参考
Trident

NetApp
October 29, 2026

This PDF was generated from https://docs.netapp.com/zh-cn/trident-2606/trident-reference/ports.html on October 29, 2026. Always check docs.netapp.com for the latest.

目录

参考 1
Trident 端口 1
概述 1
Trident REST API 3
何时使用 REST API 3
使用 REST API 3
命令行选项 3
日志记录 4
Kubernetes 4
Docker 4
REST 4
Kubernetes 和 Trident 对象 5
这些对象如何相互作用? 5
Kubernetes `PersistentVolumeClaim` 对象 5
Kubernetes `PersistentVolume` 对象 7
Kubernetes `StorageClass` 对象 7
Kubernetes `VolumeSnapshotClass` 对象 10
Kubernetes `VolumeSnapshot` 对象 11
Kubernetes `VolumeSnapshotContent` 对象 11
Kubernetes `VolumeGroupSnapshotClass` 对象 11
Kubernetes `VolumeGroupSnapshot` 对象 12
Kubernetes `VolumeGroupSnapshotContent` 对象 12
Kubernetes `CustomResourceDefinition` 对象 12
Trident `StorageClass` 对象 13
Trident 后端对象 13
Trident `StoragePool` 对象 13
Trident `Volume` 对象 13
Trident `Snapshot` 对象 15
Trident `ResourceQuota` 对象 15
Pod 安全标准 (PSS) 和安全上下文约束 (SCC) 16
必需的 Kubernetes 安全上下文和相关字段 16
Pod 安全标准 (PSS) 17
Pod 安全策略 (PSP) 17
安全上下文约束 (SCC) 18

参考

Trident 端口

详细了解 Trident 用于通信的端口。

概述

Trident 使用各种端口在 Kubernetes 集群内以及与存储后端进行通信。以下是关键端口、其用途和安全注意事项的摘要。

- 出站焦点：Kubernetes 节点（控制器和工作节点）主要启动到存储 LIF/IP 的流量，因此 iptables 规则应允许从节点 IP 出站到这些端口上的特定存储 IP。避免宽泛的"任意对任意"规则。
- 入站限制：将内部 Trident 端口限制为集群内部流量（例如，使用像 Calico 这样的 CNI）。主机防火墙上没有不必要的入站曝光。
- 协议安全性：
  - 尽可能使用 TCP（更可靠）。
  - 如果敏感，为 iSCSI 启用 CHAP/IPsec；为管理启用 TLS/HTTPS（端口 443/8443）。
  - 对于 NFSv4（Trident 中的默认值），如果不需要，请修剪 UDP/较旧的 NFSv3 端口（例如，4045-4049）。
  - 仅限于受信任的子网；使用 Prometheus（可选端口 8001）等工具进行监控。

控制器节点的端口

这些端口主要用于 Trident operator（后端管理）。所有内部端口都是 pod 级；仅当主机防火墙干扰 CNI 时才允许在节点上使用。

端口/协议	方向	目的	驱动程序/协议	安全说明
TCP 8000	入站/出站（集群内部）	Trident REST 服务器（操作员-控制器通信）	全部	仅限于 pod CIDR；无外部暴露。
TCP 8443	入站/出站（集群内部）	反向通道 HTTPS（安全内部 API）	全部	TLS 加密；如果使用，则限制为 Kubernetes 服务网格。
TCP 8001	入站（集群内部，可选）	Prometheus 指标	全部	仅暴露给监控工具（例如，使用 RBAC）；如果未使用，则禁用。
TCP 443	出站	HTTPS 到 ONTAP SVM/集群管理 LIF	ONTAP（all）、ANF	需要 TLS 证书验证；仅限于管理 LIF IP。
TCP 8443	出站	HTTPS 到 E-Series Web Services Proxy	E 系列 (iSCSI)	默认 REST API；使用证书；可在后端 YAML 中配置。

工作节点的端口

这些端口用于 CSI 节点守护程序集和 pod 挂载。数据端口出站到存储数据 LIF；如果使用 NFSv3，则包括 NFSv3 附加组件（NFSv4 可选）。

端口/协议	方向	目的	驱动程序/协议	安全说明
TCP 17546	入站（本地到 pod）	CSI 节点活跃度/就绪探测	全部	可配置（--probe-port）；确保没有主机冲突；仅限本地。
TCP 8000	入站/出站（集群内部）	Trident REST 服务器	全部	如上所述；pod-internal。
TCP 8443	入站/出站（集群内部）	反向通道 HTTPS	全部	同上。
TCP 8001	入站（集群内部，可选）	Prometheus 指标	全部	同上。
TCP 443	出站	HTTPS 到 ONTAP SVM/集群管理 LIF	ONTAP（all ）、ANF	如上；用于发现。
TCP 8443	出站	HTTPS 到 E-Series Web Services Proxy	E 系列 (iSCSI)	同上。
TCP/UDP 111	出站	RPCBIND/portmapper	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	v3 必需；v4 可选（防火墙卸载）；如果仅使用 NFSv4，则限制。
TCP/UDP 2049	出站	NFS 守护进程	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	核心数据；众所周知；使用 TCP 实现可靠性。
TCP/UDP 635	出站	挂载守护进程	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	挂载；可以进行双向回调（如果需要，允许入站临时连接）。
UDP 4045	出站	NFS 锁定管理器 (nlockmgr)	ONTAP-NAS（NFSv3）	文件锁定；跳过 v4（pNFS 句柄）；仅限 UDP。
UDP 4046	出站	NFS 状态监控 (statd)	ONTAP-NAS（NFSv3）	通知；可能需要入站临时端口 (1024-65535) 进行回拨。
UDP 4049	出站	NFS 配额守护进程 (rquotad)	ONTAP-NAS（NFSv3）	配额；跳过 v4。
TCP 3260	出站	iSCSI 目标（发现/数据/CHAP）	ONTAP-SAN (iSCSI)、E-Series (iSCSI)	众所周知；通过此端口进行 CHAP 身份验证；启用双方 CHAP 以实现安全。
TCP 445	出站	SMB/CIFS	ONTAP-NAS (SMB)、ANF (SMB)	众所周知；使用 SMB3 进行加密（Trident annotation netapp.io/smb-encryption=true）。
TCP/UDP 88（可选）	出站	Kerberos 身份验证	ONTAP（带 Kerb 的 NFS/SMB/iSCSI）	如果使用 Kerberos（非默认）；到 AD 服务器，而不是存储。
TCP/UDP 389（可选）	出站	LDAP	ONTAP（使用 LDAP 的 NFS/SMB）	类似；用于名称解析/身份验证；限制为 AD。

在安装过程中，可以使用 `--probe-port`标志更改活动/就绪探头端口。请务必确保此端口未被工作

节点上的其他进程使用。

Trident REST API

虽然 "tridentctl 命令和选项" 是与 Trident REST API 交互的最简单方法，但如果您愿意，可以直接使用 REST 端点。

何时使用 REST API

REST API 对于在非 Kubernetes 部署中使用 Trident 作为独立二进制文件的高级安装非常有用。

为了提高安全性，默认情况下在 pod 内运行时，Trident `REST API` 被限制为 localhost。要更改此行为，需要在其 pod 配置中设置 Trident 的 `-address` 参数。

使用 REST API

有关如何调用这些 API 的示例，请传递 debug (`-d` 标志。有关详细信息，请参阅 "使用 tridentctl 管理 Trident"。

API 的工作原理如下：

GET

`GET <trident-address>/trident/v1/<object-type>`

列出此类型的所有对象。

`GET <trident-address>/trident/v1/<object-type>/<object-name>`

获取命名对象的详细信息。

POST

`POST <trident-address>/trident/v1/<object-type>`

创建指定类型的对象。

- 需要为要创建的对象提供 JSON 配置。有关每种对象类型的规格，请参阅 "使用 tridentctl 管理 Trident"。
- 如果对象已存在，行为会有所不同：后端更新现有对象，而所有其他对象类型将使操作失败。

DELETE

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`

删除命名资源。

与后端或存储类关联的卷将继续存在；必须单独删除这些卷。有关详细信息，请参阅 "使用 tridentctl 管理 Trident"。

命令行选项

Trident 为 Trident orchestrator 提供了多个命令行选项。您可以使用这些选项修改部署。

日志记录

-debug

启用调试输出。

-loglevel <level>

设置日志记录级别（debug、info、warn、error、fatal）。默认为 info。

Kubernetes

-k8s_pod

使用此选项或 `-k8s_api_server` 来启用 Kubernetes 支持。设置此选项会导致 Trident 使用其包含的 pod 的 Kubernetes 服务帐户凭据来联系 API 服务器。仅当 Trident 在启用了服务帐户的 Kubernetes 集群中作为 pod 运行时，此功能才有效。

-k8s_api_server <insecure-address:insecure-port>

使用此选项或 `-k8s_pod` 来启用 Kubernetes 支持。指定时，Trident 使用提供的不安全地址和端口连接到 Kubernetes API 服务器。这使 Trident 能够部署在 pod 之外；但是，它仅支持到 API 服务器的不安全连接。要安全地连接，请使用 `-k8s_pod` 选项在 pod 中部署 Trident。

Docker

-volume_driver <name>

注册 Docker 插件时使用的驱动程序名称。默认为 `netapp`。

-driver_port <port-number>

侦听此端口而不是 UNIX 域套接字。

-config <file>

必需；必须指定后端配置文件的此路径。

REST

-address <ip-or-host>

指定 Trident 的 REST 服务器应侦听的地址。默认为 localhost。在 localhost 上监听并在 Kubernetes pod 内运行时，无法从 pod 外部直接访问 REST 接口。使用 `-address ""`使 REST 接口可从 pod IP 地址访问。

可以将 Trident REST 接口配置为仅在 127.0.0.1（用于 IPv4）或 [::1]（用于 IPv6）下侦听和服务。

-port <port-number>

指定 Trident 的 REST 服务器应监听的端口。默认为 8000。

-rest

启用 REST 接口。默认为 true。

Kubernetes 和 Trident 对象

您可以通过读取和写入资源对象，使用 REST API 与 Kubernetes 和 Trident 进行交互。有几个资源对象决定了 Kubernetes 与 Trident、Trident 与存储以及 Kubernetes 与存储之间的关系。其中一些对象通过 Kubernetes 进行管理，其他对象通过 Trident 进行管理。

这些对象如何相互作用?

也许理解对象、它们的用途以及它们如何交互的最简单方法是遵循 Kubernetes 用户的单个存储请求：

1. 用户创建了一个 `PersistentVolumeClaim`，请求从管理员之前配置的 Kubernetes `StorageClass`中分配一个特定大小的 `PersistentVolume`。
2. Kubernetes `StorageClass` 将 Trident 识别为其配置程序，并包含告诉 Trident 如何为请求的类配置卷的参数。
3. Trident 查看其自己的 `StorageClass`，该名称用于标识匹配的 `Backends`和 `StoragePools`，它可以使用这些来为类配置卷。
4. Trident 在匹配的后端上配置存储并创建两个对象：Kubernetes 中的一个 `PersistentVolume`，它告诉 Kubernetes 如何查找、挂载和处理卷，以及 Trident 中的一个卷，它保留 `PersistentVolume`与实际存储之间的关系。
5. Kubernetes 将 `PersistentVolumeClaim` 绑定到新的 `PersistentVolume`。包含 `PersistentVolumeClaim` 的 Pod 会在其运行的任何主机上挂载该 PersistentVolume。
6. 用户使用指向 Trident 的 `VolumeSnapshotClass`，为现有 PVC 创建 `VolumeSnapshot`。
7. Trident 标识与 PVC 关联的卷，并在其后端创建卷的快照。它还创建了一个 `VolumeSnapshotContent`，指导 Kubernetes 如何识别快照。
8. 用户可以创建一个 `PersistentVolumeClaim`，并使用 `VolumeSnapshot`作为源。
9. Trident 识别所需的快照，并执行创建 `PersistentVolume`和 `Volume`所涉及的同一组步骤。

如需进一步了解 Kubernetes 对象，我们强烈建议您阅读 Kubernetes 文档的 "持久卷"部分。

Kubernetes `PersistentVolumeClaim` 对象

Kubernetes `PersistentVolumeClaim` 对象是 Kubernetes 集群用户发出的存储请求。

除了标准规范之外，如果要覆盖您在后端配置中设置的默认值，Trident 允许用户指定以下特定于卷的注释：

标注	卷选项	支持的驱动程序
trident.netapp.io/fileSystem	fileSystem	ontap-san, solidfire-san, ontap-san-economy
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas，ontap-san，solidfire-san，azure-netapp-files，ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ontap-nas，ontap-san
trident.netapp.io/protocol	protocol	任意

标注	卷选项	支持的驱动程序
trident.netapp.io/exportPolicy	exportPolicy	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	snapshotPolicy	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san
trident.netapp.io/snapshotReserve	snapshotReserve	ontap-nas、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotDirectory	snapshotDirectory	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/blockSize	blockSize	solidfire-san
trident.netapp.io/skipRecoveryQueue	skipRecoveryQueue	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，ontap-san-economy

如果创建的 PV 具有 `Delete`回收策略，则 `Trident` 会在 `PV` 释放时（即用户删除 `PVC` 时）同时删除 `PV` 和备份卷。如果删除操作失败，`Trident` 会将 `PV` 标记为此类状态，并定期重试该操作，直到操作成功或 `PV` 被手动删除。如果 `PV` 使用 `Retain`策略，`Trident` 会忽略它，并假设管理员将从 `Kubernetes` 和后端清理它，允许在删除之前备份或检查卷。请注意，删除 `PV` 不会导致 `Trident` 删除备份卷。您应该使用 `REST API (`tridentctl）将其删除。

Trident 支持使用 CSI 规范创建卷快照：您可以创建卷快照并将其用作克隆现有 PVC 的数据源。这样，PV 的时间点副本可以以快照的形式暴露给 Kubernetes。然后，可使用快照创建新的 PV。看一 `On-Demand Volume Snapshots`看这是如何运作的。

Trident 还提供用于创建克隆的 `cloneFromPVC`和 `splitOnClone`注释。您可以使用这些注释来克隆 PVC，而无需使用 CSI 实现。

示例如下：如果用户已经有一个名为 `mysql`的 `PVC`，则用户可以使用注释创建一个名为 `mysqlclone`的新 `PVC`，例如 `trident.netapp.io/cloneFromPVC: mysql`。使用此注释集，Trident 克隆与 mysql PVC 对应的卷，而不是从头开始配置卷。

请考虑以下几点：

- NetApp 建议克隆空闲卷。
- PVC 及其克隆应位于同一个 Kubernetes 命名空间中，并具有相同的存储类。
- 使用 `ontap-nas`和 `ontap-san`驱动程序时，可能需要将 PVC 注释 `trident.netapp.io/splitOnClone`与 `trident.netapp.io/cloneFromPVC`结合设置。将 `trident.netapp.io/splitOnClone`设置为 `true`时，Trident 会将克隆卷从父卷中拆分出来，从而使克隆卷的生命周期与其父卷完全脱钩，代价是失去一些存储效率。不设置 `trident.netapp.io/splitOnClone`或将其设置为 `false`会减少后端的空间消耗，代价是在父卷和克隆卷之间创建依赖关系，因此除非先删除克隆，否则无法删除父卷。拆分克隆有意义的场景是克隆一个空的数据库卷，预计该卷及其克隆会大大分化，并且不会从 ONTAP 提供的存储效率中受益。

该 `sample-input`目录包含用于 Trident 的 PVC 定义示例。有关与 Trident 卷相关的参数和设置的完整说明，请参阅。

Kubernetes `PersistentVolume` 对象

Kubernetes `PersistentVolume` 对象表示可供 Kubernetes 集群使用的一块存储。它的生命周期与使用它的 Pod 无关。

Trident 创建 `PersistentVolume`对象，并根据其配置的卷自动将其注册到 Kubernetes 集群。您无需自行管理。

当您创建引用基于 Trident 的 `StorageClass` PVC 时，Trident 使用相应的存储类来配置新卷，并为该卷注册新的 PV。在配置已配置的卷和对应的 PV 时，Trident 遵循以下规则：

- Trident 为 Kubernetes 生成一个 PV 名称以及用于配置存储的内部名称。在这两种情况下，它都确保名称在其范围内是唯一的。
- 卷的大小与 PVC 中请求的大小尽可能相匹配，但可能会四舍五入到最接近的可分配数量，具体取决于平台。

Kubernetes `StorageClass` 对象

Kubernetes `StorageClass` 对象在 `PersistentVolumeClaims`中按名称指定，以配置具有一组属性的存储。存储类本身标识要使用的置备程序，并以置备程序理解的术语定义该属性集。

它是需要由管理员创建和管理的两个基本对象之一。另一个是 Trident 后端对象。

使用 Trident 的 Kubernetes `StorageClass` 对象如下所示：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters: <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

这些参数是 Trident 特定的，告诉 Trident 如何为该类配置卷。

storage class 参数为：

属性	类型	必填项	说明
属性	map[string]string	不可以	请参见下面的属性部分
storagePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射
additionalStoragePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射

属性	类型	必填项	说明
excludeStoragePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射

存储属性及其可能的值可以分为存储池选择属性和 Kubernetes 属性。

存储池选择属性

这些参数确定应使用哪些 Trident 管理的存储池来配置给定类型的卷。

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

[1]：ONTAP Select 系统不支持

在大多数情况下，请求的值直接影响调配；例如，请求厚调配会导致厚调配的卷。但是，Element 存储池使用其提供的 IOPS 最小值和最大值来设置 QoS 值，而不是请求的值。在这种情况下，请求的值仅用于选择存储池。

理想情况下，您可以单独使用 `attributes`来模拟满足特定类别需求所需的存储质量。Trident 会自动发现并选择与您指定的所有 `attributes`匹配的存储池。

如果您发现自己无法使用 `attributes` 为某个类自动选择正确的池，则可以使用 `storagePools` 和 `additionalStoragePools` 参数进一步优化池，甚至可以选择一组特定的池。

您可以使用 `storagePools` 参数进一步限制与任何指定 `attributes` 匹配的池集。换句话说，Trident 使用由 `attributes` 和 `storagePools` 参数标识的池的交集进行配置。您可以单独使用一个参数，也可以同时使用两个参数。

您可以使用 `additionalStoragePools` 参数扩展 Trident 用于配置的池集，而不考虑 `attributes` 和 `storagePools` 参数选择的任何池。

您可以使用 `excludeStoragePools` 参数来筛选 Trident 用于配置的池集。使用此参数将删除所有匹配的池。

在 `storagePools`和 `additionalStoragePools`参数中，每个条目都采用 `<backend>:<storagePoolList>`形式，其中 `<storagePoolList>`是指定后端的存储池的逗号分隔列表。例如， `additionalStoragePools`的值可能看起来像 `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`。这些列表接受后端和列表值的正则表达式值。您可以使用 `tridentctl get backend`获取后端及其池的列表。

Kubernetes 属性

在动态配置期间，这些属性不会影响 Trident 对存储池/后端的选择。相反，这些属性只提供 Kubernetes 持久卷支持的参数。工作节点负责文件系统创建操作，并且可能需要文件系统实用程序，例如 xfsprogs。

属性	类型	值	说明	相关驱动程序	**Kubernetes** 版本
fsType	string	ext4、ext3、xfs	块卷的文件系统类型	solidfire-san，ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，ontap-san-economy	全部
allowVolumeExpansion	布尔值	true，false	启用或禁用对增加 PVC 尺寸的支持	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，ontap-san-economy，solidfire-san，azure-netapp-files	1.11+

属性	类型	值	说明	相关驱动程序	**Kubernetes** 版本
volumeBindingMode	string	立即，WaitForFirstConsumer	选择何时进行卷绑定和动态配置	全部	1.19 - 1.26

- `fsType` 参数用于控制 SAN LUNs 所需的文件系统类型。此外，Kubernetes 还会在存储类中使用 `fsType` 的存在来指示文件系统已存在。只有在设置了 `fsType` 时，才能使用 pod 的 `fsGroup` 安全上下文来控制卷的所有权。请参阅 "Kubernetes：为 Pod 或容器配置安全上下文"，了解如何使用 `fsGroup` 上下文设置卷所有权的概述。Kubernetes 仅在以下情况下应用 `fsGroup` 值：
  - `fsType` 在存储类中设置。
  - PVC 访问模式为 RWO。

  对于 NFS 存储驱动程序，文件系统已作为 NFS 导出的一部分存在。要使用 `fsGroup`，存储类仍需要指定 `fsType`。您可以将其设置为 `nfs`或任何非空值。
- 有关卷扩展的更多详细信息，请参见 "扩展卷"。
- Trident 安装程序捆绑包提供了几个示例存储类定义，可与 `sample-input/storage-class-*.yaml` 中的 Trident 一起使用。删除 Kubernetes 存储类也会导致相应的 Trident 存储类被删除。

Kubernetes `VolumeSnapshotClass` 对象

Kubernetes `VolumeSnapshotClass` 对象类似于 `StorageClasses`。它们有助于定义多个存储类，并由卷快照引用，以将快照与所需的快照类相关联。每个卷快照都与单个卷快照类相关联。

`VolumeSnapshotClass` 应由管理员定义以创建快照。使用以下定义创建卷快照类：

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

`driver`向 Kubernetes 指定 `csi-snapclass`类的卷快照请求由 Trident 处理。`deletionPolicy`指定必须删除快照时要采取的操作。当 `deletionPolicy`设置为 `Delete`时，删除快照时将删除卷快照对象以及存储集群上的底层快照。或者，将其设置为 `Retain` 表示保留 `VolumeSnapshotContent` 和物理快照。

Kubernetes `VolumeSnapshot` 对象

Kubernetes `VolumeSnapshot` 对象是创建卷的快照的请求。正如 PVC 表示用户对卷的请求一样，卷快照是用户为创建现有 PVC 的快照而发出的请求。

当卷快照请求进入时，Trident 会自动管理后端上卷的快照创建，并通过创建唯一 `VolumeSnapshotContent`对象公开快照。您可以从现有 PVC 创建快照，并在创建新 PVC 时将快照用作 DataSource。

> VolumeSnapshot 的生命周期独立于源 PVC：即使删除了源 PVC，快照也会持续存在。删除具有关联快照的 PVC 时，Trident 会将此 PVC 的后备卷标记为 **Deleting** 状态，但不会将其完全删除。删除所有关联的快照后，该卷将被删除。

Kubernetes `VolumeSnapshotContent` 对象

Kubernetes `VolumeSnapshotContent` 对象表示从已配置的卷中获取的快照。它类似于 `PersistentVolume`，表示存储集群上已配置的快照。与 `PersistentVolumeClaim`和 `PersistentVolume`对象类似，创建快照时， `VolumeSnapshotContent`对象会维护到已请求创建快照的 `VolumeSnapshot`对象的一对一映射。

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

当快照请求进入时，Trident 会在后端创建快照。创建快照后，Trident 将配置一个 `VolumeSnapshotContent`对象，从而将快照公开给 Kubernetes API。

> 通常，您无需管理 `VolumeSnapshotContent` 对象。此操作的例外情况是，您希望 "导入卷快照" 在 Trident 外部创建。

Kubernetes `VolumeGroupSnapshotClass` 对象

Kubernetes `VolumeGroupSnapshotClass` 对象类似于 `VolumeSnapshotClass`。它们有助于定义多个存储类，并由卷组快照引用，以将快照与所需的快照类相关联。每个卷组快照都与单个卷组快照类相关联。

`VolumeGroupSnapshotClass`
应由管理员定义，以创建快照组。使用以下定义创建卷组快照类：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
```

```
deletionPolicy: Delete
```

`driver` 向 Kubernetes 指定 `csi-group-snap-class` 类的卷组快照请求由 Trident 处理。 `deletionPolicy` 指定必须删除组快照时要采取的操作。当 `deletionPolicy` 设置为 `Delete`
时，删除快照时将删除卷组快照对象以及存储集群上的底层快照。或者，将其设置为 `Retain` 表示保留 `VolumeGroupSnapshotContent` 和物理快照。

Kubernetes `VolumeGroupSnapshot` 对象

Kubernetes `VolumeGroupSnapshot` 对象是创建多个卷的快照的请求。正如 PVC 表示用户对卷的请求一样，卷组快照是用户为创建现有 PVC 的快照而提出的请求。

当卷组快照请求进入时，Trident 会自动管理后端上卷的组快照的创建，并通过创建唯一 `VolumeGroupSnapshotContent`对象来公开快照。您可以从现有 PVC 创建快照，并在创建新 PVC 时将快照用作 DataSource。

VolumeGroupSnapshot 的生命周期独立于源 PVC：即使删除了源 PVC，快照也会持续存在。删除具有关联快照的 PVC 时，Trident 会将此 PVC 的后备卷标记为 **Deleting** 状态，但不会将其完全删除。删除所有关联的快照后，会删除卷组快照。

Kubernetes `VolumeGroupSnapshotContent` 对象

Kubernetes `VolumeGroupSnapshotContent` 对象表示从已配置的卷中获取的组快照。它类似于 `PersistentVolume`，表示存储集群上已配置的快照。与 `PersistentVolumeClaim`和 `PersistentVolume`对象类似，创建快照时， `VolumeSnapshotContent`对象会维护到已请求创建快照的 `VolumeSnapshot`对象的一对一映射。

`VolumeGroupSnapshotContent` 对象包含标识快照组的详细信息，例如 `volumeGroupSnapshotHandle` 和存储系统上存在的各个 `volumeSnapshotHandles`。

当快照请求进入时，Trident 在后端创建卷组快照。创建卷组快照后，Trident 配置一个 `VolumeGroupSnapshotContent`对象，从而将快照公开给 Kubernetes API。

Kubernetes `CustomResourceDefinition` 对象

Kubernetes 自定义资源是 Kubernetes API 中的端点，由管理员定义，用于对类似对象进行分组。Kubernetes 支持创建用于存储对象集合的自定义资源。您可以通过运行 `kubectl get crds` 来获取这些资源定义。

自定义资源定义 (CRD) 及其关联的对象元数据由 Kubernetes 存储在其元数据存储中。这消除了对 Trident 单独存储的需要。

Trident 使用 `CustomResourceDefinition`对象来保留 Trident 对象的身份，例如 Trident 后端、Trident 存储类和 Trident 卷。这些对象由 Trident 管理。此外，CSI 卷快照框架还引入了定义卷快照所需的一些 CRD。

CRD 是 Kubernetes 构造。上述定义的资源对象由 Trident 创建。举个简单的例子，当使用 `tridentctl`创建

后端时，会创建相应的 `tridentbackends CRD 对象供 Kubernetes 使用。

以下是关于 Trident CRD 需要记住的几点：

- 安装 Trident 后，将创建一组 CRD，可以像任何其他资源类型一样使用。
- 使用 `tridentctl uninstall` 命令卸载 Trident 时，Trident pod 会被删除，但不会清理创建的 CRD。请参阅"卸载 Trident"了解如何从头开始完全移除和重新配置 Trident。

Trident `StorageClass` 对象

Trident 为 Kubernetes `StorageClass`对象创建匹配的存储类，这些对象在其 `provisioner` 字段中指定 `csi.trident.netapp.io。存储类名称与其所代表的 Kubernetes `StorageClass`对象名称相同。

> 使用 Kubernetes 时，当注册使用 Trident 作为 provisioner 的 Kubernetes `StorageClass` 时，这些对象会自动创建。

存储类包括卷的一组要求。Trident 将这些要求与每个存储池中存在的属性进行匹配；如果它们匹配，则该存储池是使用该存储类调配卷的有效目标。

您可以创建存储类配置，通过使用 REST API 直接定义存储类。但是，对于 Kubernetes 部署，我们希望在注册新的 Kubernetes `StorageClass` 对象时创建它们。

Trident 后端对象

后端代表 Trident 在其上配置卷的存储提供程序；单个 Trident 实例可以管理任意数量的后端。

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

有关如何构造这些对象的详细信息，请参阅 "配置后端"。

Trident `StoragePool` 对象

存储池表示每个后端上可用于设置的不同位置。对于 ONTAP，这些对应于 SVM 中的聚合。对于 NetApp HCI/SolidFire，这些对应于管理员指定的 QoS 频段。每个存储池都有一组不同的存储属性，这些属性定义了其性能特征和数据保护特征。

与这里的其他对象不同，存储池候选项始终会被自动发现和管理。

Trident `Volume` 对象

卷是资源调配的基本单位，包括后端端点，例如 NFS 共享以及 iSCSI 和 FC LUN。在 Kubernetes 中，这些直接对应于 `PersistentVolumes`。创建卷时，请确保该卷具有存储类，该类确定可以调配该卷的位置以及大小。

> - 在 Kubernetes 中，这些对象是自动管理的。您可以查看它们以查看 Trident 配置的内容。
> - 删除具有关联快照的 PV 时，相应的 Trident 卷将更新为 **Deleting** 状态。要删除 Trident 卷，应删除卷的快照。

卷配置定义了已配置卷应具有的属性。

属性	类型	必填项	说明
version	string	不可以	Trident API 版本（"1"）
name	string	可以	要创建的卷的名称
storageClass	string	可以	调配卷时要使用的存储类
大小	string	可以	要设置的卷的大小（以字节为单位）
protocol	string	不可以	要使用的协议类型；"file" 或 "block"
internalName	string	不可以	存储系统上的对象名称；由 Trident 生成
cloneSourceVolume	string	不可以	ontap (nas, san) & solidfire-*：要从中克隆的卷的名称
splitOnClone	string	不可以	ONTAP (nas, san)：从其父项拆分克隆
snapshotPolicy	string	不可以	ontap-*：要使用的 Snapshot 策略
snapshotReserve	string	不可以	ontap-*：为 Snapshot 预留的卷百分比
exportPolicy	string	不可以	ontap-nas*：要使用的导出策略
snapshotDirectory	布尔值	不可以	ontap-nas*：快照目录是否可见
unixPermissions	string	不可以	ontap-nas*：初始 UNIX 权限
blockSize	string	不可以	solidfire-*：块/扇区大小
fileSystem	string	不可以	文件系统类型
skipRecoveryQueue	string	不可以	在卷删除过程中，绕过存储中的恢复队列并立即删除卷。

Trident 在创建卷时会生成 `internalName`。这包括两个步骤。首先，它将存储前缀（默认 `trident`或后端配置中的前缀）添加到卷名前面，从而生成形式为 `<prefix>-<volume-name>`的名称。然后继续清理名称，替换后端不允许的字符。对于 `ONTAP` 后端，它用下划线替换连字符（因此，内部名称变为 `<prefix>_<volume-name>`）。对于 Element 后端，它用连字符替换下划线。

您可以使用卷配置直接使用 REST API 配置卷，但在 Kubernetes 部署中，我们希望大多数用户使用标准的 Kubernetes `PersistentVolumeClaim` 方法。Trident 会在配置过程中自动创建此卷对象。

Trident `Snapshot` 对象

快照是卷的时间点副本，可用于配置新卷或恢复状态。在 Kubernetes 中，它们直接对应于 `VolumeSnapshotContent`对象。每个快照都与一个卷相关联，该卷是快照的数据源。

每个 `Snapshot` 对象都包含以下列出的属性：

属性	类型	必填项	说明
version	字符串	是	Trident API 版本（"1"）
name	字符串	是	Trident 快照对象的名称
internalName	字符串	是	存储系统上的 Trident 快照对象的名称
volumeName	字符串	是	为其创建快照的永久卷的名称
volumeInternalName	字符串	是	存储系统上关联的 Trident 卷对象的名称

在 Kubernetes 中，这些对象是自动管理的。您可以查看它们以查看 Trident 配置的内容。

当创建 Kubernetes `VolumeSnapshot`对象请求时，`Trident` 通过在后端存储系统上创建一个快照对象来工作。该快照对象的 `internalName`是通过将前缀 `snapshot-`与 `UID`的 `VolumeSnapshot`对象组合生成的（例如， `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。`volumeName`和 `volumeInternalName`通过获取后端卷的详细信息来填充。

Trident `ResourceQuota` 对象

Trident daemonset 使用 `system-node-critical`优先级类（Kubernetes 中可用的最高优先级类），以确保 Trident 能够在正常节点关闭期间识别和清理卷，并允许 Trident daemonset pod 在资源压力较大的集群中抢占较低优先级的工作负载。

为了实现这一点，Trident 使用了一个 `ResourceQuota`对象来确保满足 Trident daemonset 上的"system-node-critical"优先级类。在部署和创建 daemonset 之前，Trident 会查找 `ResourceQuota`对象，如果未发现，则应用它。

如果需要对默认 Resource Quota 和 Priority Class 进行更多控制，则可以使用 Helm chart 生成 `custom.yaml`或配置 `ResourceQuota`对象。

以下是优先处理 Trident daemonset 的 `ResourceQuota` 对象示例。

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
```

```
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values:
          - system-node-critical
```

有关 Resource Quotas 的详细信息，请参阅 "Kubernetes：资源配额"。

安装失败时清理 `ResourceQuota`

在创建 `ResourceQuota` 对象后安装失败的极少数情况下，请先尝试 "卸载"，然后重新安装。

如果这不起作用，请手动删除 `ResourceQuota` 对象。

删除 `ResourceQuota`

如果您希望控制自己的资源分配，则可以使用以下命令删除 Trident `ResourceQuota` 对象：

```
kubectl delete quota trident-csi -n trident
```

Pod 安全标准 (PSS) 和安全上下文约束 (SCC)

Kubernetes Pod 安全标准 (PSS) 和 Pod 安全策略 (PSP) 定义权限级别并限制 Pod 的行为。OpenShift 安全上下文约束 (SCC) 同样定义了特定于 OpenShift Kubernetes 引擎的 Pod 限制。为了提供这种定制，Trident 在安装期间启用某些权限。以下各节详细介绍 Trident 设置的权限。

PSS 取代了 Pod Security Policies (PSP)。PSP 在 Kubernetes v1.21 中被弃用，并将在 v1.25 中删除。有关详细信息，请参阅 "Kubernetes：安全"。

必需的 Kubernetes 安全上下文和相关字段

权限	说明
特权	CSI 要求挂载点为双向，这意味着 Trident 节点 Pod 必须运行特权容器。有关详细信息，请参阅 "Kubernetes：挂载传播"。
主机网络	iSCSI 守护程序所需。 `iscsiadm`管理 iSCSI 挂载并使用主机网络与 iSCSI 守护程序通信。
主机 IPC	NFS 使用进程间通信 (IPC) 与 NFSD 通信。
主机 PID	需要启动 `rpc-statd` 以用于 NFS。Trident 查询主机进程以确定 `rpc-statd`是否正在运行，然后再挂载 NFS 卷。

权限	说明

功能	该 `SYS_ADMIN`功能是特权容器默认功能的一部分。例如，`Docker` 为特权容器设置这些功能：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccomp 配置文件在特权容器中始终为"Unconfined"；因此，它不能在 Trident 中启用。
SELinux	在 OpenShift 上，特权容器在 `spc_t`（"Super Privileged Container"）域中运行，非特权容器在 `container_t`域中运行。在 `containerd`上，安装 `container-selinux`后，所有容器都在 `spc_t`域中运行，这有效地禁用了 `SELinux`。因此，`Trident` 不会向容器添加 `seLinuxOptions`。
DAC	特权容器必须以 root 身份运行。非特权容器以 root 身份运行，以访问 CSI 所需的 unix 套接字。

Pod 安全标准 (PSS)

标签	说明	默认
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	允许 Trident 控制器和节点进入安装命名空间。请勿更改命名空间标签。	`enforce: privileged enforce-version: <version of the current cluster or highest version of PSS tested.>`

更改命名空间标签可能会导致 pod 未被调度，出现"Error creating: ……"或"Warning: trident-csi-……"。如果发生这种情况，请检查 `privileged` 的命名空间标签是否已更改。如果是，请重新安装 Trident。

Pod 安全策略 (PSP)

字段	说明	默认
`allowPrivilegeEscalation`	特权容器必须允许特权提升。	`true`
`allowedCSIDrivers`	Trident 不使用内联 CSI 短暂卷。	空
`allowedCapabilities`	非特权 Trident 容器不需要比默认设置更多的功能，特权容器被授予所有可能的功能。	空
`allowedFlexVolumes`	Trident 不使用 "FlexVolume 驱动程序"，因此它们不包括在允许的卷列表中。	空
`allowedHostPaths`	Trident 节点 Pod 挂载节点的根文件系统，因此设置此列表没有任何好处。	空

字段	说明	默认
allowedProcMountTypes	Trident 不使用任何 ProcMountTypes。	空
allowedUnsafeSysctls	Trident 不需要任何不安全的 sysctls。	空
defaultAddCapabilities	特权容器中不需要添加任何功能。	空
defaultAllowPrivilegeEscalation	允许权限提升在每个 Trident pod 中处理。	false
forbiddenSysctls	不允许 sysctls。	空
fsGroup	Trident 容器以 root 身份运行。	RunAsAny
hostIPC	装载 NFS 卷需要主机 IPC 与 `nfsd` 进行通信	true
hostNetwork	iscsiadm 要求主机网络与 iSCSI 后台程序进行通信。	true
hostPID	需要主机 PID 来检查 `rpc-statd`是否在节点上运行。	true
hostPorts	Trident 不使用任何主机端口。	空
privileged	Trident 节点 Pod 必须运行特权容器才能挂载卷。	true
readOnlyRootFilesystem	Trident 节点 Pod 必须写入节点文件系统。	false
requiredDropCapabilities	Trident 节点 Pod 运行特权容器，不能丢弃功能。	none
runAsGroup	Trident 容器以 root 身份运行。	RunAsAny
runAsUser	Trident 容器以 root 身份运行。	runAsAny
runtimeClass	Trident 不使用 RuntimeClasses。	空
seLinux	Trident 没有设置 seLinuxOptions，因为当前容器运行时和 Kubernetes 发行版处理 SELinux 的方式存在差异。	空
supplementalGroups	Trident 容器以 root 身份运行。	RunAsAny
volumes	Trident Pod 需要这些卷插件。	hostPath, projected, emptyDir

安全上下文约束 (SCC)

标签	说明	默认
allowHostDirVolumePlugin	Trident 节点 Pod 挂载节点的根文件系统。	true

标签	说明	默认
`allowHostIPC`	装载 NFS 卷需要主机 IPC 与 `nfsd` 进行通信。	`true`
`allowHostNetwork`	iscsiadm 要求主机网络与 iSCSI 后台程序进行通信。	`true`
`allowHostPID`	需要主机 PID 来检查 `rpc-statd`是否在节点上运行。	`true`
`allowHostPorts`	Trident 不使用任何主机端口。	`false`
`allowPrivilegeEscalation`	特权容器必须允许特权提升。	`true`
`allowPrivilegedContainer`	Trident 节点 Pod 必须运行特权容器才能挂载卷。	`true`
`allowedUnsafeSysctls`	Trident 不需要任何不安全的 `sysctls`。	`none`
`allowedCapabilities`	非特权 Trident 容器不需要比默认设置更多的功能，特权容器被授予所有可能的功能。	空
`defaultAddCapabilities`	特权容器中不需要添加任何功能。	空
`fsGroup`	Trident 容器以 root 身份运行。	`RunAsAny`
`groups`	此 SCC 特定于 Trident 并绑定到其用户。	空
`readOnlyRootFilesystem`	Trident 节点 Pod 必须写入节点文件系统。	`false`
`requiredDropCapabilities`	Trident 节点 Pod 运行特权容器，不能丢弃功能。	`none`
`runAsUser`	Trident 容器以 root 身份运行。	`RunAsAny`
`seLinuxContext`	Trident 没有设置 `seLinuxOptions`，因为当前容器运行时和 Kubernetes 发行版处理 SELinux 的方式存在差异。	空
`seccompProfiles`	特权容器始终运行 "Unconfined"。	空
`supplementalGroups`	Trident 容器以 root 身份运行。	`RunAsAny`
`users`	提供了一个条目来将此 SCC 绑定到 Trident 命名空间中的 Trident 用户。	不适用
`volumes`	Trident Pod 需要这些卷插件。	`hostPath, downwardAPI, projected, emptyDir`

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。 美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 http://www.netapp.com/TM 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。