trident 的灾难恢复工作流

Astra Trident 19.07 及更高版本使用 Kubernetes CRD 存储和管理其自身状态。它使用Kubernetes集群 `etcd` 以存储其元数据。这里我们假设Kubernetes `etcd` 数据文件和证书存储在NetApp FlexVolume上。此 FlexVolume 驻留在 SVM 中， SVM 与二级站点的目标 SVM 具有 SnapMirror SVM-DR 关系。

以下步骤介绍如何在发生灾难时使用 Astra Trident 恢复单个主 Kubernetes 集群：

1. 如果源 SVM 发生故障，请激活 SnapMirror 目标 SVM 。为此，您应停止计划的 SnapMirror 传输，中止正在进行的 SnapMirror 传输，中断复制关系，停止源 SVM 并启动目标 SVM 。

2. 从目标SVM挂载包含Kubernetes的卷 `etcd` 将设置为主节点的主机上的数据文件和证书。
3. 复制下与Kubernetes集群相关的所有必需证书 `/etc/kubernetes/pki` 和etcd `member` 下的文件 `/var/lib/etcd`。
4. 使用创建Kubernetes集群 `kubeadm init` 命令 `--ignore-preflight-errors=DirAvailable—var-lib-etcd` 标志。Kubernetes 节点使用的主机名应与源 Kubernetes 集群相同。
5. 运行 `kubectl get crd` 用于验证是否已启动所有Trident自定义资源的命令、并检索Trident对象以验证所有数据是否可用。
6. 运行以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>` 命令：

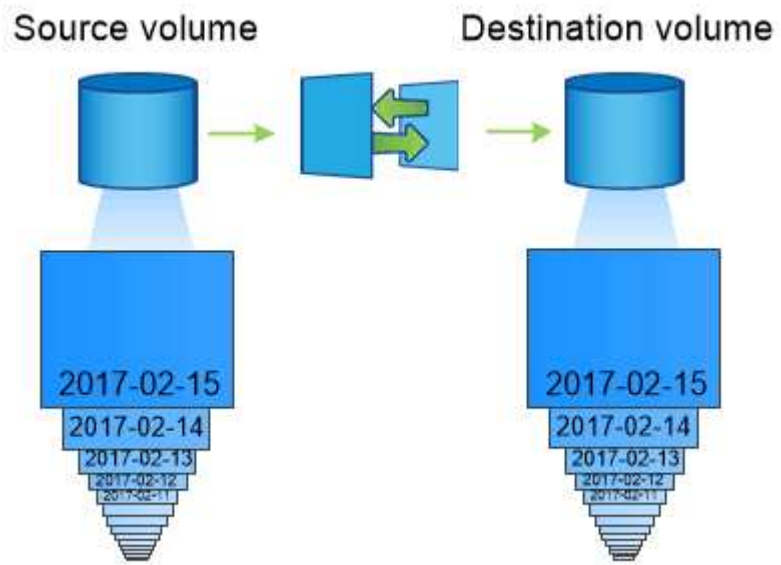
> 对于应用程序永久性卷，激活目标 SVM 后，由 Trident 配置的所有卷将开始提供数据。按照上述步骤在目标端设置 Kubernetes 集群后，所有部署和 Pod 均会启动，容器化应用程序应正常运行。

SnapMirror 卷复制

ONTAP SnapMirror 卷复制是一项灾难恢复功能，可用于在卷级别从主存储故障转移到目标存储。SnapMirror 通过同步快照在二级存储上创建主存储的卷副本或镜像。

下面简要介绍了 ONTAP SnapMirror 卷复制设置步骤：

- 在卷所在的集群与从卷提供数据的 SVM 之间设置对等关系。
- 创建一个 SnapMirror 策略，用于控制关系的行为并指定该关系的配置属性。
- 使用在目标卷和源卷之间创建 SnapMirror 关系[`snapmirror create` 命令^)并分配相应的SnapMirror策略。
- 创建 SnapMirror 关系后，初始化此关系，以便完成从源卷到目标卷的基线传输。



Trident 的 SnapMirror 卷灾难恢复工作流

以下步骤介绍如何使用 Astra Trident 恢复单个主 Kubernetes 集群。

1. 发生灾难时，停止所有计划的 SnapMirror 传输并中止所有正在进行的 SnapMirror 传输。中断目标卷和源卷之间的复制关系，使目标卷变为读 / 写卷。
2. 从目标SVM挂载包含Kubernetes的卷 `etcd` 主机上的数据文件和证书、该主机将设置为主节点。
3. 复制下与Kubernetes集群相关的所有必需证书 `/etc/kubernetes/pki` 和etcd `member` 下的文件 `/var/lib/etcd`。
4. 运行创建Kubernetes集群 `kubeadm init` 命令 `--ignore-preflight-errors=DirAvailable—var-lib-etcd` 标志。主机名应与源 Kubernetes 集群相同。
5. 运行 `kubectl get crd` 用于验证是否已启动所有Trident自定义资源并检索Trident对象的命令、以确保所有数据均可用。
6. 清理先前的后端并在 Trident 上创建新的后端。指定目标 SVM 的新管理和数据 LIF ，新 SVM 名称和密码。

应用程序永久性卷的灾难恢复工作流

以下步骤介绍了在发生灾难时如何为容器化工作负载提供 SnapMirror 目标卷：

1. 停止所有计划的 SnapMirror 传输并中止所有正在进行的 SnapMirror 传输。中断目标卷和源卷之间的复制关系，使目标卷变为读 / 写卷。清理使用与源 SVM 上的卷绑定的 PVC 的部署。
2. 按照上述步骤在目标端设置 Kubernetes 集群后，请从 Kubernetes 集群中清理部署， PVC 和 PV 。
3. 通过指定目标 SVM 的新管理和数据 LIF ，新 SVM 名称和密码，在 Trident 上创建新的后端。
4. 使用 Trident 导入功能将所需卷作为 PV 导入，并绑定到新 PVC 。
5. 使用新创建的 PVC 重新部署应用程序部署。

使用 Element 快照恢复数据

通过为 Element 卷设置快照计划并确保按所需间隔创建快照来备份此卷上的数据。您应使用 Element UI 或 API 设置快照计划。目前、无法通过为卷设置快照计划 `solidfire-san` 驱动程序。

如果发生数据损坏，您可以使用 Element UI 或 API 选择特定快照并手动将卷回滚到快照。此操作将还原自创建快照以来对卷所做的任何更改。

安全性

使用此处列出的建议确保Astra Trident安装安全。

在自己的命名空间中运行 Astra 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 Astra Trident 对象定义或 Pod ，以确保存储可靠并阻止潜在的恶意活动。

要将其他应用程序和用户与Astra Trident分开、请始终在自己的Kubernetes命名空间中安装Astra Trident (`trident`）。将 Astra Trident 置于自己的命名空间中可确保只有 Kubernetes 管理人员才能访问 Astra Trident Pod 以及存储在命名空间 CRD 对象中的项目（如适用，还包括后端和 CHAP 密码）。您应确保仅允许管理员

访问Astra 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

Astra 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 建议将双向 CHAP 与 Astra Trident 结合使用，以便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、Astra Trident可以通过设置双向CHAP并管理CHAP用户名和密码 `tridentctl`。请参见 "此处" 了解 Astra Trident 如何在 ONTAP 后端配置 CHAP 。

Trident 20.04 及更高版本支持 ONTAP 后端的 CHAP 。

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。Astra Trident 使用一个机密对象，每个租户包含两个 CHAP 密码。当Trident作为CSI配置程序安装时、它会管理CHAP密码并将其存储在中 `tridentvolume` 相应PV的CR对象。创建 PV 时， CSI Astra 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。

CSI Trident 创建的卷不与任何卷访问组关联。

在非 CSI 前端中，作为辅助节点上的设备连接卷的操作由 Kubernetes 处理。创建卷后，如果该租户的密钥尚不存在，则 Astra Trident 会对 NetApp HCI/SolidFire 系统进行 API 调用，以检索这些密钥。然后， Astra Trident 将这些机密传递给 Kubernetes 。位于每个节点上的 kubelet 通过 Kubernetes API 访问这些机密，并使用它们在访问卷的每个节点与卷所在的 NetApp HCI/SolidFire 系统之间运行 / 启用 CHAP 。

将Astra 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Astra Trident中配置的任何卷都将启用NAE。
- 如果后端未启用NAE、则在Astra Trident中配置的任何卷都将启用NVE、除非将NVE加密标志设置为 `false` 在后端配置中。

在启用了NAE的后端的Astra Trident中创建的卷必须经过NVE或NAE加密。

- 您可以将NVE加密标志设置为 `true` 在Trident后端配置中、覆盖NAE加密并按卷使用特定的加密密钥。
- 将NVE加密标志设置为 `false` 在启用了NAE的后端、将创建启用了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Astra 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"

使用Linux统一密钥设置(Unified Key Setup、LUKS)启用每个卷的主机端加密

您可以启用Linux统一密钥设置(LUKS)来对Astra Trident上的ONTAP SAN和ONTAP SAN经济卷进行加密。在Astra Trident中、LUKS加密的卷会按照建议使用AES-XTS-plain64 Cypher和模式 "NIST"。

有关ONTAP SAN的后端配置选项的详细信息、请参见 "ONTAP SAN配置选项"

开始之前

- 工作节点必须安装Cryptsetup 2.1或更高版本。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)"。

步骤

1. 在后端配置中定义LUKS加密属性。

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

2. 使用 ... `parameters.selector` 使用LUKS加密定义存储池。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
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. 创建包含LUKS密码短语的密码短语。例如：

```
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
```

限制

- LUKS加密卷将无法利用ONTAP 重复数据删除和数据压缩功能。
- 目前不支持LUKS密码短语轮换。要更改密码短语、请手动将数据从一个PVC复制到另一个PVC。

参考

Astra Trident 端口

详细了解 Astra Trident 所通过的端口。

Astra Trident 通过以下端口进行通信：

Port	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

在安装期间、可以使用更改活动/就绪探测端口 `--probe-port` 标志。请务必确保此端口未被工作节点上的其他进程使用。

Astra Trident REST API

同时 "tridentctl 命令和选项" 是与Astra Trident REST API交互的最简单方式、您可以根据需要直接使用REST端点。

何时使用REST API

REST API适用于在非Kubernetes部署中使用Astra Trident作为独立二进制文件的高级安装。

为了提高安全性、我们使用了Astra Trident `REST API` 默认情况下、在Pod内部运行时、仅限于localhost。要更改此行为、您需要设置Astra Trident `-address` 参数。

使用REST API

API 的工作原理如下：

`GET`

- `GET <trident-address>/trident/v1/<object-type>`：列出此类型的所有对象。
- `GET <trident-address>/trident/v1/<object-type>/<object-name>`：获取已命名对象的详细信息。

`POST`

`POST <trident-address>/trident/v1/<object-type>`：创建指定类型的对象。

- 需要为要创建的对象配置 JSON 。有关每个对象类型的规范、请参见链接：tridentctl.html[`tridentctl` 命令和选项]。

- 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。

`DELETE`

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`：删除已命名的资源。

> 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、请参见链接：tridentctl.html[`tridentctl` 命令和选项]。

有关如何调用这些API的示例、请通过调试 (`-d`)标志。有关详细信息、请参见链接：tridentctl.html[`tridentctl` 命令和选项]。

命令行选项

Astra 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(debug、info、warn、error、fal)。默认为 INFO 。

Kubernetes

- `-k8s_pod`：使用此选项或 `-k8s_api_server` 以启用Kubernetes支持。如果设置此值，则 Trident 将使用其所属 POD 的 Kubernetes 服务帐户凭据来联系 API 服务器。只有当 Trident 在启用了服务帐户的 Kubernetes 集群中作为 POD 运行时，此功能才有效。
- `-k8s_api_server <insecure-address:insecure-port>`：使用此选项或 `-k8s_pod` 以启用Kubernetes支持。指定后， Trident 将使用提供的不安全地址和端口连接到 Kubernetes API 服务器。这样可以在 Pod 之外部署 Trident ；但是，它仅支持与 API 服务器的不安全连接。要安全连接、请使用在Pod中部署Trident `-k8s_pod` 选项
- `-k8s_config_path <file>`：必需；必须指定KubeConfig文件的路径。

Docker

- `-volume_driver <name>`：注册Docker插件时使用的驱动程序名称。默认为 `netapp`。
- `-driver_port <port-number>`：侦听此端口、而不是UNIX域套接字。
- `-config <file>`：必需；必须指定后端配置文件的路径。

REST

- `-address <ip-or-host>`：指定Trident的REST服务器应侦听的地址。默认为 localhost 。在本地主机上侦听并在 Kubernetes Pod 中运行时，无法从 Pod 外部直接访问 REST 接口。使用 ... `-address ""` 可从Pod IP地址访问REST接口。

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Trident的REST服务器应侦听的端口。默认为 8000 。
- `-rest`：启用REST接口。默认为 true 。

与 Kubernetes 集成的 NetApp 产品

NetApp 存储产品组合可与 Kubernetes 集群的许多不同方面相集成，从而提供高级数据管理功能，从而增强 Kubernetes 部署的功能，功能，性能和可用性。

Astra

"Astra" 使企业能够更轻松地管理，保护和移动在公有云内部和内部环境中以及之间的 Kubernetes 上运行的数据丰富的容器化工作负载。Astra 使用 NetApp 在公有云和内部环境中成熟而广泛的存储产品组合中的 Trident 配置和提供永久性容器存储。此外，它还提供了一组丰富的高级应用程序感知型数据管理功能，例如快照，备份和还原，活动日志和主动克隆，用于数据保护，灾难 / 数据恢复，数据审核以及 Kubernetes 工作负载的迁移用例。

ONTAP

ONTAP 是 NetApp 的多协议统一存储操作系统，可为任何应用程序提供高级数据管理功能。ONTAP 系统采用全闪存，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 支持上述所有 ONTAP 部署模式。

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。您可以将 Cloud Volumes ONTAP 用于生产工作负载、灾难恢复、 DevOps 、文件共享和数据库管理。它通过提供存储效率，高可用性，数据复制，数据分层和应用程序一致性，将企业级存储扩展到云。

适用于 NetApp ONTAP 的 Amazon FSX

"适用于 NetApp ONTAP 的 Amazon FSX" 是一种完全托管的 AWS 服务，可使客户启动和运行由 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 ，客户可以利用他们熟悉的 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 的许多文件系统功能和管理 API 。

Element 软件

"Element" 通过保证性能并简化存储占用空间，使存储管理员能够整合工作负载。Element 与 API 相结合，可实现存储管理各个方面的自动化，可帮助存储管理员事半功倍。

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 完全支持 NetApp HCI 。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。

Azure NetApp Files

"Azure NetApp Files" 是一种企业级 Azure 文件共享服务，由 NetApp 提供支持。您可以在 Azure 中以本机方式运行要求最苛刻的基于文件的工作负载，同时享受 NetApp 应有的性能和丰富的数据管理功能。

适用于 Google Cloud 的 Cloud Volumes Service

"适用于 Google Cloud 的 NetApp Cloud Volumes Service" 是一种云原生文件服务，可通过 NFS 和 SMB 提供具有全闪存性能的 NAS 卷。此服务支持在 GCP 云中运行任何工作负载，包括原有应用程序。它提供了一种完全托管的服务，可提供稳定一致的高性能，即时克隆，数据保护以及对 Google Compute Engine （ GCE ）实例的安全访问。

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 `PersistentVolumeClaim` 请求新的 `PersistentVolume` 的大小 `StorageClass` 之前由管理员配置的。
2. Kubernetes `StorageClass` 将Trident标识为其配置程序、并包含一些参数、用于指示Trident如何为请求的类配置卷。
3. Trident独立查看 `StorageClass` 名称相同、用于标识匹配项 `Backends` 和 `StoragePools` 可用于为类配置卷。
4. Trident在匹配的后端配置存储并创建两个对象：A `PersistentVolume` 在Kubernetes中、此命令告诉Kubernetes如何查找、挂载和处理卷、以及在Trident中保留两个卷之间关系的卷 `PersistentVolume` 和实际存储。
5. Kubernetes绑定 `PersistentVolumeClaim` 到新的 `PersistentVolume`。包含的Pod `PersistentVolumeClaim` 将此PersistentVolume挂载到其运行所在的任何主机上。
6. 用户创建 `VolumeSnapshot` 现有PVC、使用 `VolumeSnapshotClass` 这就是Trident。
7. Trident 标识与 PVC 关联的卷，并在其后端创建卷的快照。它还会创建 `VolumeSnapshotContent` 这将指示Kubernetes如何识别快照。
8. 用户可以创建 `PersistentVolumeClaim` 使用 `VolumeSnapshot` 作为源。
9. Trident可确定所需的快照、并执行与创建相同的一组步骤 `PersistentVolume` 和 A `Volume`。

要进一步了解 Kubernetes 对象，强烈建议您阅读 "永久性卷" Kubernetes 文档的一节。

Kubernetes `PersistentVolumeClaim` 对象

一个Kubernetes `PersistentVolumeClaim` 对象是Kubernetes集群用户发出的存储请求。

除了标准规范之外，如果用户要覆盖在后端配置中设置的默认值， Trident 还允许用户指定以下特定于卷的标注：

标注	卷选项	支持的驱动程序
trident.netapp.io/fileSystem	文件系统	ontap-san、solidfire-san、ontap-san-economy.
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas ， ontap-san ， solidfire-san ， azure-netapp-files ， 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` reclaiming policy、Trident会在PV释放时(即用户删除PVC时)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。PV使用时 `Retain` 策略中、Trident会忽略该策略、并假定管理员将从Kubernetes和后端清理该卷、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 。请查看 `On-Demand Volume Snapshots` 以了解其工作原理。

Trident还提供 `cloneFromPVC` 和 `splitOnClone` 用于创建克隆的标注。您可以使用这些标注克隆 PVC ，而无需使用 CSI 实施（在 Kubernetes 1.13 及更早版本上），或者您的 Kubernetes 版本不支持测试版卷快照（ Kubernetes 1.16 及更早版本）。请注意， Trident 19.10 支持从 PVC 克隆的 CSI 工作流。

您可以使用 `cloneFromPVC` 和 `splitOnClone` 使用CSI Trident以及传统非CSI前端的标注。

以下是一个示例：如果用户已有一个名为的PVC `mysql`、用户可以创建一个名为的新PVC `mysqlclone` 使用标注、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。
- 使用 `ontap-nas` 和 `ontap-san` 驱动程序、可能需要设置PVC标注 `trident.netapp.io/splitOnClone` 与结合使用 `trident.netapp.io/cloneFromPVC`。使用 `trident.netapp.io/splitOnClone` 设置为 `true`、Trident会将克隆的卷与父卷拆分、从而将克隆的卷与其父卷的生命周期完全分离、从而降低了存储效率。未设置 `trident.netapp.io/splitOnClone` 或将其设置为 `false` 这样可以减少后端的空间消耗、而不会在父卷和克隆卷之间创建依赖关系、因此除非先删除克隆、否则无法删除父卷。拆分克隆是有意义的一种情形，即克隆空数据库卷时，该卷及其克隆会发生很大的差异，无法从 ONTAP 提供的存储效率中受益。

。 `sample-input` 目录包含用于Trident的PVC定义示例。有关与 Trident 卷关联的参数和设置的完整问题描述，请参见 Trident 卷对象。

Kubernetes `PersistentVolume` 对象

一个Kubernetes `PersistentVolume` 对象表示可供Kubernetes集群使用的一段存储。它的生命周期与使用它的 POD 无关。

> Trident创建 `PersistentVolume` 根据Kubernetes集群所配置的卷、自动将这些对象注册到Kubernetes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一个新卷、并为该卷注册一个新的PV。在配置已配置的卷和相应的 PV 时， Trident 会遵循以下规则：

- Trident 会为 Kubernetes 生成 PV 名称及其用于配置存储的内部名称。在这两种情况下，它都可以确保名称在其范围内是唯一的。
- 卷的大小与 PVC 中请求的大小尽可能匹配，但可能会根据平台将其取整为最接近的可分配数量。

Kubernetes `StorageClass` 对象

Kubernetes `StorageClass` 对象在中按名称指定 `PersistentVolumeClaims` 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。

一个Kubernetes `StorageClass` 使用Trident的对象如下所示：

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

这些参数是 Trident 专用的，可告诉 Trident 如何为类配置卷。

存储类参数包括：

属性	Type	Required	Description
属性	map[string]string	否	请参见下面的属性部分
存储池	map[string]StringList	否	后端名称映射到中的存储池列表
附加 StoragePools	map[string]StringList	否	后端名称映射到中的存储池列表
排除 StoragePools	map[string]StringList	否	后端名称映射到中的存储池列表

存储属性及其可能值可以分类为存储池选择属性和 Kubernetes 属性。

存储池选择属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	Type	值	优惠	请求	支持
介质[1]	string	HDD ，混合，SSD	Pool 包含此类型的介质；混合表示两者	指定的介质类型	ontap-nas ，ontap-nas-economy. ontap-nas-flexgroup ，ontap-san ，solidfire-san
配置类型	string	精简，厚	Pool 支持此配置方法	指定的配置方法	Thick：All ONTAP ；Thin ：All ONTAP & solidfire-san

属性	Type	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、您可以使用 `attributes` 您需要单独为满足特定类需求所需的存储质量建模。Trident会自动发现并选择与的_all_匹配的存储池 `attributes` 您指定的。

如果您发现自己无法使用 `attributes` 要自动为某个类选择合适的池、您可以使用 `storagePools` 和 `additionalStoragePools` 用于进一步细化池甚至选择一组特定池的参数。

您可以使用 `storagePools` 参数以进一步限制与指定的任何池匹配的池集 `attributes`。换言之、Trident使用由标识的池的交叉点 `attributes` 和 `storagePools` 用于配置的参数。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` 参数以扩展Trident用于配置的一组池、而不管选择的任何池如何 `attributes` 和 `storagePools` parameters

您可以使用 `excludeStoragePools` 用于筛选Trident用于配置的一组池的参数。使用此参数将删除任何匹配的池。

在中 `storagePools` 和 `additionalStoragePools` 参数、每个条目采用的形式 `<backend>:<storagePoolList>`、其中 `<storagePoolList>` 是指定后端的存储池列表、以英文逗号分隔。例如、的值 `additionalStoragePools` 可能如下所示 `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` 以获取后端及其池的列表。

Kubernetes 属性

这些属性不会影响 Trident 在动态配置期间选择的存储池 / 后端。相反，这些属性仅提供 Kubernetes 永久性卷支持的参数。工作节点负责文件系统创建操作，并且可能需要文件系统实用程序，例如 xfsprogs 。

属性	Type	值	Description	相关驱动程序	Kubernetes 版本
FSType	string	ext4 ， ext3 ， xfs 等	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 及更高版本
卷绑定模式	string	即时， WaitForFirstConsumer"	选择何时进行卷绑定和动态配置	全部	1.19 - 1.25

- 。 `fsType` 参数用于控制SAN LUN所需的文件系统类型。此外、Kubernetes还会使用 `fsType` 在存储类中以指示文件系统已存在。可以使用控制卷所有权 `fsGroup` 仅当出现此情况时、Pod的安全上下文才会显示 `fsType` 已设置。请参见 "Kubernetes ： 为 Pod 或容器配置安全上下文" 有关使用设置卷所有权的概述 `fsGroup` 环境。Kubernetes将应用 `fsGroup` 只有在以下情况下才为值：
  - `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。以便使用 `fsGroup` 存储类仍需要指定 `fsType`。您可以将其设置为 `nfs` 或任何非空值。
- 请参见 "展开卷" 有关卷扩展的更多详细信息。
- Trident安装程序包提供了几个示例存储类定义、用于中的Trident `sample-input/storage-class-*.yaml`。删除 Kubernetes 存储类也会删除相应的 Trident 存储类。

Kubernetes `VolumeSnapshotClass` 对象

Kubernetes `VolumeSnapshotClass` 对象类似于 `StorageClasses`。它们有助于定义多个存储类，并由卷快照引用以将快照与所需的快照类关联。每个卷快照都与一个卷快照类相关联。

答 `VolumeSnapshotClass` 要创建快照、应由管理员定义。此时将使用以下定义创建卷快照类：

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 `driver` 指定请求的卷快照的Kubernetes `csi-snapclass` 类由Trident处理。。 `deletionPolicy` 指定必须删除快照时要执行的操作。时间 `deletionPolicy` 设置为 `Delete`、卷快照对象以及存储集群上的底层快照会在删除快照时被删除。或者、也可以将其设置为 `Retain` 这意味着 `VolumeSnapshotContent` 并保留物理快照。

Kubernetes `VolumeSnapshot` 对象

一个Kubernetes `VolumeSnapshot` 对象是创建卷快照的请求。就像 PVC 代表用户对卷发出的请求一样，卷快照也是用户为现有 PVC 创建快照的请求。

收到卷快照请求后、Trident会自动管理在后端为卷创建快照的操作、并通过创建唯一快照来公开快照 `VolumeSnapshotContent` 对象。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。

Kubernetes VolumeSnapshotContent 对象

一个Kubernetes VolumeSnapshotContent 对象表示从已配置的卷创建的快照。它类似于 PersistentVolume 和表示存储集群上配置的快照。类似于 PersistentVolumeClaim 和 PersistentVolume 对象、创建快照时、 VolumeSnapshotContent 对象保持与的一对一映射 VolumeSnapshot 对象、该对象已请求创建快照。

> Trident创建 VolumeSnapshotContent 根据Kubernetes集群所配置的卷、自动将这些对象注册到Kubernetes集群中。您不应自行管理它们。

。 VolumeSnapshotContent 对象包含用于唯一标识快照的详细信息、例如 snapshotHandle。这 snapshotHandle 是PV名称和名称的唯一组合 VolumeSnapshotContent 对象。

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 VolumeSnapshotContent 对象、从而将快照公开到Kubernetes API。

Kubernetes CustomResourceDefinition 对象

Kubernetes 自定义资源是 Kubernetes API 中的端点，由管理员定义并用于对类似对象进行分组。Kubernetes 支持创建自定义资源以存储对象集合。您可以通过运行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。

从19.07版开始、Trident会使用多个 CustomResourceDefinition 用于保留Trident对象身份的对象、例如Trident后端、Trident存储类和Trident卷。这些对象由 Trident 管理。此外， CSI 卷快照框架还引入了一些定义卷快照所需的 CRD 。

CRD 是一种 Kubernetes 构造。上述资源的对象由 Trident 创建。例如、使用创建后端时 tridentctl、对应的 tridentbackends 创建CRD对象供Kubernetes使用。

有关 Trident 的 CRD ，请注意以下几点：

- 安装 Trident 时，系统会创建一组 CRD ，并可像使用任何其他资源类型一样使用。
- 从先前版本的Trident (使用的版本)升级时 etcd 为了保持状态)、Trident安装程序会从迁移数据 etcd 键值数据存储并创建相应的CRD对象。
- 使用卸载Trident时 tridentctl uninstall 命令中、Trident Pod会被删除、但创建的CRD不会被清理。请参见 "卸载 Trident" 了解如何从头开始完全删除和重新配置 Trident 。

Trident StorageClass 对象

Trident会为Kubernetes创建匹配的存储类 StorageClass 指定的对象 csi.trident.netapp.io /netapp.io/trident 在其配置程序字段中。存储类名称与Kubernetes的名称匹配 StorageClass 它所代表的对象。

> 使用Kubernetes时、这些对象会在Kubernetes时自动创建 StorageClass 使用Trident作为配置程序进行注册。

存储类包含一组卷要求。Trident 会将这些要求与每个存储池中的属性进行匹配；如果匹配，则该存储池是使用

该存储类配置卷的有效目标。

您可以使用 REST API 创建存储类配置以直接定义存储类。但是、对于Kubernetes部署、我们希望在注册新Kubernetes时创建这些部署 `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Kubernetes `StorageClass` 对象。

有关如何构建这些对象的详细信息、请参见 "正在配置后端"。

Trident `StoragePool` 对象

存储池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Kubernetes中、这些关系直接对应于 `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	Type	Required	Description
version	string	否	Trident API 版本（ "1" ）
name	string	是的。	要创建的卷的名称
存储类	string	是的。	配置卷时要使用的存储类
size	string	是的。	要配置的卷大小（以字节为单位）
协议	string	否	要使用的协议类型； "file" 或 "block"
内部名称	string	否	存储系统上的对象名称；由 Trident 生成
cloneSourceVolume	string	否	ONTAP （ NAS ， SAN ）和 SolidFire — * ： 要从中克隆的卷的名称

属性	Type	Required	Description
splitOnClone	string	否	ONTAP （ NAS ， SAN ）： 将克隆从其父级拆分
snapshotPolicy	string	否	Snapshot-*： 要使用的 ONTAP 策略
SnapshotReserve	string	否	Snapshot-*： 为快照预留的卷百分比 ONTAP
导出策略	string	否	ontap-nas*： 要使用的导出策略
snapshotDirectory	池	否	ontap-nas*： 是否显示快照目录
unixPermissions	string	否	ontap-nas*： 初始 UNIX 权限
块大小	string	否	SolidFire — *： 块 / 扇区大小
文件系统	string	否	文件系统类型

生成Trident `internalName` 创建卷时。这包括两个步骤。首先、它会预先添加存储前缀(默认值 `trident` 或后端配置中的前缀)添加到卷名称、从而生成表单的名称 `<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。

您可以使用卷配置使用REST API直接配置卷、但在Kubernetes部署中、我们希望大多数用户都使用标准Kubernetes `PersistentVolumeClaim` 方法Trident 会在配置过程中自动创建此卷对象。

Trident `Snapshot` 对象

快照是卷的时间点副本，可用于配置新卷或还原状态。在Kubernetes中、这些关系直接对应于 `VolumeSnapshotContent` 对象。每个快照都与一个卷相关联，该卷是快照的数据源。

每个 `Snapshot` 对象包括以下属性：

属性	Type	Required	Description
version	string	是的。	Trident API 版本（ "1" ）
name	string	是的。	Trident Snapshot 对象的名称
内部名称	string	是的。	存储系统上 Trident Snapshot 对象的名称
volumeName	string	是的。	为其创建快照的永久性卷的名称
volumeInternalName	string	是的。	存储系统上关联的 Trident 卷对象的名称

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。

当Kubernetes时 `VolumeSnapshot` 对象请求已创建、Trident可通过在备用存储系统上创建快照对象来工作。。 `internalName` 的快照对象是通过合并前缀来生成的 `snapshot-` 使用 `UID` 的 `VolumeSnapshot` 对象(例如、 `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。 `volumeName` 和 `volumeInternalName` 通过获取后备卷的详细信息来填充。

Astra Trident `ResourceQuota` 对象

Trident的降级使用 `system-node-critical` 优先级类—Kubernetes中可用的最高优先级类—用于确保Astra Trident能够在正常节点关闭期间识别和清理卷、并允许Trident demonset 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、Astra Trident采用了 `ResourceQuota` 用于确保满足Trident子集上的"系统节点关键"优先级类的对象。在部署和创建emonset之前、Astra Trident会查找 `ResourceQuota` 对象、如果未发现、则应用此对象。

如果您需要对默认资源配额和优先级类进行更多控制、可以生成 `custom.yaml` 或配置 `ResourceQuota` 使用Helm图表的对象。

以下是一个`ResourceQuota`对象的示例、该对象会优先处理Trident子集。

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
     matchExpressions:
       - operator : In
         scopeName: PriorityClass
         values: ["system-node-critical"]
```

有关资源配额的详细信息、请参见 "Kubernetes：资源配额"。

清理 `ResourceQuota` 如果安装失败

在极少数情况下、安装会在之后失败 `ResourceQuota` 对象已创建、请先尝试 "正在卸载" 然后重新安装。

如果不起作用、请手动删除 `ResourceQuota` 对象。

删除 `ResourceQuota`

如果您希望控制自己的资源分配、则可以删除Astra Trident `ResourceQuota` 使用命令的对象：

```
kubectl delete quota trident-csi -n trident
```

tridentctl 命令和选项

。 "Trident 安装程序包" 包括命令行实用程序、 `tridentctl`、可轻松访问Astra Trident。具有足够权限的 Kubernetes 用户可以使用它来安装 Astra Trident ，并直接与其交互以管理包含 Astra Trident Pod 的命名空间。

可用的命令和选项

有关使用情况信息、请运行 `tridentctl --help`。

可用的命令和全局选项包括：

```
Usage:
  tridentctl [command]
```

可用命令：

- `create`：将资源添加到Astra Trident。
- `delete`：从Astra Trident中删除一个或多个资源。
- `get`：从Astra Trident获取一个或多个资源。
- `help`：有关任何命令的帮助。
- `images`：打印包含Astra Trident所需容器映像的表。
- `import`：将现有资源导入到Astra Trident。
- `install`：安装Astra Trident。
- `logs`：从Astra Trident打印日志。
- `send`：从Astra Trident发送资源。
- `uninstall`：卸载Astra Trident。
- `update`：修改Astra Trident中的资源。
- `upgrade`：升级Astra Trident中的资源。
- `version`：打印Astra Trident的版本。

flags

- `-d, --debug`：调试输出。
- `-h, --help`：帮助 `tridentctl`。
- `-n, --namespace string`：Astra 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`：Astra 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。

> 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。

create

您可以使用运行 `create` 用于向Astra Trident添加资源的命令。

```
Usage:
  tridentctl create [option]
```

可用选项：
`backend`：将后端添加到Astra Trident。

delete

您可以运行 `delete` 用于从Astra Trident中删除一个或多个资源的命令。

```
Usage:
  tridentctl delete [option]
```

可用选项：

- `backend`：从Astra Trident中删除一个或多个存储后端。
- `snapshot`：从Astra Trident中删除一个或多个卷快照。
- `storageclass`：从Astra Trident中删除一个或多个存储类。
- `volume`：从Astra Trident中删除一个或多个存储卷。

get

您可以运行 `get` 用于从Astra Trident获取一个或多个资源的命令。

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

可用选项：

- `backend`：从Astra Trident获取一个或多个存储后端。
- `snapshot`：从Astra Trident获取一个或多个快照。
- `storageclass`：从Astra Trident获取一个或多个存储类。
- `volume`：从Astra Trident获取一个或多个卷。

`volume` 标志：* `-h, --help`：卷帮助。* `--parentOfSubordinate string`：将查询限制为从源卷。* `--subordinateOf string`：将查询限制为卷的下属。

images

您可以运行 `images` 用于打印Astra Trident所需容器映像表的标志。

```
Usage:
  tridentctl images [flags]
```

标志：* `-h, --help`: Help for images.
* -v、-k8s-version字符串`：Kubernetes集群的语义版本。

import volume

您可以运行 `import volume` 用于将现有卷导入到Astra Trident的命令。

```
Usage:
  tridentctl import volume <backendName> <volumeName> [flags]
```

别名：
`volume, v`

flags

- `-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。

install

您可以运行 `install` 用于安装Astra Trident的标志。

```
Usage:
  tridentctl install [flags]
```

flags

- `--autosupport-image string`：AutoSupport 遥测的容器映像(默认为"netapp/trident autosupport ：20.07.0")。
- `--autosupport-proxy string`：用于发送AutoSupport 遥测的代理的地址/端口。
- `--csi`：安装CSI Trident (仅适用于Kubernetes 1.13、需要功能门)。
- `--enable-node-prep`：尝试在节点上安装所需的软件包。
- `--generate-custom-yaml`：在不安装任何内容的情况下生成YAML文件。
- `-h, --help`：安装帮助。
- `--http-request-timeout`：覆盖Trident控制器的REST API的HTTP请求超时(默认值为1m30s)。
- `--image-registry string`：内部映像注册表的地址/端口。
- `--k8s-timeout duration`：所有Kubernetes操作的超时(默认值为3m0)。
- `--kubelet-dir string`：kubelet内部状态的主机位置(默认值为"/var/lib/kubelet")。
- `--log-format string`：Astra Trident日志记录格式(文本、json)(默认为"text")。
- `--pv string`：Astra Trident使用的原有PV的名称可确保此名称不存在(默认为"trident ")。
- `--pvc string`：Astra Trident使用的原有PVC的名称可确保此名称不存在(默认为"trident ")。
- `--silence-autosupport`：不要自动向NetApp发送AutoSupport 捆绑包(默认为true)。
- `--silent`：在安装期间禁用大多数输出。
- `--trident-image string`：要安装的Astra Trident映像。
- `--use-custom-yaml`：使用设置目录中的任何现有YAML文件。
- `--use-ipv6`：使用IPv6进行Astra Trident的通信。

logs

您可以运行 `logs` 用于从Astra Trident打印日志的标志。

```
Usage:
  tridentctl logs [flags]
```

flags

- `-a, --archive`：除非另有说明、否则使用所有日志创建支持归档。
- `-h, --help`：日志帮助。
- `-l, --log string`：要显示的Astra Trident日志。Trident 中的一个 "auto"|trident 操作符 "All" （默认为 "auto" ）。
- `--node string`：要从中收集节点Pod日志的Kubernetes节点名称。
- `-p, --previous`：获取先前容器实例的日志(如果存在)。
- `--sidecars`：获取sidecar容器的日志。

send

您可以运行 `send` 用于从Astra Trident发送资源的命令。

```
Usage:
  tridentctl send [option]
```

可用选项：
`autosupport`：将AutoSupport 归档发送给NetApp。

uninstall

您可以运行 `uninstall` 用于卸载Astra Trident的标志。

```
Usage:
  tridentctl uninstall [flags]
```

标志：* `-h, --help`：卸载帮助。* `--silent`：卸载期间禁用大多数输出。

update

您可以运行 `update` 用于在Astra Trident中修改资源的命令。

```
Usage:
  tridentctl update [option]
```

可用选项：
`backend`：在Astra Trident中更新后端。

upgrade

您可以运行 `upgrade` 用于在Astra Trident中升级资源的命令。

```
Usage:
tridentctl upgrade [option]
```

可用选项：
`volume`：将一个或多个永久性卷从NFS/iSCSI升级到CSI。

version

您可以运行 `version` 用于打印版本的标志 `tridentctl` 以及正在运行的Trident服务。

```
Usage:
  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。

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。为了提供此自定义设置、Astra Trident会在安装期间启用某些权限。以下各节详细介绍了Astra 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，请参见 "Kubernetes：安全性"。

所需的Kubernetes安全上下文和相关字段

权限	Description
特权	CSI要求挂载点为双向挂载点、这意味着Trident节点POD必须运行特权容器。有关详细信息，请参见 "Kubernetes：挂载传播"。
主机网络连接	iSCSI守护进程所需的。 `iscsiadm` 管理iSCSI挂载并使用主机网络连接与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	需要启动 `rpc-statd` 对于NFS。Astra Trident会查询主机进程以确定是否 `rpc-statd` 正在挂载NFS卷之前运行。

功能	。 `SYS_ADMIN` 此功能是特权容器的默认功能的一部分。例如、Docker为有权限的容器设置了以下功能：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccomp配置文件在特权容器中始终处于"非受限"状态；因此、无法在Astra Trident中启用它。
SELinux	在OpenShift上、特权容器在中运行 `spc_t` ("超级特权容器")域和无特权容器在中运行 `container_t` 域。开启 `containerd`、使用 `container-selinux` 安装后、所有容器都会在中运行 `spc_t` 域、这会有效禁用SELinux。因此、Astra Trident不会添加 `seLinuxOptions` 到容器。
DAC	有权限的容器必须以root用户身份运行。非特权容器以root用户身份运行、以访问CSI所需的UNIX套接字。

POD安全标准(PSS)

Label	Description	Default

`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 `privileged` 已更改。如果是、请重新安装Trident。

POD安全策略(PSP)

字段	Description	Default
`allowPrivilegeEscalation`	有权限的容器必须允许权限升级。	`true`
`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"、因此它们不会包含在允许的卷列表中。	空
`allowedHostPaths`	Trident节点POD挂载节点的根文件系统、因此设置此列表没有好处。	空
`allowedProcMountTypes`	Trident不使用任何 `ProcMountTypes`。	空
`allowedUnsageSysctls`	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`	否 `sysctls` 允许。	空
`fsGroup`	Trident容器以root身份运行。	`RunAsAny`
`hostIPC`	挂载NFS卷需要主机IPC才能与进行通信 `nfsd`	`true`
`hostNetwork`	iscsiadm要求主机网络与iSCSI守护进程进行通信。	`true`
`hostPID`	需要使用主机PID检查是否存在 `rpc-statd` 正在节点上运行。	`true`

字段	Description	Default
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 因为容器运行时间和Kubernetes分发程序处理SELinux的方式目前存在差异。	空
supplementalGroups	Trident容器以root身份运行。	RunAsAny
volumes	Trident Pod需要这些卷插件。	hostPath, projected, emptyDir

安全上下文限制(SCC)

标签	Description	Default
allowHostDirVolumePlugin	Trident节点Pod挂载节点的根文件系统。	true
allowHostIPC	挂载NFS卷需要主机IPC才能与进行通信 nfsd。	true
allowHostNetwork	iscsiadm要求主机网络与iSCSI守护进程进行通信。	true
allowHostPID	需要使用主机PID检查是否存在 rpc-statd 正在节点上运行。	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

标签	Description	Default
`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` 因为容器运行时间和Kubernetes分发程序处理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`
`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`

文档的早期版本

如果您运行的不是Asta Trident 22.10、则可以查看以前版本的文档。

- "Astra Trident 22.07"
- "Astra Trident 22.04"
- "Astra Trident 22.01"

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

通知文件提供有关 NetApp 软件中使用的第三方版权和许可证的信息。