Referenz

Trident

NetApp
July 01, 2026

Inhalt

Referenz

Trident-Ports

Erfahren Sie mehr über die Ports, die Trident für die Kommunikation verwendet.

Überblick

Trident nutzt verschiedene Ports für die Kommunikation innerhalb von Kubernetes-Clustern und mit Storage-Backends. Im Folgenden finden Sie eine Zusammenfassung der wichtigsten Ports, ihrer Zwecke und Sicherheitsaspekte.

- **Ausgehender Fokus**: Kubernetes-Knoten (Controller und Worker) initiieren primär Datenverkehr zu Storage-LIFs/IPs, daher sollten iptables-Regeln ausgehenden Datenverkehr von Knoten-IPs zu bestimmten Storage-IPs auf diesen Ports zulassen. Vermeiden Sie allgemeine „Beliebig-zu-Beliebig“-Regeln.
- **Eingehende Beschränkungen**: Beschränken Sie interne Trident-Ports auf clusterinternen Datenverkehr (zum Beispiel mit CNI wie Calico). Keine unnötige eingehende Exponierung auf den Host-Firewalls.
- **Protokollsicherheit**:
  - Verwenden Sie nach Möglichkeit TCP (zuverlässiger).
  - Aktivieren Sie CHAP/IPsec für iSCSI, falls sensibel; TLS/HTTPS für das Management (Port 443/8443).
  - Für NFSv4 (Standard in Trident) entfernen Sie UDP-/ältere NFSv3-Ports (zum Beispiel 4045-4049), wenn sie nicht benötigt werden.
  - Beschränken Sie auf vertrauenswürdige Subnetze; überwachen Sie mit Tools wie Prometheus (optional Port 8001).

Anschlüsse für Controller-Knoten

Diese Ports sind primär für den Trident operator (Backend-Management) vorgesehen. Alle internen Ports sind auf Pod-Ebene; erlauben Sie sie auf Knoten nur, wenn die Host-Firewall mit CNI interferiert.

Port/Protokoll	Richtung	Zweck	Treiber/Protokoll	Sicherheitshinweise
TCP 8000	Eingehend/Ausgehend (cluster-intern)	Trident REST server (Operator-Controller-Kommunikation)	Alle	Beschränken Sie auf Pod-CIDRs; keine externe Exposition.
TCP 8443	Eingehend/Ausgehend (cluster-intern)	Backchannel HTTPS (sichere interne API)	Alle	TLS-verschlüsselt; auf Kubernetes Service Mesh beschränken, falls verwendet.
TCP 8001	Eingehend (clusterintern, optional)	Prometheus-Metriken	Alle	Nur für Überwachungstools (zum Beispiel mit RBAC) zugänglich machen; deaktivieren, wenn nicht verwendet.
TCP 443	Ausgehend	HTTPS zu ONTAP SVM/Cluster-Mgmt LIF	ONTAP (alle), ANF	TLS-Zertifikatvalidierung erforderlich; nur auf mgmt LIF-IPs beschränken.

Port/Protokoll	Richtung	Zweck	Treiber/Protokoll	Sicherheitshinweise
TCP 8443	Ausgehend	HTTPS-zu-E-Series-Web Services Proxy	E-Series (iSCSI)	Standardmäßige REST API; Verwendung von Zertifikaten; konfigurierbar im Backend-YAML.

Ports für Worker-Knoten

Diese Ports sind für CSI-Knoten-Daemonsets und Pod-Mounts vorgesehen. Datenports sind ausgehende Verbindungen zu Storage Data LIFs; fügen Sie NFSv3-Extras hinzu, wenn Sie NFSv3 verwenden (optional für NFSv4).

Port/Protokoll	Richtung	Zweck	Treiber/Protokoll	Sicherheitshinweise
TCP 17546	Eingehend (lokal zum Pod)	CSI-Knoten-Liveness/Readiness-Probes	Alle	Konfigurierbar (--probe-port); sicherstellen, dass keine Host-Konflikte bestehen; nur lokal.
TCP 8000	Eingehend/Ausgehend (cluster-intern)	Trident REST-Server	Alle	Wie oben; pod-internal.
TCP 8443	Eingehend/Ausgehend (cluster-intern)	Backchannel HTTPS	Alle	Wie oben.
TCP 8001	Eingehend (clusterintern, optional)	Prometheus-Metriken	Alle	Wie oben.
TCP 443	Ausgehend	HTTPS zu ONTAP SVM/Cluster-Mgmt LIF	ONTAP (alle), ANF	Wie oben; wird zur Ermittlung verwendet.
TCP 8443	Ausgehend	HTTPS-zu-E-Series-Web Services Proxy	E-Series (iSCSI)	Wie oben.
TCP/UDP 111	Ausgehend	RPCBIND/portmapper	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Erforderlich für v3; optional für v4 (firewall offload); einschränken, wenn ausschließlich NFSv4 verwendet wird.
TCP/UDP 2049	Ausgehend	NFS-Daemon	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Kerndaten; bekannt; TCP für Zuverlässigkeit verwenden.
TCP/UDP 635	Ausgehend	Mount-Daemon	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Montage; bidirektionale Rückrufe möglich (bei Bedarf eingehende ephemere Verbindungen zulassen).
UDP 4045	Ausgehend	NFS-Sperrmanager (nlockmgr)	ONTAP-NAS (NFSv3)	Dateisperrung; für v4 überspringen (pNFS handles); nur UDP.

Port/Protokoll	Richtung	Zweck	Treiber/Protokoll	Sicherheitshinweise
UDP 4046	Ausgehend	NFS-Statusmonitor (statd)	ONTAP-NAS (NFSv3)	Benachrichtigungen; möglicherweise werden eingehende ephemere Ports (1024-65535) für Rückrufe benötigt.
UDP 4049	Ausgehend	NFS-Quota-Daemon (rquotad)	ONTAP-NAS (NFSv3)	Kontingente; für v4 überspringen.
TCP 3260	Ausgehend	iSCSI-Ziel (Erkennung/Daten/CHAP)	ONTAP-SAN (iSCSI), E-Series (iSCSI)	Bekannt; CHAP-Authentifizierung über diesen Port; gegenseitiges CHAP zur Sicherheit aktivieren.
TCP 445	Ausgehend	SMB/CIFS	ONTAP-NAS (SMB), ANF (SMB)	Bekannt; verwenden Sie SMB3 mit Verschlüsselung (Trident annotation netapp.io/smb-encryption=true).
TCP/UDP 88 (optional)	Ausgehend	Kerberos-Authentifizierung	ONTAP (NFS/SMB/iSCSI mit Kerb)	Bei Verwendung von Kerberos (nicht Standardeinstellung); zu AD-Servern, nicht zum Storage.
TCP/UDP 389 (optional)	Ausgehend	LDAP	ONTAP (NFS/SMB mit LDAP)	Ähnlich; für Namensauflösung/Authentifizierung; auf AD beschränken.

Der Port für die Liveness-/Readiness-Prüfung kann während der Installation mit dem `--probe-port`-Flag geändert werden. Es ist wichtig sicherzustellen, dass dieser Port nicht von einem anderen Prozess auf den Worker-Knoten verwendet wird.

Trident REST API

Während "tridentctl commands und Optionen" die einfachste Möglichkeit sind, mit der Trident REST API zu interagieren, können Sie den REST-Endpunkt auch direkt verwenden, wenn Sie dies bevorzugen.

Wann sollte die REST API verwendet werden?

REST API ist nützlich für fortgeschrittene Installationen, die Trident als eigenständige Binärdatei in Nicht-Kubernetes-Bereitstellungen verwenden.

Für bessere Sicherheit ist Trident `REST API` standardmäßig auf localhost beschränkt, wenn es in einem Pod ausgeführt wird. Um dieses Verhalten zu ändern, müssen Sie das Argument von Trident `-address` in seiner Pod-Konfiguration festlegen.

Verwendung der REST API

Beispiele dafür, wie diese APIs aufgerufen werden, erhalten Sie, wenn Sie das Debug-(`-d`-Flag übergeben. Weitere Informationen finden sich unter "Trident mit tridentctl verwalten".

Die API funktioniert wie folgt:

GET

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

Listet alle Objekte dieses Typs auf.

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

Ruft die Details des benannten Objekts ab.

POST

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

Erzeugt ein Objekt des angegebenen Typs.

- Für die Erstellung des Objekts ist eine JSON-Konfiguration erforderlich. Die Spezifikation der einzelnen Objekttypen finden Sie unter "Trident mit tridentctl verwalten".
- Wenn das Objekt bereits existiert, variiert das Verhalten: Backends aktualisieren das bestehende Objekt, während bei allen anderen Objekttypen der Vorgang fehlschlägt.

LÖSCHEN

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

Löscht die benannte Ressource.

> Zu Backends oder Speicherklassen gehörende Volumes bleiben bestehen; diese müssen separat gelöscht werden. Weitere Informationen finden sich unter "Trident mit tridentctl verwalten".

Befehlszeilenoptionen

Trident stellt mehrere Befehlszeilenoptionen für den Trident Orchestrator bereit. Mit diesen Optionen können Sie Ihre Bereitstellung anpassen.

Protokollierung

`-debug`

Aktiviert die Debugging-Ausgabe.

`-loglevel <level>`

Legt das Protokollierungslevel (debug, info, warn, error, fatal) fest. Standardmäßig auf info gesetzt.

Kubernetes

-k8s_pod

Verwenden Sie diese Option oder `-k8s_api_server`, um die Kubernetes-Unterstützung zu aktivieren. Wenn Sie diese Einstellung festlegen, verwendet Trident die Kubernetes-Dienstkontoinformationen seines eigenen Pods, um den API-Server zu kontaktieren. Dies funktioniert nur, wenn Trident als Pod in einem Kubernetes-Cluster mit aktivierten Dienstkonten ausgeführt wird.

-k8s_api_server <insecure-address:insecure-port>

Verwenden Sie diese Option oder `-k8s_pod`, um die Kubernetes-Unterstützung zu aktivieren. Wenn angegeben, verbindet sich Trident mit dem Kubernetes-API-Server über die bereitgestellte unsichere Adresse und den Port. Dadurch kann Trident außerhalb eines Pods bereitgestellt werden; es werden jedoch nur unsichere Verbindungen zum API-Server unterstützt. Um eine sichere Verbindung herzustellen, stellen Sie Trident in einem Pod mit der `-k8s_pod` Option bereit.

Docker

-volume_driver <name>

Name des Treibers, der bei der Registrierung des Docker-Plugins verwendet wird. Standardmäßig `netapp`.

-driver_port <port-number>

Lauschen Sie auf diesem Port statt auf einem UNIX-Domain-Socket.

-config <file>

Erforderlich; Sie müssen diesen Pfad zu einer Backend-Konfigurationsdatei angeben.

REST

-address <ip-or-host>

Gibt die Adresse an, auf der der REST-Server von Trident lauschen soll. Standardmäßig ist dies localhost. Wenn auf localhost gelauscht wird und Trident in einem Kubernetes-Pod ausgeführt wird, ist die REST-Schnittstelle von außerhalb des Pods nicht direkt zugänglich. Verwenden Sie `-address ""`, um die REST-Schnittstelle von der Pod-IP-Adresse aus zugänglich zu machen.

> Trident REST interface kann so konfiguriert werden, dass sie nur unter 127.0.0.1 (für IPv4) oder [::1] (für IPv6) lauscht und Anfragen beantwortet.

-port <port-number>

Gibt den Port an, auf dem der REST-Server von Trident lauschen soll. Standardmäßig 8000.

-rest

Aktiviert die REST-Schnittstelle. Standardmäßig auf true gesetzt.

Kubernetes- und Trident-Objekte

Sie können mit Kubernetes und Trident über REST-APIs interagieren, indem Sie Ressourcenobjekte lesen und schreiben. Es gibt verschiedene Ressourcenobjekte, die die Beziehungen zwischen Kubernetes und Trident, Trident und Speicher sowie Kubernetes und Speicher bestimmen. Einige dieser Objekte werden über Kubernetes

und die anderen über Trident verwaltet.

Wie interagieren die Objekte miteinander?

Vielleicht ist der einfachste Weg, die Objekte, ihren Zweck und ihre Interaktion zu verstehen, einer einzelnen Speicheranfrage eines Kubernetes-Benutzers zu folgen:

1. Ein Benutzer erstellt eine `PersistentVolumeClaim` Anfrage für eine neue `PersistentVolume` einer bestimmten Größe von einem Kubernetes `StorageClass`, der zuvor vom Administrator konfiguriert wurde.
2. Kubernetes `StorageClass` identifiziert Trident als seinen Provisioner und enthält Parameter, die Trident angeben, wie ein Volume für die angeforderte Klasse bereitgestellt werden soll.
3. Trident betrachtet seine eigenen `StorageClass` mit demselben Namen, der die passende `Backends` und `StoragePools` identifiziert, die es zur Bereitstellung von Volumes für die Klasse verwenden kann.
4. Trident stellt Speicher auf einem passenden Backend bereit und erstellt zwei Objekte: ein `PersistentVolume` in Kubernetes, das Kubernetes mitteilt, wie das Volume gefunden, eingebunden und behandelt werden soll, und ein Volume in Trident, das die Beziehung zwischen dem `PersistentVolume` und dem eigentlichen Speicher aufrechterhält.
5. Kubernetes bindet die `PersistentVolumeClaim` an die neue `PersistentVolume`. Pods, die die `PersistentVolumeClaim` Mount-Anweisung enthalten, mounten dieses PersistentVolume auf jedem Host, auf dem sie ausgeführt werden.
6. Ein Benutzer erstellt eine `VolumeSnapshot` einer bestehenden PVC, indem er eine `VolumeSnapshotClass` verwendet, die auf Trident zeigt.
7. Trident identifiziert das dem PVC zugeordnete Volume und erstellt einen Snapshot dieses Volumes im Backend. Es erstellt außerdem eine `VolumeSnapshotContent`, die Kubernetes anweist, wie der Snapshot identifiziert werden kann.
8. Ein Benutzer kann eine `PersistentVolumeClaim` mit `VolumeSnapshot` als Quelle erstellen.
9. Trident identifiziert den erforderlichen Snapshot und führt die gleichen Schritte aus, die auch beim Erstellen eines `PersistentVolume` und eines `Volume` erforderlich sind.

Für weiterführende Informationen zu Kubernetes-Objekten empfehlen wir Ihnen dringend, den "Persistente Volumes" Abschnitt der Kubernetes-Dokumentation zu lesen.

Kubernetes `PersistentVolumeClaim`-Objekte

Ein Kubernetes `PersistentVolumeClaim`-Objekt ist eine Speicheranforderung, die von einem Kubernetes-Cluster-Benutzer gestellt wird.

Zusätzlich zur Standardspezifikation ermöglicht Trident den Benutzern, die folgenden volumespezifischen Annotationen anzugeben, wenn sie die in der Backend-Konfiguration festgelegten Standardeinstellungen überschreiben möchten:

Anmerkung	Volume-Option	Unterstützte Treiber
trident.netapp.io/fileSystem	fileSystem	ontap-san, solidfire-san,ontap-san-economy

Anmerkung	Volume-Option	Unterstützte Treiber
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	Protokoll	beliebig
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

Wenn das erstellte PV die `Delete` Reclaim-Richtlinie hat, löscht Trident sowohl das PV als auch das zugehörige Volume, sobald das PV freigegeben wird (das heißt, wenn der Benutzer das PVC löscht). Schlägt der Löschvorgang fehl, markiert Trident das PV entsprechend und wiederholt die Operation regelmäßig, bis sie erfolgreich ist oder das PV manuell gelöscht wird. Verwendet das PV die `Retain` Richtlinie, ignoriert Trident es und geht davon aus, dass der Administrator es aus Kubernetes und dem Backend entfernt, sodass das Volume vor seiner Entfernung gesichert oder überprüft werden kann. Beachten Sie, dass das Löschen des PV nicht dazu führt, dass Trident das zugehörige Volume löscht. Sie sollten es über die REST-API entfernen (`tridentctl`.

Trident unterstützt die Erstellung von Volume-Snapshots gemäß der CSI-Spezifikation: Sie können einen Volume-Snapshot erstellen und ihn als Datenquelle zum Klonen vorhandener PVCs verwenden. Auf diese Weise können zeitpunktgenaue Kopien von PVs in Form von Snapshots für Kubernetes bereitgestellt werden. Die Snapshots können dann zum Erstellen neuer PVs verwendet werden. Sehen Sie sich `On-Demand Volume Snapshots` an, um zu sehen, wie das funktioniert.

Trident bietet außerdem die `cloneFromPVC` und `splitOnClone` Annotationen zum Erstellen von Klonen. Mit diesen Annotationen können Sie ein PVC klonen, ohne die CSI-Implementierung verwenden zu müssen.

Hier ist ein Beispiel: Wenn ein Benutzer bereits eine PVC namens `mysql` besitzt, kann der Benutzer eine neue PVC namens `mysqlclone` erstellen, indem er die Annotation wie `trident.netapp.io/cloneFromPVC: mysql` verwendet. Mit dieser Annotation klont Trident das Volume, das der mysql PVC entspricht, anstatt ein Volume von Grund auf bereitzustellen.

Beachten Sie folgende Punkte:

- NetApp empfiehlt das Klonen eines ungenutzten Volumes.

- Ein PVC und sein Klon sollten sich im selben Kubernetes-Namespace befinden und die gleiche Storage Class haben.
- Mit den `ontap-nas` und `ontap-san` Treibern kann es wünschenswert sein, die PVC-Annotation `trident.netapp.io/splitOnClone` in Verbindung mit `trident.netapp.io/cloneFromPVC` festzulegen. Wenn `trident.netapp.io/splitOnClone` auf `true` gesetzt ist, trennt Trident das geklonte Volume vom übergeordneten Volume und entkoppelt so den Lebenszyklus des geklonten Volumes vollständig vom übergeordneten Volume, allerdings auf Kosten eines geringeren Speicherwirkungsgrads. Wenn `trident.netapp.io/splitOnClone` nicht gesetzt ist oder auf `false` gesetzt wird, führt dies zu einem geringeren Speicherplatzverbrauch im Backend, allerdings entstehen dadurch Abhängigkeiten zwischen dem übergeordneten und dem geklonten Volume, sodass das übergeordnete Volume erst gelöscht werden kann, wenn der Klon zuvor gelöscht wurde. Ein Szenario, in dem das Aufteilen des Klons sinnvoll ist, ist das Klonen eines leeren Datenbank-Volumes, bei dem zu erwarten ist, dass sich Volume und Klon stark unterscheiden und nicht von den Speichereffizienzen profitieren, die ONTAP bietet.

Das `sample-input` Verzeichnis enthält Beispiele für PVC-Definitionen zur Verwendung mit Trident. Siehe für eine vollständige Beschreibung der Parameter und Einstellungen, die mit Trident-Volumes verbunden sind.

Kubernetes `PersistentVolume`-Objekte

Ein Kubernetes `PersistentVolume`-Objekt repräsentiert einen Speicherbereich, der dem Kubernetes-Cluster zur Verfügung gestellt wird. Es hat einen Lebenszyklus, der unabhängig von dem Pod ist, der ihn verwendet.

Trident erstellt `PersistentVolume` Objekte und registriert sie automatisch beim Kubernetes-Cluster basierend auf den Volumes, die es bereitstellt. Sie müssen diese nicht selbst verwalten.

Wenn Sie eine PVC erstellen, die auf ein Trident-basiertes `StorageClass` verweist, stellt Trident ein neues Volume mit der entsprechenden Speicherklasse bereit und registriert ein neues PV für dieses Volume. Bei der Konfiguration des bereitgestellten Volumes und des zugehörigen PV befolgt Trident die folgenden Regeln:

- Trident generiert einen PV-Namen für Kubernetes und einen internen Namen, den es zur Bereitstellung des Speichers verwendet. In beiden Fällen stellt es sicher, dass die Namen innerhalb ihres Geltungsbereichs eindeutig sind.
- Die Größe des Volumens entspricht so genau wie möglich der im PVC angeforderten Größe, kann jedoch je nach Plattform auf die nächstzuweisbare Menge aufgerundet werden.

Kubernetes `StorageClass`-Objekte

Kubernetes `StorageClass`-Objekte werden anhand ihres Namens in `PersistentVolumeClaims` spezifiziert, um Speicher mit einer Reihe von Eigenschaften bereitzustellen. Die Speicherklasse selbst identifiziert den zu verwendenden Provisionierer und definiert diese Eigenschaften in einer für den Provisionierer verständlichen Weise.

Es handelt sich um eines von zwei grundlegenden Objekten, die vom Administrator erstellt und verwaltet werden müssen. Das andere ist das Trident Backend-Objekt.

Ein Kubernetes `StorageClass`-Objekt, das Trident verwendet, sieht folgendermaßen aus:

```
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
```

Diese Parameter sind Trident-spezifisch und geben Trident an, wie Volumes für die Klasse bereitgestellt werden sollen.

Die Parameter der Speicherklasse sind:

Attribut	Typ	Erforderlich	Beschreibung
Attribute	map[string]string	nein	Siehe den Abschnitt Attribute unten
storagePools	map[string]StringList	nein	Zuordnung von Backend-Namen zu Listen von Speicherpools innerhalb
additionalStoragePools	map[string]StringList	nein	Zuordnung von Backend-Namen zu Listen von Speicherpools innerhalb
excludeStoragePools	map[string]StringList	nein	Zuordnung von Backend-Namen zu Listen von Speicherpools innerhalb

Speicherattribute und ihre möglichen Werte können in Attribute zur Auswahl von Speicherpools und Kubernetes-Attribute eingeteilt werden.

Auswahlattribute für Speicherpools

Diese Parameter legen fest, welche von Trident verwalteten Speicherpools zur Bereitstellung von Volumes eines bestimmten Typs verwendet werden sollen.

Attribut	Typ	Werte	Angebot	Anfrage	Unterstützt von
Medien[1]	Zeichenkette	hdd, hybrid, ssd	Pool enthält Medien dieses Typs; hybrid bedeutet beides	Medientyp angegeben	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san
provisioningType	Zeichenkette	dünn, dick	Pool unterstützt diese Bereitstellungsmethode	Bereitstellungsmethode angegeben	dick: alle ontap; dünn: alle ontap & solidfire-san

Attribut	Typ	Werte	Angebot	Anfrage	Unterstützt von
backendType	Zeichenkette	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san, azure-netapp-files, ontap-san-economy	Pool gehört zu dieser Art von Backend	Backend angegeben	Alle Treiber
Momentaufnahmen	bool	wahr, falsch	Pool unterstützt Volumes mit Snapshots	Volume mit aktivierten Snapshots	ontap-nas, ontap-san, solidfire-san
Klone	bool	wahr, falsch	Pool unterstützt das Klonen von Volumes	Volume mit aktivierten Klonen	ontap-nas, ontap-san, solidfire-san
Verschlüsselung	bool	wahr, falsch	Pool unterstützt verschlüsselte Volumes	Volume mit aktivierter Verschlüsselung	ontap-nas, ontap-nas-economy, ontap-nas-flexgroups, ontap-san
IOPS	int	positive ganze Zahl	Pool ist in der Lage, IOPS in diesem Bereich zu garantieren	Volume garantiert diese IOPS	solidfire-san

[1]: Wird von ONTAP Select-Systemen nicht unterstützt

In den meisten Fällen beeinflussen die angeforderten Werte die Bereitstellung direkt; beispielsweise führt die Anforderung von Thick Provisioning zu einem Thick-Provisioning-Volume. Ein Element-Speicherpool verwendet jedoch seine angebotenen minimalen und maximalen IOPS-Werte zur Festlegung der QoS-Werte, anstatt des angeforderten Wertes. In diesem Fall wird der angeforderte Wert nur zur Auswahl des Speicherpools verwendet.

Im Idealfall können Sie `attributes` allein verwenden, um die Eigenschaften des Speichers zu modellieren, die Sie benötigen, um die Anforderungen einer bestimmten Klasse zu erfüllen. Trident erkennt und wählt automatisch Speicherpools aus, die *allen* `attributes` entsprechen, die Sie angeben.

Falls Sie nicht in der Lage sind, `attributes` die richtigen Pools für eine Klasse automatisch auszuwählen, können Sie die `storagePools` und `additionalStoragePools` Parameter verwenden, um die Pools weiter einzugrenzen oder sogar eine bestimmte Gruppe von Pools auszuwählen.

Sie können den `storagePools` Parameter verwenden, um die Menge der Pools, die mit einem angegebenen `attributes` übereinstimmen, weiter einzuschränken. Anders ausgedrückt: Trident verwendet für die Bereitstellung die Schnittmenge der Pools, die durch die `attributes` und `storagePools` Parameter identifiziert werden. Sie können entweder einen der Parameter allein oder beide zusammen verwenden.

Sie können den `additionalStoragePools` Parameter verwenden, um die Menge der Pools zu erweitern, die Trident für die Bereitstellung verwendet, unabhängig von den durch die `attributes` und `storagePools` Parameter ausgewählten Pools.

Sie können den `excludeStoragePools` Parameter verwenden, um die von Trident für die Bereitstellung verwendeten Pools zu filtern. Durch die Verwendung dieses Parameters werden alle übereinstimmenden Pools entfernt.

In den `storagePools` und `additionalStoragePools` Parametern hat jeder Eintrag die Form `<backend>:<storagePoolList>`, wobei `<storagePoolList>` eine durch Kommas getrennte Liste von Speicherpools für das angegebene Backend ist. Ein Wert für `additionalStoragePools` könnte beispielsweise so aussehen: `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`. Diese Listen akzeptieren Regex-Werte sowohl für das Backend als auch für die Listenwerte. Sie können `tridentctl get backend` verwenden, um die Liste der Backends und ihrer Pools zu erhalten.

Kubernetes-Attribute

Diese Attribute haben keinen Einfluss auf die Auswahl von Speicherpools/Backends durch Trident während der dynamischen Bereitstellung. Stattdessen liefern diese Attribute einfach Parameter, die von Kubernetes Persistent Volumes unterstützt werden. Worker-Knoten sind für Dateisystem-Erstellungsvorgänge verantwortlich und benötigen möglicherweise Dateisystem-Dienstprogramme wie xfsprogs.

Attribut	Typ	Werte	Beschreibung	Relevante Treiber	Kubernetes-Version
fsType	Zeichenkette	ext4, ext3, xfs	Der Dateisystemtyp für Blockvolumes	solidfire-san, ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy	Alle
allowVolumeExpansion	boolescher Wert	wahr, falsch	Aktivieren oder Deaktivieren der Unterstützung für das Vergrößern der PVC-Größe	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy, solidfire-san, azure-netapp-files	1.11+
volumeBindingMode	Zeichenkette	Sofort, WaitForFirstConsumer	Wählen Sie aus, wann die Volumenbindung und die dynamische Bereitstellung erfolgt.	Alle	1.19 - 1.26

- Der `fsType` Parameter wird verwendet, um den gewünschten Dateisystemtyp für SAN-LUNs zu steuern. Außerdem verwendet Kubernetes das Vorhandensein von `fsType` in einer Storage-Klasse, um anzuzeigen, dass ein Dateisystem existiert. Die Volume-Eigentümerschaft kann mit dem `fsGroup` Sicherheitskontext eines Pods nur gesteuert werden, wenn `fsType` gesetzt ist. Siehe "Kubernetes: Konfigurieren eines Security Context für einen Pod oder Container" für einen Überblick zur Festlegung der Volume-Eigentümerschaft mithilfe des `fsGroup` Kontexts. Kubernetes wendet den `fsGroup` Wert nur an, wenn:

  - `fsType` wird in der Speicherklasse festgelegt.
  - Der PVC-Zugriffsmodus ist RWO.

  Bei NFS-Speichertreibern ist ein Dateisystem bereits als Teil des NFS-Exports vorhanden. Um `fsGroup` die Speicherklasse zu verwenden, muss dennoch ein `fsType` angegeben werden. Sie können diesen auf `nfs` oder einen beliebigen Wert ungleich null setzen.

- Weitere Einzelheiten zur Volumenerweiterung finden Sie unter "Volumen erweitern".
- Das Trident-Installationspaket enthält mehrere Beispieldefinitionen für Speicherklassen zur Verwendung mit Trident in `sample-input/storage-class-*.yaml`. Das Löschen einer Kubernetes-Speicherklasse führt dazu, dass die entsprechende Trident-Speicherklasse ebenfalls gelöscht wird.

Kubernetes `VolumeSnapshotClass`-Objekte

Kubernetes `VolumeSnapshotClass`-Objekte sind analog zu `StorageClasses`. Sie helfen dabei, mehrere Speicherklassen zu definieren und werden von Volume-Snapshots referenziert, um den Snapshot mit der erforderlichen Snapshot-Klasse zu verknüpfen. Jeder Volume-Snapshot ist mit einer einzigen Volume-Snapshot-Klasse verknüpft.

A `VolumeSnapshotClass` sollte von einem Administrator definiert werden, um Snapshots zu erstellen. Eine Volume-Snapshot-Klasse wird mit der folgenden Definition erstellt:

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

Das `driver` gibt Kubernetes an, dass Anfragen für Volume-Snapshots der `csi-snapclass` Klasse von Trident verarbeitet werden. Das `deletionPolicy` gibt die Aktion an, die ausgeführt werden soll, wenn ein Snapshot gelöscht werden muss. Wenn `deletionPolicy` auf `Delete` gesetzt ist, werden sowohl die Volume-Snapshot-Objekte als auch der zugrunde liegende Snapshot auf dem Storage-Cluster entfernt, wenn ein Snapshot gelöscht wird. Alternativ bedeutet das Setzen auf `Retain`, dass `VolumeSnapshotContent` und der physische Snapshot erhalten bleiben.

Kubernetes `VolumeSnapshot`-Objekte

Ein Kubernetes `VolumeSnapshot` Objekt ist eine Anfrage zum Erstellen eines Snapshots eines Volumes.

Genau wie ein PVC eine Anfrage eines Benutzers für ein Volume darstellt, ist ein Volume Snapshot eine Anfrage eines Benutzers zum Erstellen eines Snapshots eines bestehenden PVC.

Wenn eine Volume-Snapshot-Anfrage eingeht, verwaltet Trident automatisch die Erstellung des Snapshots für das Volume im Backend und stellt den Snapshot durch die Erstellung eines eindeutigen `VolumeSnapshotContent` Objekts bereit. Sie können Snapshots von bestehenden PVCs erstellen und die Snapshots als DataSource beim Erstellen neuer PVCs verwenden.

> Der Lebenszyklus eines VolumeSnapshot ist unabhängig vom Quell-PVC: Ein Snapshot bleibt auch nach dem Löschen des Quell-PVC erhalten. Beim Löschen eines PVC mit zugehörigen Snapshots markiert Trident das zugehörige Datenträgervolumen im Status **Deleting**, entfernt es aber nicht vollständig. Das Volume wird entfernt, wenn alle zugehörigen Snapshots gelöscht sind.

Kubernetes `VolumeSnapshotContent`-Objekte

Ein Kubernetes `VolumeSnapshotContent`-Objekt stellt einen Snapshot eines bereits bereitgestellten Volumes dar. Es ist analog zu einem `PersistentVolume` und kennzeichnet einen bereitgestellten Snapshot im Speichercluster. Ähnlich wie `PersistentVolumeClaim` und `PersistentVolume` Objekten, wenn ein Snapshot erstellt wird, verwaltet das `VolumeSnapshotContent` Objekt eine Eins-zu-Eins-Zuordnung zu dem `VolumeSnapshot` Objekt, das die Snapshot-Erstellung angefordert hat.

Das `VolumeSnapshotContent` Objekt enthält Details, die den Snapshot eindeutig identifizieren, wie zum Beispiel das `snapshotHandle`. Dies `snapshotHandle` ist eine eindeutige Kombination aus dem Namen des PV und dem Namen des `VolumeSnapshotContent` Objekts.

Wenn eine Snapshot-Anfrage eingeht, erstellt Trident den Snapshot im Backend. Nachdem der Snapshot erstellt wurde, konfiguriert Trident ein `VolumeSnapshotContent` Objekt und stellt den Snapshot somit der Kubernetes-API zur Verfügung.

> Normalerweise müssen Sie das `VolumeSnapshotContent` Objekt nicht verwalten. Eine Ausnahme besteht, wenn Sie ein "einen Volume-Snapshot importieren" außerhalb von Trident erstellt haben.

Kubernetes `VolumeGroupSnapshotClass`-Objekte

Kubernetes `VolumeGroupSnapshotClass`-Objekte sind analog zu `VolumeSnapshotClass`. Sie helfen dabei, mehrere Speicherklassen zu definieren und werden von Volume-Gruppen-Snapshots referenziert, um den Snapshot mit der erforderlichen Snapshot-Klasse zu verknüpfen. Jeder Volume-Gruppen-Snapshot ist mit genau einer Volume-Gruppen-Snapshot-Klasse verknüpft.

A `VolumeGroupSnapshotClass` sollte von einem Administrator definiert werden, um eine Gruppe von Snapshots zu erstellen. Eine Volume Group Snapshot Class wird mit der folgenden Definition erstellt:

```
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
```

Die `driver` gibt in Kubernetes an, dass Anfragen für Volume-Gruppen-Snapshots der `csi-group-snap-class` Klasse von Trident verarbeitet werden. Die `deletionPolicy` gibt die Aktion an, die ausgeführt werden soll, wenn ein Gruppen-Snapshot gelöscht werden muss. Wenn `deletionPolicy` auf `Delete` gesetzt ist, werden sowohl die Volume-Gruppen-Snapshot-Objekte als auch der zugrunde liegende Snapshot auf dem Storage-Cluster entfernt, wenn ein Snapshot gelöscht wird. Alternativ bedeutet das Setzen auf `Retain`, dass `VolumeGroupSnapshotContent` und der physische Snapshot erhalten bleiben.

Kubernetes `VolumeGroupSnapshot`-Objekte

Ein Kubernetes `VolumeGroupSnapshot`-Objekt ist eine Anfrage zum Erstellen eines Snapshots mehrerer Volumes. Genau wie ein PVC eine Anfrage eines Benutzers für ein Volume darstellt, ist ein Volume-Gruppen-Snapshot eine Anfrage eines Benutzers zum Erstellen eines Snapshots eines bestehenden PVC.

Wenn eine Volume-Gruppen-Snapshot-Anfrage eingeht, verwaltet Trident automatisch die Erstellung des Gruppen-Snapshots für die Volumes im Backend und stellt den Snapshot durch die Erstellung eines eindeutigen `VolumeGroupSnapshotContent` Objekts bereit. Sie können Snapshots von bestehenden PVCs erstellen und die Snapshots als DataSource beim Erstellen neuer PVCs verwenden.

> Der Lebenszyklus eines VolumeGroupSnapshot ist unabhängig vom Quell-PVC: Ein Snapshot bleibt auch nach dem Löschen des Quell-PVC erhalten. Beim Löschen eines PVC mit zugehörigen Snapshots markiert Trident das zugehörige Datenträgervolumen im Status **Deleting**, entfernt es aber nicht vollständig. Die Volume Group Snapshot wird entfernt, wenn alle zugehörigen Snapshots gelöscht sind.

Kubernetes `VolumeGroupSnapshotContent`-Objekte

Ein Kubernetes `VolumeGroupSnapshotContent`-Objekt repräsentiert einen Gruppen-Snapshot, der von einem bereits bereitgestellten Volume erstellt wurde. Es ist analog zu einem `PersistentVolume` und kennzeichnet einen bereitgestellten Snapshot im Speichercluster. Ähnlich wie `PersistentVolumeClaim` und `PersistentVolume` Objekten, wenn ein Snapshot erstellt wird, verwaltet das `VolumeSnapshotContent` Objekt eine Eins-zu-Eins-Zuordnung zu dem `VolumeSnapshot` Objekt, das die Snapshot-Erstellung angefordert hat.

Das `VolumeGroupSnapshotContent` Objekt enthält Details, die die Snapshot-Gruppe identifizieren, wie zum Beispiel das `volumeGroupSnapshotHandle` und einzelne volumeSnapshotHandles, die auf dem Speichersystem vorhanden sind.

Wenn eine Snapshot-Anfrage eingeht, erstellt Trident den Volume-Group-Snapshot im Backend. Nachdem der Volume-Group-Snapshot erstellt wurde, konfiguriert Trident ein `VolumeGroupSnapshotContent` Objekt und stellt den Snapshot somit der Kubernetes-API zur Verfügung.

Kubernetes `CustomResourceDefinition`-Objekte

Kubernetes Custom Resources sind Endpunkte in der Kubernetes-API, die vom Administrator definiert werden und zur Gruppierung ähnlicher Objekte verwendet werden. Kubernetes unterstützt die Erstellung von Custom Resources zum Speichern einer Sammlung von Objekten. Sie können diese Ressourcendefinitionen abrufen, indem Sie `kubectl get crds` ausführen.

Benutzerdefinierte Ressourcendefinitionen (CRDs) und die zugehörigen Objektmetadaten werden von Kubernetes in seinem Metadatenspeicher abgelegt. Dadurch entfällt die Notwendigkeit eines separaten Speichers für Trident.

Trident verwendet `CustomResourceDefinition` Objekte, um die Identität von Trident-Objekten wie Trident Backends, Trident Storage Classes und Trident Volumes zu erhalten. Diese Objekte werden von Trident verwaltet. Darüber hinaus führt das CSI Volume Snapshot Framework einige CRDs ein, die zur Definition von Volume Snapshots erforderlich sind.

CRDs sind ein Konstrukt von Kubernetes. Objekte der oben definierten Ressourcen werden von Trident erstellt. Als einfaches Beispiel: Wenn ein Backend mit `tridentctl` erstellt wird, wird ein entsprechendes `tridentbackends` CRD-Objekt zur Verwendung durch Kubernetes erzeugt.

Hier sind einige Punkte, die Sie bei den CRDs von Trident beachten sollten:

- Bei der Installation von Trident wird ein Satz von CRDs erstellt, die wie jeder andere Ressourcentyp verwendet werden können.
- Bei der Deinstallation von Trident mit dem `tridentctl uninstall`-Befehl werden die Trident-Pods gelöscht, aber die erstellten CRDs werden nicht bereinigt. Siehe "Trident deinstallieren", um zu verstehen, wie Trident vollständig entfernt und von Grund auf neu konfiguriert werden kann.

Trident `StorageClass` Objekte

Trident erstellt passende Speicherklassen für Kubernetes `StorageClass`-Objekte, die `csi.trident.netapp.io` in ihrem Provisioner-Feld angegeben sind. Der Name der Speicherklasse entspricht dem Namen des Kubernetes `StorageClass`-Objekts, das sie repräsentiert.

> Mit Kubernetes werden diese Objekte automatisch erstellt, wenn ein Kubernetes `StorageClass`, das Trident als Provisioner verwendet, registriert wird.

Speicherklassen umfassen eine Reihe von Anforderungen für Volumes. Trident gleicht diese Anforderungen mit den Attributen ab, die in jedem Speicherpool vorhanden sind; wenn sie übereinstimmen, ist dieser Speicherpool ein gültiges Ziel für die Bereitstellung von Volumes mit dieser Speicherklasse.

Sie können Speicherklassenkonfigurationen erstellen, um Speicherklassen direkt über die REST-API zu definieren. Bei Kubernetes-Bereitstellungen erwarten wir jedoch, dass sie bei der Registrierung neuer Kubernetes `StorageClass` Objekte erstellt werden.

Trident Backend-Objekte

Backends stellen die Speicheranbieter dar, auf denen Trident Volumes bereitstellt; eine einzelne Trident Instanz kann beliebig viele Backends verwalten.

Dies ist einer der beiden Objekttypen, die Sie selbst erstellen und verwalten. Der andere ist das Kubernetes `StorageClass`-Objekt.

Weitere Informationen darüber, wie Sie diese Objekte erstellen, finden Sie unter "Backends konfigurieren".

Trident `StoragePool` Objekte

Speicherpools repräsentieren die verschiedenen, für die Bereitstellung auf jedem Backend verfügbaren Speicherorte. Für ONTAP entsprechen diese Aggregaten in SVMs. Für NetApp HCI/SolidFire entsprechen sie den vom Administrator festgelegten QoS-Bändern. Jeder Speicherpool verfügt über eine Reihe spezifischer Speicherattribute, die seine Leistungs- und Datenschutzmerkmale definieren.

Im Gegensatz zu den anderen Objekten hier werden Speicherpool-Kandidaten immer automatisch erkannt und verwaltet.

Trident `Volume` Objekte

Volumes sind die grundlegende Bereitstellungseinheit und umfassen Backend-Endpunkte wie NFS-Freigaben sowie iSCSI- und FC-LUNs. In Kubernetes entsprechen diese direkt `PersistentVolumes`. Wenn Sie ein Volume erstellen, stellen Sie sicher, dass es eine Speicherklasse hat, die bestimmt, wo dieses Volume bereitgestellt werden kann, sowie eine Größe.

- In Kubernetes werden diese Objekte automatisch verwaltet. Sie können sie anzeigen, um zu sehen, was Trident bereitgestellt hat.
- Beim Löschen eines Persistent Volume (PV) mit zugehörigen Snapshots wird das entsprechende Trident-Volume auf den Status **Deleting** aktualisiert. Damit das Trident-Volume gelöscht werden kann, sollten Sie die Snapshots des Volumes entfernen.

Eine Volumenkonfiguration definiert die Eigenschaften, die ein bereitgestelltes Volumen haben sollte.

Attribut	Typ	Erforderlich	Beschreibung
Version	Zeichenkette	nein	Version der Trident API ("1")
Name	Zeichenkette	Ja	Name des zu erstellenden Volumes
storageClass	Zeichenkette	Ja	Speicherklasse, die beim Bereitstellen des Volumes verwendet werden soll
Größe	Zeichenkette	Ja	Größe des Volumens, das in Bytes bereitgestellt werden soll
Protokoll	Zeichenkette	nein	Zu verwendender Protokolltyp; „Datei“ oder „Block“
internalName	Zeichenkette	nein	Name des Objekts auf dem Speichersystem; generiert von Trident

Attribut	Typ	Erforderlich	Beschreibung
cloneSourceVolume	Zeichenkette	nein	ontap (nas, san) & solidfire-*: Name des Volumes, von dem geklont werden soll
splitOnClone	Zeichenkette	nein	ontap (nas, san): Trennen Sie den Klon von seinem übergeordneten Objekt
snapshotPolicy	Zeichenkette	nein	ontap-*: Zu verwendende Snapshot-Richtlinie
snapshotReserve	Zeichenkette	nein	ontap-*: Prozentsatz des für Snapshots reservierten Volumens
exportPolicy	Zeichenkette	nein	ontap-nas*: Zu verwendende Exportrichtlinie
snapshotDirectory	bool	nein	ontap-nas*: Ob das Snapshot-Verzeichnis sichtbar ist
unixPermissions	Zeichenkette	nein	ontap-nas*: Initial UNIX-Berechtigungen
blockSize	Zeichenkette	nein	solidfire-*: Block-/Sektorgröße
fileSystem	Zeichenkette	nein	Dateisystemtyp
skipRecoveryQueue	Zeichenkette	nein	Während des Löschens eines Volumes wird die Wiederherstellungswartes chlange im Speicher umgangen und das Volume sofort gelöscht.

Trident generiert `internalName` beim Erstellen des Volumes einen Namen. Dies besteht aus zwei Schritten. Zuerst wird dem Volume-Namen das Speicherpräfix vorangestellt (entweder das Standard `trident` oder das in der Backend-Konfiguration festgelegte Präfix), sodass ein Name der Form `<prefix>-<volume-name>` entsteht. Anschließend wird der Name bereinigt, indem Zeichen ersetzt werden, die im Backend nicht zulässig sind. Für ONTAP-Backends werden Bindestriche durch Unterstriche ersetzt (dadurch wird der interne Name zu `<prefix>_<volume-name>`). Für Element-Backends werden Unterstriche durch Bindestriche ersetzt.

Sie können Volume-Konfigurationen verwenden, um Volumes direkt über die REST-API bereitzustellen, aber bei Kubernetes-Bereitstellungen gehen wir davon aus, dass die meisten Benutzer die Standardmethode von Kubernetes `PersistentVolumeClaim` verwenden. Trident erstellt dieses Volume-Objekt automatisch im Rahmen des Bereitstellungsprozesses.

Trident `Snapshot` Objekte

Snapshots sind eine zeitpunktgenaue Kopie von Volumes, die zur Bereitstellung neuer Volumes oder zur Wiederherstellung des Zustands verwendet werden können. In Kubernetes entsprechen diese direkt

`VolumeSnapshotContent` Objekten. Jeder Snapshot ist einem Volume zugeordnet, das die Quelle der Daten für den Snapshot ist.

Jedes `Snapshot` Objekt umfasst die unten aufgeführten Eigenschaften:

Attribut	Typ	Erforderlich	Beschreibung
Version	Zeichenkette	Ja	Version der Trident API ("1")
Name	Zeichenkette	Ja	Name des Trident Snapshot-Objekts
internalName	Zeichenkette	Ja	Name des Trident-Snapshot-Objekts auf dem Speichersystem
volumeName	Zeichenkette	Ja	Name des Persistent Volume, für das der Snapshot erstellt wird
volumeInternalName	Zeichenkette	Ja	Name des zugehörigen Trident volume object im Speichersystem

In Kubernetes werden diese Objekte automatisch verwaltet. Sie können sie anzeigen, um zu sehen, was Trident bereitgestellt hat.

Wenn eine Kubernetes `VolumeSnapshot`-Objektanforderung erstellt wird, arbeitet Trident, indem es ein Snapshot-Objekt auf dem zugrunde liegenden Speichersystem erstellt. Die `internalName` dieses Snapshot-Objekts wird durch die Kombination des Präfixes `snapshot-` mit der `UID` des `VolumeSnapshot` Objekts generiert (zum Beispiel, `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`). `volumeName` und `volumeInternalName` werden durch das Abrufen der Details des zugrunde liegenden Volumes befüllt.

Trident `ResourceQuota` Objekt

Der Trident Daemonset verwendet eine `system-node-critical` Priority Class – die höchste in Kubernetes verfügbare Priority Class –, um sicherzustellen, dass Trident Volumes während des ordnungsgemäßen Herunterfahrens von Knoten identifizieren und bereinigen kann und damit Trident Daemonset Pods Workloads mit niedrigerer Priorität in Clustern mit hohem Ressourcendruck verdrängen können.

Um dies zu erreichen, verwendet Trident ein `ResourceQuota` Objekt, um sicherzustellen, dass die „system-node-critical“ Priority Class beim Trident-Daemonset erfüllt ist. Vor der Bereitstellung und der Erstellung des Daemonsets sucht Trident nach dem `ResourceQuota` Objekt und wendet es an, falls es nicht gefunden wird.

Wenn Sie mehr Kontrolle über das Standardressourcenkontingent und die Prioritätsklasse benötigen, können Sie eine `custom.yaml` oder das `ResourceQuota` Objekt mithilfe eines Helm-Charts generieren oder konfigurieren.

Nachfolgend ein Beispiel für ein `ResourceQuota`-Objekt, das dem Trident daemonset Priorität einräumt.

```
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
```

Weitere Informationen zu Ressourcenquoten finden Sie unter "Kubernetes: Ressourcenkontingente".

Aufräumarbeiten durchführen `ResourceQuota` falls die Installation fehlschlägt

Sollte die Installation in dem seltenen Fall fehlschlagen, nachdem das `ResourceQuota` Objekt erstellt wurde, versuchen Sie zuerst "Deinstallation" und installieren Sie dann erneut.

Sollte das nicht funktionieren, entfernen Sie das `ResourceQuota` Objekt manuell.

Entfernen `ResourceQuota`

Wenn Sie Ihre Ressourcenzuweisung lieber selbst steuern möchten, können Sie das Trident `ResourceQuota` Objekt mit dem folgenden Befehl entfernen:

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

Pod-Sicherheitsstandards (PSS) und Security Context Constraints (SCC)

Die Kubernetes Pod Security Standards (PSS) und Pod Security Policies (PSP) definieren Berechtigungsstufen und schränken das Verhalten von Pods ein. OpenShift Security Context Constraints (SCC) definieren analog dazu pod-spezifische Einschränkungen für die OpenShift Kubernetes Engine. Um diese Anpassung zu ermöglichen, aktiviert Trident während der Installation bestimmte Berechtigungen. Die folgenden Abschnitte beschreiben die von Trident gesetzten Berechtigungen.

PSS ersetzt Pod Security Policies (PSP). PSP wurde in Kubernetes v1.21 als veraltet markiert und wird in v1.25 entfernt. Weitere Informationen finden Sie unter "Kubernetes: Sicherheit".

Erforderlicher Kubernetes-Sicherheitskontext und zugehörige Felder

Berechtigung	Beschreibung
Privilegiert	CSI erfordert bidirektionale Mountpunkte, was bedeutet, dass der Trident-Node-Pod einen privilegierten Container ausführen muss. Weitere Informationen finden sich unter "Kubernetes: Mount Propagation".
Host-Netzwerk	Erforderlich für den iSCSI-Daemon. `iscsiadm` Verwaltet iSCSI-Mounts und nutzt das Host-Netzwerk zur Kommunikation mit dem iSCSI-Daemon.
Host-IPC	NFS nutzt Interprozesskommunikation (IPC), um mit dem NFSD zu kommunizieren.
Host-PID	Erforderlich, um `rpc-statd` für NFS zu starten. Trident fragt Hostprozesse ab, um festzustellen, ob `rpc-statd` läuft, bevor NFS-Volumes eingebunden werden.
Fähigkeiten	Die `SYS_ADMIN` Fähigkeit wird als Teil der Standardfähigkeiten für privilegierte Container bereitgestellt. Zum Beispiel setzt Docker diese Fähigkeiten für privilegierte Container: `CapPrm: 0000003fffffffff` `CapEff: 0000003fffffffff`
Seccomp	Das Seccomp-Profil ist in privilegierten Containern immer "Unconfined"; daher kann es in Trident nicht aktiviert werden.
SELinux	Auf OpenShift werden privilegierte Container in der `spc_t` („Super Privileged Container“) Domäne ausgeführt, und unprivilegierte Container werden in der `container_t` Domäne ausgeführt. Auf `containerd`, mit `container-selinux` installiert, werden alle Container in der `spc_t` Domäne ausgeführt, was SELinux effektiv deaktiviert. Daher fügt Trident `seLinuxOptions` nicht zu Containern hinzu.
DAC	Privilegierte Container müssen als Root ausgeführt werden. Nicht privilegierte Container werden als Root ausgeführt, um auf die von CSI benötigten Unix-Sockets zuzugreifen.

Pod Security Standards (PSS)

Etikett	Beschreibung	Standard
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	Ermöglicht dem Trident Controller und den Knoten die Aufnahme in den Installations-Namespace. Ändern Sie das Namespace-Label nicht.	`enforce: privileged enforce-version: <version of the current cluster or highest version of PSS tested.>`

Das Ändern der Namespace-Labels kann dazu führen, dass Pods nicht eingeplant werden, eine „Error creating: …“- oder „Warning: trident-csi-…“-Meldung erscheint. Wenn dies geschieht, überprüfen Sie, ob das Namespace-Label für `privileged` geändert wurde. Falls ja, installieren Sie Trident neu.

Pod-Sicherheitsrichtlinien (PSP)

Feld	Beschreibung	Standard
`allowPrivilegeEscalation`	Privilegierte Container müssen eine Privilegien-Eskalation zulassen.	`true`
`allowedCSIDrivers`	Trident verwendet keine Inline-CSI-Ephemeral-Volumes.	Leer
`allowedCapabilities`	Nicht privilegierte Trident-Container benötigen nicht mehr Berechtigungen als die Standardmenge, und privilegierten Containern werden alle möglichen Berechtigungen gewährt.	Leer
`allowedFlexVolumes`	Trident verwendet keine "FlexVolume-Treiber", daher sind sie nicht in der Liste der erlaubten Volumes.	Leer
`allowedHostPaths`	Der Trident-Node-Pod mountet das Root-Dateisystem des Nodes, daher bringt das Festlegen dieser Liste keinen Vorteil.	Leer
`allowedProcMountTypes`	Trident verwendet kein `ProcMountTypes`.	Leer
`allowedUnsafeSysctls`	Trident benötigt keine unsicheren `sysctls`.	Leer
`defaultAddCapabilities`	Für privilegierte Container müssen keine Funktionen hinzugefügt werden.	Leer
`defaultAllowPrivilegeEscalation`	Die Gewährung von Privilegienerweiterungen wird in jedem Trident Pod gehandhabt.	`false`
`forbiddenSysctls`	Keine `sysctls` sind erlaubt.	Leer

Feld	Beschreibung	Standard
`fsGroup`	Trident Container werden als root ausgeführt.	`RunAsAny`
`hostIPC`	Das Einbinden von NFS-Volumes erfordert Host-IPC, um mit `nfsd`	`true`
`hostNetwork`	iscsiadm benötigt das Host-Netzwerk, um mit dem iSCSI-Daemon zu kommunizieren.	`true`
`hostPID`	Host-PID ist erforderlich, um zu überprüfen, ob `rpc-statd` auf dem Knoten läuft.	`true`
`hostPorts`	Trident verwendet keine Host-Ports.	Leer
`privileged`	Trident node pods müssen einen privilegierten Container ausführen, um Volumes einzubinden.	`true`
`readOnlyRootFilesystem`	Trident node pods müssen in das Node-Dateisystem schreiben.	`false`
`requiredDropCapabilities`	Trident node pods führen einen privilegierten Container aus und können keine Capabilities ablegen.	`none`
`runAsGroup`	Trident Container werden als root ausgeführt.	`RunAsAny`
`runAsUser`	Trident Container werden als root ausgeführt.	`runAsAny`
`runtimeClass`	Trident verwendet `RuntimeClasses` nicht.	Leer
`seLinux`	Trident setzt `seLinuxOptions` nicht, da es derzeit Unterschiede gibt, wie Container-Runtimes und Kubernetes-Distributionen SELinux handhaben.	Leer
`supplementalGroups`	Trident Container werden als root ausgeführt.	`RunAsAny`
`volumes`	Trident Pods benötigen diese Volume-Plugins.	`hostPath, projected, emptyDir`

Security Context Constraints (SCC)

Etiketten	Beschreibung	Standard
`allowHostDirVolumePlugin`	Trident-Knoten-Pods mounten das Root-Dateisystem des Knotens.	`true`

Etiketten	Beschreibung	Standard
`allowHostIPC`	Das Mounten von NFS-Volumes erfordert Host-IPC, um mit `nfsd` zu kommunizieren.	`true`
`allowHostNetwork`	iscsiadm benötigt das Host-Netzwerk, um mit dem iSCSI-Daemon zu kommunizieren.	`true`
`allowHostPID`	Host-PID ist erforderlich, um zu überprüfen, ob `rpc-statd` auf dem Knoten läuft.	`true`
`allowHostPorts`	Trident verwendet keine Host-Ports.	`false`
`allowPrivilegeEscalation`	Privilegierte Container müssen eine Privilegien-Eskalation zulassen.	`true`
`allowPrivilegedContainer`	Trident node pods müssen einen privilegierten Container ausführen, um Volumes einzubinden.	`true`
`allowedUnsafeSysctls`	Trident benötigt keine unsicheren `sysctls`.	`none`
`allowedCapabilities`	Nicht privilegierte Trident-Container benötigen nicht mehr Berechtigungen als die Standardmenge, und privilegierten Containern werden alle möglichen Berechtigungen gewährt.	Leer
`defaultAddCapabilities`	Für privilegierte Container müssen keine Funktionen hinzugefügt werden.	Leer
`fsGroup`	Trident Container werden als root ausgeführt.	`RunAsAny`
`groups`	Diese SCC ist spezifisch für Trident und ist an ihren Benutzer gebunden.	Leer
`readOnlyRootFilesystem`	Trident node pods müssen in das Node-Dateisystem schreiben.	`false`
`requiredDropCapabilities`	Trident node pods führen einen privilegierten Container aus und können keine Capabilities ablegen.	`none`
`runAsUser`	Trident Container werden als root ausgeführt.	`RunAsAny`
`seLinuxContext`	Trident setzt `seLinuxOptions` nicht, da es derzeit Unterschiede gibt, wie Container-Runtimes und Kubernetes-Distributionen SELinux handhaben.	Leer

Etiketten	Beschreibung	Standard
`seccompProfiles`	Privilegierte Container laufen immer "Unconfined".	Leer
`supplementalGroups`	Trident Container werden als root ausgeführt.	`RunAsAny`
`users`	Ein Eintrag wird bereitgestellt, um diese SCC an den Trident Benutzer im Trident-Namensraum zu binden.	n/a
`volumes`	Trident Pods benötigen diese Volume-Plugins.	`hostPath, downwardAPI, projected, emptyDir`