Trident 26.02 documentation

Trident

NetApp
August 04, 2026

This PDF was generated from https://docs.netapp.com/us-en/trident-2602/index.html on August 04, 2026.
Always check docs.netapp.com for the latest.

Table of Contents

Trident 26.02 documentation . . . 1
Release notes . . . 2
What's new . . . 2
What's new in 26.02.1 . . . 2
What's new in 26.02 . . . 3
What's new in 25.10 . . . 5
Changes in 25.06.2 . . . 7
Changes in 25.06.1 . . . 7
Changes in 25.06 . . . 7
Changes in 25.02.1 . . . 9
Changes in 25.02 . . . 10
Changes in 24.10.1 . . . 11
Changes in 24.10 . . . 12
Changes in 24.06 . . . 13
Changes in 24.02 . . . 14
Changes in 23.10 . . . 15
Changes in 23.07.1 . . . 15
Changes in 23.07 . . . 15
Changes in 23.04 . . . 16
Changes in 23.01.1 . . . 17
Changes in 23.01 . . . 17
Changes in 22.10 . . . 18
Changes in 22.07 . . . 19
Changes in 22.04 . . . 20
Changes in 22.01.1 . . . 21
Changes in 22.01.0 . . . 21
Changes in 21.10.1 . . . 22
Changes in 21.10.0 . . . 22
Known issues . . . 23
Find more information . . . 24
Earlier versions of documentation . . . 24
NetApp Trident support for ONTAP ASA r2 storage systems . . . 24
Supported operations . . . 25
Unsupported operations . . . 25
Known issues . . . 25
VolumeSnapshots not reaching ReadyToUse state . . . 25
Restoring Restic backups of large files can fail . . . 26
Get started . . . 27
Learn about Trident . . . 27
Learn about Trident . . . 27
Trident architecture . . . 28
Concepts . . . 31
Quick start for Trident . . . 35

What's next? 36
Requirements 36
Critical information about Trident 36
Supported frontends (orchestrators) 36
Supported backends (storage) 37
Trident support for KubeVirt and OpenShift Virtualization 37
Feature requirements 38
Tested host operating systems 38
Host configuration 38
Storage system configuration 39
Trident ports 39
Container images and corresponding Kubernetes versions 39
Install Trident 40
Learn about Trident installation 40
Critical information about Trident 26.02 40
Before you begin 40
Choose your installation method 40
Choose your installation mode 42
Select the process based on your method and mode 43
Moving between installation methods 44
Other known configuration options 44
Install using Trident operator 44
Manually deploy the Trident operator (Standard mode) 44
Manually deploy the Trident operator (Offline mode) 50
Deploy Trident operator using Helm (Standard mode) 56
Deploy Trident operator using Helm (Offline mode) 64
Customize Trident operator installation 72
Install using tridentctl 84
Install using tridentctl 84
Customize tridentctl installation 88
Install using OpenShift certified operator 89
Install Trident using OpenShift OperatorHub 89
Switch to the OpenShift certified Trident operator 92
Use Trident 96
Prepare the worker node 96
Selecting the right tools 96
Node service discovery 96
NFS volumes 97
iSCSI volumes 97
NVMe/TCP volumes 101
SCSI over FC volumes 102
Prepare to provision SMB volumes 105
Configure and manage backends 106
Configure backends 106
Azure NetApp Files 106

Google Cloud NetApp Volumes . . . 126
Configure a NetApp HCI or SolidFire backend . . . 153
ONTAP SAN drivers . . . 158
ONTAP NAS drivers . . . 187
Amazon FSx for NetApp ONTAP . . . 224
Create backends with kubectl . . . 261
Manage backends . . . 268
Create and manage storage classes . . . 278
Create a storage class . . . 278
Manage storage classes . . . 281
Provision and manage volumes . . . 283
Provision a volume . . . 283
Expand volumes . . . 287
Understand RWX NVMe subsystem limits . . . 298
Controller scalability . . . 299
Automatic volume expansion . . . 304
Manage Autogrow Policies . . . 310
Import volumes . . . 319
Customize volume names and labels . . . 331
Share an NFS volume across namespaces . . . 334
Clone volumes across namespaces . . . 338
Replicate volumes using SnapMirror . . . 340
Use CSI Topology . . . 347
Work with snapshots . . . 354
Work with volume group snapshots . . . 362
Manage and monitor Trident . . . 367
Upgrade Trident . . . 367
Upgrade Trident . . . 367
Upgrade with the operator . . . 368
Upgrade with tridentctl . . . 373
Manage Trident using tridentctl . . . 373
Commands and global flags . . . 373
Command options and flags . . . 375
Plugin support . . . 380
Monitor Trident . . . 380
Overview . . . 380
Step 1: Define a Prometheus target . . . 380
Step 2: Create a Prometheus ServiceMonitor . . . 381
Step 3: Query Trident metrics with PromQL . . . 382
Learn about Trident AutoSupport telemetry . . . 384
Disable Trident metrics . . . 384
Uninstall Trident . . . 385
Determine the original installation method . . . 385
Uninstall a Trident operator installation . . . 385
Uninstall a `tridentctl` installation . . . 386

Trident for Docker . . . 387
Prerequisites for deployment . . . 387
Verify the requirements . . . 387
NVMe tools . . . 389
FC tools . . . 390
Deploy Trident . . . 392
Docker managed plugin method (version 1.13/17.03 and later) . . . 392
Traditional method (version 1.12 or earlier) . . . 394
Start Trident at system startup . . . 395
Upgrade or uninstall Trident . . . 396
Upgrade . . . 396
Uninstall . . . 398
Work with volumes . . . 398
Create a volume . . . 398
Remove a volume . . . 399
Clone a volume . . . 399
Access externally created volumes . . . 400
Driver-specific volume options . . . 401
Collect logs . . . 406
Collect logs for troubleshooting . . . 406
General troubleshooting tips . . . 406
Manage multiple Trident instances . . . 407
Steps for Docker managed plugin (version 1.13/17.03 or later) . . . 407
Steps for traditional (version 1.12 or earlier) . . . 408
Storage configuration options . . . 408
Global configuration options . . . 408
ONTAP configuration . . . 409
Element software configuration . . . 417
Known issues and limitations . . . 419
Upgrading Trident Docker Volume Plugin to 20.10 and later from older versions results in upgrade failure with the no such file or directory error. . . . 419
Volume names must be a minimum of 2 characters in length. . . . 420
Docker Swarm has certain behaviors that prevent Trident from supporting it with every storage and driver combination. . . . 420
If a FlexGroup is being provisioned, ONTAP does not provision a second FlexGroup if the second FlexGroup has one or more aggregates in common with the FlexGroup being provisioned. . . . 420
Best practices and recommendations . . . 421
Deployment . . . 421
Deploy to a dedicated namespace . . . 421
Use quotas and range limits to control storage consumption . . . 421
Storage configuration . . . 421
Platform overview . . . 421
ONTAP and Cloud Volumes ONTAP best practices . . . 421
SolidFire best practices . . . 426
Where to find more information? . . . 427

- Integrate Trident . . . 428
  - Driver selection and deployment . . . 428
  - Storage class design . . . 431
  - Virtual pool design . . . 432
  - Volume operations . . . 433
  - Metrics service . . . 436
- Data protection and disaster recovery . . . 437
  - Trident replication and recovery . . . 437
  - SVM replication and recovery . . . 438
  - Volume replication and recovery . . . 439
  - Snapshot data protection . . . 439
- Automating the failover of stateful applications with Trident . . . 439
  - Details about force detach . . . 439
  - Details about automated failover . . . 440
- Security . . . 445
  - Security . . . 445
  - Linux Unified Key Setup (LUKS) . . . 446
  - Kerberos in-flight encryption . . . 452

Protect applications with Trident Protect . . . 460
- Learn about Trident Protect . . . 460
  - What's next? . . . 460
- Install Trident Protect . . . 460
  - Trident Protect requirements . . . 460
  - Install and configure Trident Protect . . . 464
  - Install the Trident Protect CLI plugin . . . 467
  - Customize Trident Protect installation . . . 471
- Manage Trident Protect . . . 476
  - Manage Trident Protect authorization and access control . . . 476
  - Monitor Trident Protect resources . . . 483
  - Generate a Trident Protect support bundle . . . 488
  - Upgrade Trident Protect . . . 490
- Manage and protect applications . . . 492
  - Use Trident Protect AppVault objects to manage buckets . . . 492
  - Define an application for management with Trident Protect . . . 506
  - Protect applications using Trident Protect . . . 511
  - Restore applications . . . 524
  - Replicate applications using NetApp SnapMirror and Trident Protect . . . 542
  - Migrate applications using Trident Protect . . . 558
  - Manage Trident Protect execution hooks . . . 562
- Uninstall Trident Protect . . . 573

Trident and Trident Protect blogs . . . 575
- Trident blogs . . . 575
- Trident Protect blogs . . . 575

Knowledge and support . . . 576
- Frequently asked questions . . . 576

General questions . . . 576
Install and use Trident on a Kubernetes cluster . . . 576
Troubleshooting and support . . . 577
Upgrade Trident . . . 578
Manage backends and volumes . . . 579
Troubleshooting . . . 582
General troubleshooting . . . 583
Unsuccessful Trident deployment using the operator . . . 584
Unsuccessful Trident deployment using `tridentctl` . . . 586
Completely remove Trident and CRDs . . . 586
NVMe node unstaging failure with RWX raw block namespaces o Kubernetes 1.26 . . . 587
NFSv4.2 clients report "invalid argument" after upgrading ONTAP when when expecting "v4.2-xattrs" being enabled . . . 588
Support . . . 588
Trident support lifecycle . . . 588
Self-support . . . 589
Community support . . . 589
NetApp technical support . . . 589
For more information . . . 589
Reference . . . 590
Trident ports . . . 590
Overview . . . 590
Trident REST API . . . 592
When to use the REST API . . . 592
Using REST API . . . 592
Command-line options . . . 593
Logging . . . 593
Kubernetes . . . 593
Docker . . . 593
REST . . . 594
Kubernetes and Trident objects . . . 594
How do the objects interact with one another? . . . 594
Kubernetes `PersistentVolumeClaim` objects . . . 595
Kubernetes `PersistentVolume` objects . . . 596
Kubernetes `StorageClass` objects . . . 597
Kubernetes `VolumeSnapshotClass` objects . . . 600
Kubernetes `VolumeSnapshot` objects . . . 601
Kubernetes `VolumeSnapshotContent` objects . . . 601
Kubernetes `VolumeGroupSnapshotClass` objects . . . 602
Kubernetes `VolumeGroupSnapshot` objects . . . 602
Kubernetes `VolumeGroupSnapshotContent` objects . . . 602
Kubernetes `CustomResourceDefinition` objects . . . 603
Trident `StorageClass` objects . . . 603
Trident backend objects . . . 604

Trident `StoragePool` objects . . . 604
Trident `Volume` objects . . . 604
Trident `Snapshot` objects . . . 605
Trident `ResourceQuota` object . . . 606
Pod Security Standards (PSS) and Security Context Constraints (SCC) . . . 607
Required Kubernetes Security Context and Related Fields . . . 608
Pod Security Standards (PSS) . . . 608
Pod Security Policies (PSP) . . . 609
Security Context Constraints (SCC) . . . 610
Legal notices . . . 612
Copyright . . . 612
Trademarks . . . 612
Patents . . . 612
Privacy policy . . . 612
Open source . . . 612

Trident 26.02 documentation

Release notes

What's new

Release Notes provide information about new features, enhancements, and bug fixes in the latest version of NetApp Trident.

> The `tridentctl` binary for Linux that is provided in the installer zip file is the tested and supported version. Be aware that the `macos` binary provided in the `/extras` part of the zip file is not tested or supported.

What's new in 26.02.1

Learn what's new in NetApp Trident and Trident Protect, including enhancements, fixes, and deprecations.

Trident

Fixes

- **Kubernetes**:
- Fixed Google Cloud NetApp Volumes (GCNV) SAN behavior to support Flex Unified ZONAL pools for block (iSCSI) workloads. Both zonal and regional configurations are now supported.
- Fixed Google Cloud NetApp Volumes (GCNV) NAS UNIFIED pool handling by improving volume creation, validation, naming, and fallback volume lookup operations when `InternalID` is absent.
- Fixed an issue where iSCSI device resize operations could fail when available paths and portals did not match.
- Fixed an intermittent issue where `blkid` incorrectly identified the filesystem type for LUKS devices, preventing new volume creation.
- Fixed an issue where LUKS publish and stage operations at scale could block the Trident controller. Users are now expected to track LUKS passphrases associated with volumes.
- Fixed an issue where iSCSI multipath partitions could appear as ghost devices, preventing CSI `NodeUnstage` operations from succeeding.
- Fixed an NVMe/TCP namespace race condition during concurrent volume creation (Issue #1089).
- Fixed concurrent clone snapshot operations when snapshots were executed within the same second.
- Fixed an issue preventing KubeVirt VMs using `dataVolumes` from failing over with Trident Automated Workload Failover.
- Fixed an error that occurred when deleting an invalid `TridentMirrorRelationship` custom resource while using the Trident concurrent core.
- Fixed indentation in the `tridentactionmirrorupdates` Kubernetes CRD YAML definition (Issue #1120).
- Fixed the REST qtree asynchronous delete API call (Issue #1121).
- Fixed `ControllerPublish` to use the volume configuration filesystem type instead of an incorrect default filesystem type.
- Fixed controller startup logging to correctly report concurrent driver status.

- Fixed handling of suspended backends in concurrent Trident.
- Enhanced AWS ARN handling to support special regions.

Trident Protect

Fixes

Updated Trident Protect dependencies to resolve security vulnerabilities:

- gRPC-Go: CVE-2026-33186
- Go x/crypto SSH: CVE-2024-45337
- OpenTelemetry-Go: CVE-2026-24051
- Go runtime: CVE-2025-22871
- Additional dependencies: CVE-2025-23109

What's new in 26.02

Learn what's new in NetApp Trident and Trident Protect, including enhancements, fixes, and deprecations.

Trident

Enhancements

- **Kubernetes**:
  - Added general availability (GA) support for concurrency for ONTAP-NAS (NFS only), ONTAP-SAN (iSCSI, FCP, NVMe), and Google Cloud NetApp Volumes (GCNV) drivers, graduating this functionality from Tech Preview. See the Controller scalability documentation for more information.
  - Added support for volume autogrow in Trident based on custom, user-defined Trident AutoGrow policies. See the Automatic volume expansion documentation for more information.
  - Enhanced Trident node concurrency to improve the scalability of node operations for NVMe volumes. See the NVMe volumes documentation for more information.
  - Added support for GCNV NAS volume auto-tiering through PVC annotations (`tieringPolicy`, `tieringMinimumCoolingDays`), including pool selection and clone inheritance. See the Configure auto-tiering for Google Cloud NetApp Volumes documentation for more information.
  - Added support for GCNV SAN block (iSCSI) volumes using the `google-cloud-netapp-volumes-san` driver, including provisioning, per-node host group mapping for LUN access, and clone-from-volume operations. See the Google Cloud NetApp Volumes block configuration documentation for more information.
  - Added support for automatic backend configuration for Amazon FSx for NetApp ONTAP. When you create a StorageClass that includes the required parameters, Trident automatically creates the corresponding backend and VolumeSnapshotClass, if needed. See the Amazon FSx for NetApp ONTAP backend configuration documentation for more information.
  - Added support for different Microsoft Azure clouds, such as Azure Government and Azure China, and custom cloud configuration for Azure NetApp Files backends. See the Azure NetApp Files backend configuration documentation for more information.
  - Added support for Kubernetes 1.35. See the Requirements documentation for more information.

Experimental Enhancements

Not for use in production environments.

- **[Tech Preview]**: Added support for concurrency for ONTAP-NAS-Economy and ONTAP-SAN-Economy drivers.

Known issues

There is a known issue with the `csi-snapshotter` sidecar. In all Kubernetes versions, VolumeGroupSnapshots `v1beta1` prevents VolumeSnapshots from reaching the `ReadyToUse` state.

Two workarounds are available:

1. Delete the VolumeGroupSnapshots CRDs to disable VolumeGroupSnapshots, and then reinstall Trident.
2. Install VolumeGroupSnapshots `v1beta2` and `snapshot-controller` version 8.4.0 or later, and then reinstall Trident. VolumeGroupSnapshots do not function on Kubernetes versions earlier than v1.34.

Fixes

- **Kubernetes**:
  - Fixed an issue where unpublishing a read-only clone removed export policy rules from the source volume in ONTAP-NAS, ONTAP-NAS-Economy, and Google Cloud NetApp Volumes drivers Issue #1086.
  - Switched kubectl images to lightweight Alpine-based variants to prevent pull failures following Bitnami's public image deprecation Issue #1080.
  - Fixed preserving annotation of existing deployment during Trident upgrade Issue #1004.
  - Allow clone across different storage classes if both storage classes are pointing to the same backend Issue #1104.
  - Fixed node prep failures caused by timeouts in cloud environments with network latencies. Increased timeout values for cloud-based installations.
  - Fixed an issue in LUN creation that caused the filesystem type attribute to remain unset when the process entered a retry state.
  - Fixed REST API volume lookup to ignore volume state, preventing false negatives during volume queries.
  - Improved Trident controller efficiency for ontap-nas-economy driver when used at scale.
  - Set internalID during LUN import in ontap-san-economy driver.
  - Increased Azure Resource Graph query limits to handle more subnets.
  - Improved CSI and ONTAP clone split timeouts to avoid race conditions with some backup applications Issue #1098, Issue #1100.
  - Fixed suppression of LUKS error messages Issue #1069.
  - Fixed handling of stale LUKS mappers for both iSCSI and NVMe protocols. Enhanced cleanup logic prevents mount failures from orphaned device mappers.
- Fixed scale limitations for RWX NVMe volumes.
- Updated the OpenTelemetry-Go package to fix CVE-2026-24051.

Trident Protect

Enhancements

- Trident Protect now automatically disables protection schedules and cancels in-progress operations before an in-place restore, and re-enables them after the restore completes. To learn more, refer to Restore applications using Trident Protect.
- Added the `runImmediately` field to the schedule CR and `--run-immediately` CLI flag to trigger an immediate backup or snapshot upon schedule creation. To learn more, refer to Create a data protection schedule.
- Added support for specifying a custom name for the restored application using the `destinationApplicationName` field in the restore CR or the `--destination-app-name` CLI flag. To learn more, refer to Restore applications using Trident Protect.

Fixes

- Fixed restore failures caused by pods being created before their required service accounts were available.
- Fixed `Roles` and `RoleBindings` being skipped during application restores.
- Fixed the originating cluster name not being displayed in `tridentctl-protect get appvaultcontent` output despite being correctly configured.
- Fixed Kopia restore errors that were ignored due to missing `pipefail` handling.
- Fixed snapshot and backup failures caused by resource filters that excluded persistent volumes.
- Fixed incorrect PVC restoration in multi-namespace applications with identically named PVCs across namespaces, which could result in data loss.

What's new in 25.10

Learn what's new in Trident and Trident Protect, including enhancements, fixes, and deprecations.

Trident

Enhancements

- **Kubernetes**:
  - Added support for CSI Volume Group Snapshots with v1beta1 Volume Group Snapshot Kubernetes APIs for ONTAP-NAS NFS and ONTAP-SAN-Economy drivers, in addition to ONTAP-SAN (iSCSI and FC). See Work with volume group snapshots.
  - Added support for automated workload failover with force volume detach for the ONTAP-NAS and ONTAP-NAS-Economy (excluding SMB in both the NAS drivers), and the ONTAP-SAN and ONTAP-SAN-Economy drivers. See Automating the failover of stateful applications with Trident.
  - Enhanced Trident node concurrency for higher scalability on node operations for FCP volumes.
  - Added ONTAP AFX support for the ONTAP NAS driver. See ONTAP NAS configuration options and examples.
  - Added support for configuring CPU and memory resource requests and limits for Trident containers via TridentOrchestrator CR and Helm chart values. (Issue #1000, Issue #927, Issue #853, Issue #592, Issue #110).
  - Added FC support for ASAr2 personality. See ONTAP SAN configuration options and examples.

- Added an option to serve Prometheus metrics with HTTPS, instead of HTTP. See Monitor Trident.
- Added an option `--no-rename` when importing a volume to retain the original name but let Trident manage the volume's lifecycle. See Import volumes.
- Trident deployment now runs at the system-cluster-critical priority class.

- Added an option for Trident controller to use host networking via helm, operator, and tridentctl (Issue #858).
- Added manual QoS support to the ANF driver, making it production-ready in Trident 25.10; this experimental enhancement was introduced in Trident 25.06.

Experimental Enhancements

Not for use in production environments.

- **[Tech preview]**: Added support for concurrency for ONTAP-NAS (NFS only) and ONTAP-SAN (NVMe for unified ONTAP 9), in addition to the existing Tech preview for the ONTAP-SAN driver (iSCSI and FCP protocols in unified ONTAP 9).

Fixes

- **Kubernetes**:
  - Fixed the CSI node-driver-registrar container name inconsistency by standardizing Linux DaemonSet to node-driver-registrar to match Windows DaemonSet and container image naming.
  - Fixed an issue where export policies for legacy qtrees were not properly upgraded.
- **Openshift**:
  - Fixed Trident node pod not starting on Windows nodes in Openshift due to SCC having allowHostDirVolumePlugin set to false (Issue #950).
- Fixed Kubernetes API QPS not being set via Helm (Issue #975).
- Fixed inability to mount a Persistent Volume Claim (PVC) based on a snapshot of an NVMe based XFS filesystem PVC on the same Kubernetes node.
- Fixed UUID change issue after host/Docker restart in NDVP mode by adding unique/shared subsystem names per backend (e.g., netappdvp_subsystem).
- Fixed mount errors for iSCSI volumes during Trident upgrade from versions prior to 23.10 to 24.10 and above, resolving "invalid SANType" issue.
- Fixed issue where Trident backend state was not transitioning to online/offline without restarting the Trident controller.
- Fixed intermittent race condition causing slow PVC resize.
- Fixed snapshots not being cleaned up on volume clone failures.
- Fixed failure to unstage volume when its device path was changed by the kernel.
- Fixed failure to unstage volume due to LUKS device already closed.
- Fixed issue where slow storage operations were leading to ContextDeadline errors.
- Trident Operator will wait for configurable k8s-timeout to check Trident version.

Trident Protect

NetApp Trident Protect provides advanced application data management capabilities that enhance the functionality and availability of stateful Kubernetes applications backed by NetApp ONTAP storage systems

and the NetApp Trident CSI storage provisioner.

Enhancements

- Added annotations to control snapshot CR timeouts for schedule and backup CRs:
  - `protect.trident.netapp.io/snapshot-completion-timeout`
  - `protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout`
  - `protect.trident.netapp.io/volume-snapshots-created-timeout`

  See Supported backup and schedule annotations.
- Added an annotation to the schedule CR to configure PVC bind timeout, which will be used by the backup CR: `protect.trident.netapp.io/pvc-bind-timeout-sec`. See Supported backup and schedule annotations.
- Improved `tridentctl-protect` backup and snapshot listings with a new field to indicate execution hook failures.

Changes in 25.06.2

Trident

Fixes

- **Kubernetes**: Fixed critical issue where incorrect iSCSI devices were discovered when detaching volumes from Kubernetes nodes.

Changes in 25.06.1

Trident

> For customers using SolidFire, do not upgrade to 25.06.1 due to a known issue when unpublishing volumes. 25.06.2 will be released soon to address this issue.

Fixes

- **Kubernetes**:
  - Fixed an issue where NQNs were not checked before being unmapped from subsystems.
  - Fixed an issue where multiple attempts to close a LUKS device led to failures in detaching volumes.
  - Fixed iSCSI volume unstage when the device path has changed since its creation.
  - Block cloning of volumes across storage classes.
- **OpenShift**: Fixed an issue where iSCSI node prep failed with OCP 4.19.
- Increased the timeout when cloning a volume using SolidFire backends (Issue #1008).

Changes in 25.06

Trident

Enhancements

- **Kubernetes**:
  - Added support for CSI Volume Group Snapshots with `v1beta1` Volume Group Snapshot Kubernetes APIs for ONTAP-SAN iSCSI driver. See Work with volume group snapshots.

    VolumeGroupSnapshot is a beta feature in Kubernetes with beta APIs. Kubernetes 1.32 is the minimum version required for VolumeGroupSnapshot.

  - Added support for ONTAP ASA r2 for NVMe/TCP in addition to iSCSI. See ONTAP SAN configuration options and examples.
  - Added secure SMB support for ONTAP-NAS and ONTAP-NAS-Economy volumes. Active Directory users and groups may now be used with SMB volumes for enhanced security. See Enable secure SMB.
  - Enhanced Trident node concurrency for higher scalability on node operations for iSCSI volumes.
  - Added `--allow-discards` when opening LUKS volumes to allow discard/TRIM commands for space reclamation.
  - Enhanced performance when formatting LUKS-encrypted volumes.
  - Enhanced LUKS cleanup for failed but partially formatted LUKS devices.
  - Enhanced Trident node idempotency for NVMe volume attach and detach.
  - Added `internalID` field to the Trident volume config for ONTAP-SAN-Economy driver.
  - Added support for volume replication with SnapMirror for NVMe backends. See Replicate volumes using SnapMirror.

Experimental Enhancements

Not for use in production environments.

- [Tech Preview] Enabled concurrent Trident controller operations via the `--enable-concurrency` feature flag. This allows controller operations to run in parallel, improving performance for busy or large environments.

  This feature is experimental and currently supports limited parallel workflows with the ONTAP-SAN driver (iSCSI and FCP protocols).

- [Tech Preview] Added manual QOS support with the ANF driver.

Fixes

- **Kubernetes**:
  - Fixed an issue with CSI NodeExpandVolume where multipath devices could be left with incongruent sizes when underlying SCSI disk(s) are unavailable.
  - Fixed failure to clean up duplicate export policies for ONTAP-NAS and ONTAP-NAS-Economy drivers.
  - Fixed GCNV volumes defaulting to NFSv3 when `nfsMountOptions` is unset; now both NFSv3 and NFSv4 protocols are supported. If `nfsMountOptions` is not provided, the host's default NFS version (NFSv3 or NFSv4) are used.
  - Fixed deployment issue when installing Trident using Kustomize (Issue #831).

    - Fixed missing export policies for PVCs created from snapshots (Issue #1016).
    - Fixed issue where the ANF volume sizes are not automatically aligned to 1 GiB increments.
    - Fixed issue when using NFSv3 with Bottlerocket.
- Fixed issue with ONTAP-NAS-Economy volumes expanding up to 300 TB despite resize failures.
- Fixed issue where clone split operations were being done synchronously when using ONTAP REST API.

Deprecations:

- **Kubernetes**: Updated minimum supported Kubernetes to v1.27.

Trident Protect

NetApp Trident Protect provides advanced application data management capabilities that enhance the functionality and availability of stateful Kubernetes applications backed by NetApp ONTAP storage systems and the NetApp Trident CSI storage provisioner.

Enhancements

- Improved restore times, providing the option to do more frequent full backups.
- Improved granularity of application definition and selective restore with Group-Version-Kind (GVK) filtering.
- Efficient resync and reverse replication when using AppMirrorRelationship (AMR) with NetApp SnapMirror, to avoid full PVC replication.
- Added ability to use EKS Pod Identity to create AppVault buckets, removing the need to specify a secret with the bucket credentials for EKS clusters.
- Added the ability to skip restoring labels and annotations in the restore namespace, if needed.
- AppMirrorRelationship (AMR) will now check for source PVC expansion and perform the appropriate expansion on the destination PVC as needed.

Fixes

- Fixed bug where snapshot annotation values from previous snapshots were being applied to newer snapshots. All snapshot annotations are applied correctly now.
- Defined a secret for data mover encryption (Kopia / Restic) by default, if not defined..
- Added improved validation and error messages for S3 appvault creation.
- AppMirrorRelationship (AMR) now only replicates PVs in the Bound state, to avoid failed attempts.
- Fixed issue where errors were displayed when getting AppVaultContent on an AppVault with large number of backups.
- KubeVirt VMSnapshots are excluded from restore and failover operations to avoid failures.
- Fixed issue with Kopia where snapshots were being removed prematurely due to Kopia default retention schedule overriding what was set by the user in the schedule.

Changes in 25.02.1

Trident

Fixes

- **Kubernetes**:
  - Fixed an issue in the trident-operator where sidecar image names and versions were incorrectly populated when using a non-default image registry (Issue #983).
  - Fixed the issue where multipath sessions fail to recover during an ONTAP failover giveback (Issue #961).

Changes in 25.02

Beginning with Trident 25.02, the What's New summary provides details about enhancements, fixes, and deprecations for both Trident and Trident Protect releases.

Trident

Enhancements

- **Kubernetes**:
  - Added support for ONTAP ASA r2 for iSCSI.
  - Added support for force detach for ONTAP-NAS volumes during Non-Graceful Node Shutdown scenarios. New ONTAP-NAS volumes will now utilize per-volume export policies managed by Trident. Provided an upgrade path for existing volumes to transition to the new export policy model on unpublish without affecting active workloads.
  - Added cloneFromSnapshot annotation.
  - Added support for cross namespace volume cloning.
  - Enhanced iSCSI self-healing scan remediations to initiate rescans by exact host, channel, target and LUN ID.
  - Added support for Kubernetes 1.32.
- **OpenShift**:
  - Added support for automatic iSCSI node preparation for RHCOS on ROSA clusters.
  - Added support for OpenShift Virtualization for ONTAP drivers.
- Added Fibre Channel support on ONTAP-SAN driver.
- Added NVMe LUKS support.
- Switched to scratch image for all base images.
- Added iSCSI connection state discovery and logging when iSCSI sessions should be logged in but are not (Issue #961).
- Added support for SMB volumes with google-cloud-netapp-volumes driver.
- Added support to allow ONTAP volumes to skip recovery queue on deletion.
- Added support to override default images using SHAs instead of tags.
- Added image-pull-secrets flag to tridentctl installer.

Fixes

- **Kubernetes**:
  - Fixed missing node IP addresses from automatic export policies (Issue #965).

- Fixed automatic export policies switching to per volume policy prematurely for ONTAP-NAS-Economy.
- Fixed backend config credentials to support all available AWS ARN partitions (Issue #913).
- Added option to disable the auto configurator reconciliation in the Trident operator (Issue #924).
- Added securityContext for csi-resizer container (Issue #976).

Trident Protect

NetApp Trident Protect provides advanced application data management capabilities that enhance the functionality and availability of stateful Kubernetes applications backed by NetApp ONTAP storage systems and the NetApp Trident CSI storage provisioner.

Enhancements

- Added backup and restore support for KubeVirt / OpenShift Virtualization VMs for both volumeMode: File and volumeMode: Block (raw device) storage. This support is compatible with all Trident drivers, and enhances the existing protection features when replicating storage using NetApp SnapMirror with Trident Protect.
- Added the capability to control freeze behavior at application level for Kubevirt environments.
- Added support for configuring AutoSupport proxy connections.
- Added the ability to define a secret for data mover encryption (Kopia / Restic).
- Added the ability to manually run an execution hook.
- Added the ability to configure security context constraints (SCCs) during Trident Protect installation.
- Added support for configuring nodeSelector during Trident Protect installation.
- Added support for HTTP / HTTPS egress proxy for AppVault objects.
- Extended ResourceFilter to enable exclusion of cluster-scoped resources.
- Added support for the AWS session token in S3 AppVault credentials.
- Added support for resource collection after pre-snapshot execution hooks.

Fixes

- Improved the management of temporary volumes to skip the ONTAP volume recovery queue.
- SCC annotations are now restored to original values.
- Improved restore efficiency with support for parallel operations.
- Enhanced support for execution hook timeouts for larger applications.

Changes in 24.10.1

Enhancements

- **Kubernetes**: Added support for Kubernetes 1.32.
- Added iSCSI connection state discovery and logging when iSCSI sessions should be logged in but are not (Issue #961).

Fixes

- Fixed missing node IP addresses from automatic export policies (Issue #965).

- Fixed automatic export policies switching to per volume policy prematurely for ONTAP-NAS-Economy.
- Updated Trident and Trident-ASUP dependencies to address CVE-2024-45337 and CVE-2024-45310.
- Removed logouts for intermittently unhealthy non-CHAP portals during iSCSI self-healing (Issue #961).

Changes in 24.10

Enhancements

- Google Cloud NetApp Volumes driver is now generally available for NFS volumes and supports zone-aware provisioning.
- GCP Workload Identity will be used as Cloud Identity for Google Cloud NetApp Volumes with GKE.
- Added `formatOptions` configuration parameter to ONTAP-SAN and ONTAP-SAN-Economy drivers to allow users to specify LUN format options.
- Reduced Azure NetApp Files minimum volume size to 50 GiB. Azure new minimum size expected to be generally available in November.
- Added `denyNewVolumePools` configuration parameter to restrict ONTAP-NAS-Economy and ONTAP-SAN-Economy drivers to preexisting Flexvol pools.
- Added detection for the addition, removal, or renaming of aggregates from the SVM across all ONTAP drivers.
- Added 18 MiB overhead to LUKS LUNs to ensure reported PVC size is usable.
- Improved ONTAP-SAN and ONTAP-SAN-Economy node stage and unstage error handling to allow unstage to remove devices after a failed stage.
- Added a custom role generator allowing customers to create a minimalistic role for Trident in ONTAP.
- Added additional logging for troubleshooting `lsscsi` (Issue #792).

Kubernetes

- Added new Trident features for Kubernetes-native workflows:
  - Data protection
  - Data migration
  - Disaster recovery
  - Application mobility

  Learn more about Trident Protect.
- Added a new flag `--k8s-api-qps` to installers to set the QPS value used by Trident to communicate with the Kubernetes API server.
- Added `--node-prep` flag to installers for automatic management of storage protocol dependencies on Kubernetes cluster nodes. Tested and verified compatibility with Amazon Linux 2023 iSCSI storage protocol
- Added support for force detach for ONTAP-NAS-Economy volumes during Non-Graceful Node Shutdown scenarios.
- New ONTAP-NAS-Economy NFS volumes will use per-qtree export policies when using `autoExportPolicy` backend option. Qtrees will only be mapped to node restrictive export policies at time of publish to improve access control and security. Existing qtrees will be switched to the new export policy

model when Trident unpublishes the volume from all nodes to do so without impacting active workloads.

- Added support for Kubernetes 1.31.

Experimental Enhancements

- Added tech preview for Fibre Channel support on ONTAP-SAN driver.

Fixes

- **Kubernetes**:
  - Fixed Rancher admission webhook preventing Trident Helm installations (Issue #839).
  - Fixed Affinity key in helm chart values (Issue #898).
  - Fixed tridentControllerPluginNodeSelector/tridentNodePluginNodeSelector won't work with "true" value (Issue #899).
  - Deleted ephemeral snapshots created during cloning (Issue #901).
- Added support for Windows Server 2019.
- Fixed `go mod tidy`in Trident repo (Issue #767).

Deprecations

- **Kubernetes:**
  - Updated minimum supported Kubernetes to 1.25.
  - Removed support for POD Security Policy.

Product rebranding

Beginning with the 24.10 release, Astra Trident is rebranded to Trident (Netapp Trident). This rebranding does not affect any features, platforms supported, or interoperability for Trident.

Changes in 24.06

Enhancements

- **IMPORTANT**: The `limitVolumeSize` parameter now limits qtree/LUN sizes in the ONTAP economy drivers. Use the new `limitVolumePoolSize` parameter to control Flexvol sizes in those drivers. (Issue #341).
- Added ability for iSCSI self-healing to initiate SCSI scans by exact LUN ID if deprecated igroups are in use (Issue #883).
- Added support for volume clone and resize operations to be allowed even when the backend is in suspended mode.
- Added ability for user-configured log settings for the Trident controller to be propagated to Trident node pods.
- Added support in Trident to use REST by default instead of ONTAPI (ZAPI) for ONTAP versions 9.15.1 and later.
- Added support for custom volume names and metadata on the ONTAP storage backends for new persistent volumes.
- Enhanced the `azure-netapp-files` (ANF) driver to automatically enable the snapshot directory by

default when the NFS mount options are set to use NFS version 4.x.

- Added Bottlerocket support for NFS volumes.
- Added technical preview support for Google Cloud NetApp Volumes.

Kubernetes

- Added support for Kubernetes 1.30.
- Added ability for Trident DaemonSet to clean zombie mounts and residual tracking files at startup (Issue #883).
- Added PVC annotation `trident.netapp.io/luksEncryption` for dynamically importing LUKS volumes (Issue #849).
- Added topology awareness to ANF driver.
- Added support for Windows Server 2022 nodes.

Fixes

- Fixed Trident installation failures due to stale transactions.
- Fixed tridentctl to ignore warning messages from Kubernetes (Issue #892).
- Changed Trident controller `SecurityContextConstraint` priority to `0` (Issue #887).
- ONTAP drivers now accept volume sizes below 20 MiB (Issue[#885).
- Fixed Trident to prevent shrinking of FlexVol volumes during resize operation for the ONTAP-SAN driver.
- Fixed ANF volume import failure with NFS v4.1.

Changes in 24.02

Enhancements

- Added support for Cloud Identity.
  - AKS with ANF - Azure Workload Identity will be used as Cloud identity.
  - EKS with FSxN - AWS IAM role will be used as Cloud identity.
- Added support to install Trident as an add-on on EKS cluster from EKS console.
- Added ability to configure and disable iSCSI self-healing (Issue #864).
- Added Amazon FSx personality to ONTAP drivers to enable integration with AWS IAM and SecretsManager, and to enable Trident to delete FSx volumes with backups (Issue #453).

Kubernetes

- Added support for Kubernetes 1.29.

Fixes

- Fixed ACP warning messages, when ACP is not enabled (Issue #866).
- Added a 10-second delay before performing a clone split during snapshot delete for ONTAP drivers, when a clone is associated with the snapshot.

Deprecations

- Removed in-toto attestations framework from multi-platform image manifests.

Changes in 23.10

Fixes

- Fixed volume expansion if a new requested size is smaller than the total volume size for ontap-nas and ontap-nas-flexgroup storage drivers (Issue #834).
- Fixed volume size to display only usable size of the volume during import for ontap-nas and ontap-nas-flexgroup storage drivers (Issue #722).
- Fixed FlexVol name conversion for ONTAP-NAS-Economy.
- Fixed Trident initialization issue on a windows node when node is rebooted.

Enhancements

Kubernetes

Added support for Kubernetes 1.28.

Trident

- Added support for using Azure Managed Identities (AMI) with azure-netapp-files storage driver.
- Added support for NVMe over TCP for the ONTAP-SAN driver.
- Added ability to pause the provisioning of a volume when backend is set to suspended state by user (Issue #558).

Changes in 23.07.1

Kubernetes: Fixed daemonset deletion to support zero-downtime upgrades (Issue #740).

Changes in 23.07

Fixes

Kubernetes

- Fixed Trident upgrade to disregard old pods stuck in terminating state (Issue #740).
- Added toleration to "transient-trident-version-pod" definition (Issue #795).

Trident

- Fixed ONTAPI (ZAPI) requests to ensure LUN serial numbers are queried when getting LUN attributes to identify and fix ghost iSCSI devices during Node Staging operations.
- Fixed error handling in storage driver code (Issue #816).
- Fixed quota resize when using ONTAP drivers with use-rest=true.
- Fixed LUN clone creation in ontap-san-economy.
- Revert publish info field from `rawDevicePath` to `devicePath`; added logic to populate and recover (in

some cases) `devicePath` field.

Enhancements

Kubernetes

- Added support for importing pre-provisioned snapshots.
- Minimized deployment and daemonset linux permissions (Issue #817).

Trident

- No longer reporting the state field for "online" volumes and snapshots.
- Updates the backend state if the ONTAP backend is offline (Issues #801, #543).
- LUN Serial Number is always retrieved and published during the ControllerVolumePublish workflow.
- Added additional logic to verify iSCSI multipath device serial number and size.
- Additional verification for iSCSI volumes to ensure correct multipath device is unstaged.

Experimental Enhancement

Added tech preview support for NVMe over TCP for the ONTAP-SAN driver.

Documentation

Many organizational and formatting improvements have been made.

Deprecations

Kubernetes

- Removed support for v1beta1 snapshots.
- Removed support for pre-CSI volumes and storage classes.
- Updated minimum supported Kubernetes to 1.22.

Changes in 23.04

> Force volume detach for ONTAP-SAN-* volumes is supported only with Kubernetes versions with the Non-Graceful Node Shutdown feature gate enabled. Force detach must be enabled at install time using the `--enable-force-detach` Trident installer flag.

Fixes

- Fixed Trident Operator to use IPv6 localhost for installation when specified in spec.
- Fixed Trident Operator cluster role permissions to be in sync with the bundle permissions (Issue #799).
- Fixed issue with attaching raw block volume on multiple nodes in RWX mode.
- Fixed FlexGroup cloning support and volume import for SMB volumes.
- Fixed issue where Trident controller could not shut down immediately (Issue #811).
- Added fix to list all igroup names associated with a specified LUN provisioned with ontap-san-* drivers.

- Added a fix to allow external processes to run to completion.
- Fixed compilation error for s390 architecture (Issue #537).
- Fixed incorrect logging level during volume mount operations (Issue #781).
- Fixed potential type assertion error (Issue #802).

Enhancements

- Kubernetes:
  - Added support for Kubernetes 1.27.
  - Added support for importing LUKS volumes.
  - Added support for ReadWriteOncePod PVC access mode.
  - Added support for force detach for ONTAP-SAN-* volumes during Non-Graceful Node Shutdown scenarios.
  - All ONTAP-SAN-* volumes will now use per-node igroups. LUNs will only be mapped to igroups while actively published to those nodes to improve our security posture. Existing volumes will be opportunistically switched to the new igroup scheme when Trident determines it is safe to do so without impacting active workloads (Issue #758).
  - Improved Trident security by cleaning up unused Trident-managed igroups from ONTAP-SAN-* backends.
- Added support for SMB volumes with Amazon FSx to the ontap-nas-economy and ontap-nas-flexgroup storage drivers.
- Added support for SMB shares with the ontap-nas, ontap-nas-economy and ontap-nas-flexgroup storage drivers.
- Added support for arm64 nodes (Issue #732).
- Improved Trident shutdown procedure by deactivating API servers first (Issue #811).
- Added cross-platform build support for Windows and arm64 hosts to Makefile; see BUILD.md.

Deprecations

Kubernetes: Backend-scoped igroups will no longer be created when configuring ontap-san and ontap-san-economy drivers (Issue #758).

Changes in 23.01.1

Fixes

- Fixed Trident Operator to use IPv6 localhost for installation when specified in spec.
- Fixed Trident Operator cluster role permissions to be in sync with the bundle permissions Issue #799.
- Added a fix to allow external processes to run to completion.
- Fixed issue with attaching raw block volume on multiple nodes in RWX mode.
- Fixed FlexGroup cloning support and volume import for SMB volumes.

Changes in 23.01

Kubernetes 1.27 is now supported in Trident. Please upgrade Trident prior to upgrading Kubernetes.

Fixes

- Kubernetes: Added options to exclude Pod Security Policy creation to fix Trident installations via Helm (Issues #783, #794).

Enhancements

Kubernetes

- Added support for Kubernetes 1.26.
- Improved overall Trident RBAC resource utilization (Issue #757).
- Added automation to detect and fix broken or stale iSCSI sessions on host nodes.
- Added support for expanding LUKS encrypted volumes.
- Kubernetes: Added credential rotation support for LUKS encrypted volumes.

Trident

- Added support for SMB volumes with Amazon FSx for NetApp ONTAP to the ontap-nas storage driver.
- Added support for NTFS permissions when using SMB volumes.
- Added support for storage pools for GCP volumes with CVS service level.
- Added support for optional use of flexgroupAggregateList when creating FlexGroups with the ontap-nas-flexgroup storage driver.
- Improved performance for the ontap-nas-economy storage driver when managing multiple FlexVol volumes
- Enabled dataLIF updates for all ONTAP NAS storage drivers.
- Updated the Trident Deployment and DaemonSet naming convention to reflect the host node OS.

Deprecations

- Kubernetes: Updated minimum supported Kubernetes to 1.21.
- DataLIFs should no longer be specified when configuring `ontap-san` or `ontap-san-economy` drivers.

Changes in 22.10

You must read the following critical information before upgrading to Trident 22.10.

Critical information about Trident 22.10

- Kubernetes 1.25 is now supported in Trident. You must upgrade Trident to 22.10 prior to upgrading to Kubernetes 1.25.
- Trident now strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Fixes

- Fixed issue specific to ONTAP backend created using `credentials` field failing to come online during 22.07.0 upgrade (Issue #759).
- **Docker:** Fixed an issue causing the Docker volume plugin to fail to start in some environments (Issue #548 and Issue #760).
- Fixed SLM issue specific to ONTAP SAN backends to ensure only subset of dataLIFs belonging to reporting nodes are published.
- Fixed performance issue where unnecessary scans for iSCSI LUNs happened when attaching a volume.
- Removed granular retries within the Trident iSCSI workflow to fail fast and reduce external retry intervals.
- Fixed issue where an error was returned when flushing an iSCSI device when the corresponding multipath device was already flushed.

Enhancements

- Kubernetes:
  - Added support for Kubernetes 1.25. You must upgrade Trident to 22.10 prior to upgrading to Kubernetes 1.25.
  - Added a separate ServiceAccount, ClusterRole, and ClusterRoleBinding for the Trident Deployment and DaemonSet to allow future permissions enhancements.
  - Added support for cross-namespace volume sharing.
- All Trident `ontap-*` storage drivers now work with the ONTAP REST API.
- Added new operator yaml (`bundle_post_1_25.yaml`) without a `PodSecurityPolicy` to support Kubernetes 1.25.
- Added support for LUKS-encrypted volumes for `ontap-san` and `ontap-san-economy` storage drivers.
- Added support for Windows Server 2019 nodes.
- Added support for SMB volumes on Windows nodes through the `azure-netapp-files` storage driver.
- Automatic MetroCluster switchover detection for ONTAP drivers is now generally available.

Deprecations

- **Kubernetes:** Updated minimum supported Kubernetes to 1.20.
- Removed Astra Data Store (ADS) driver.
- Removed support for `yes` and `smart` options for `find_multipaths` when configuring worker node multipathing for iSCSI.

Changes in 22.07

Fixes

Kubernetes

- Fixed issue to handle boolean and number values for node selector when configuring Trident with Helm or the Trident Operator. (GitHub issue #700)
- Fixed issue in handling errors from non-CHAP path, so that kubelet will retry if it fails. GitHub issue #736)

Enhancements

- Transition from k8s.gcr.io to registry.k8s.io as default registry for CSI images
- ONTAP-SAN volumes will now use per-node igroups and only map LUNs to igroups while actively published to those nodes to improve our security posture. Existing volumes will be opportunistically switched to the new igroup scheme when Trident determines it is safe to do so without impacting active workloads.
- Included a ResourceQuota with Trident installations to ensure Trident DaemonSet is scheduled when PriorityClass consumption is limited by default.
- Added support for Network Features to Azure NetApp Files driver. (GitHub issue #717)
- Added tech preview automatic MetroCluster switchover detection to ONTAP drivers. (GitHub issue #228)

Deprecations

- **Kubernetes:** Updated minimum supported Kubernetes to 1.19.
- Backend config no longer allows multiple authentication types in single config.

Removals

- AWS CVS driver (deprecated since 22.04) has been removed.
- Kubernetes
  - Removed unnecessary SYS_ADMIN capability from node pods.
  - Reduces nodeprep down to simple host info and active service discovery to do a best-effort confirmation that NFS/iSCSI services are available on worker nodes.

Documentation

A new Pod Security Standards (PSS) section has been added detailing permissions enabled by Trident on installation.

Changes in 22.04

NetApp is continually improving and enhancing its products and services. Here are some of the latest features in Trident. For previous releases, Refer to Earlier versions of documentation.

> If you are upgrading from any previous Trident release and use Azure NetApp Files, the `location` config parameter is now a mandatory, singleton field.

Fixes

- Improved parsing of iSCSI initiator names. (GitHub issue #681)
- Fixed issue where CSI storage class parameters weren't allowed. (GitHub issue #598)
- Fixed duplicate key declaration in Trident CRD. (GitHub issue #671)
- Fixed inaccurate CSI Snapshot logs. (GitHub issue #629))
- Fixed issue with unpublishing volumes on deleted nodes. (GitHub issue #691)
- Added handling of filesystem inconsistencies on block devices. (GitHub issue #656)
- Fixed issue pulling auto-support images when setting the `imageRegistry` flag during installation. (GitHub

issue #715)

- Fixed issue where Azure NetApp Files driver failed to clone a volume with multiple export rules.

Enhancements

- Inbound connections to Trident's secure endpoints now require a minimum of TLS 1.3. (GitHub issue #698)
- Trident now adds HSTS headers to responses from its secure endpoints.
- Trident now attempts to enable the Azure NetApp Files unix permissions feature automatically.
- **Kubernetes**: Trident daemonset now runs at system-node-critical priority class. (GitHub issue #694)

Removals

E-Series driver (disabled since 20.07) has been removed.

Changes in 22.01.1

Fixes

- Fixed issue with unpublishing volumes on deleted nodes. (GitHub issue #691)
- Fixed panic when accessing nil fields for aggregate space in ONTAP API responses.

Changes in 22.01.0

Fixes

- **Kubernetes:** Increase node registration backoff retry time for large clusters.
- Fixed issue where azure-netapp-files driver could be confused by multiple resources with the same name.
- ONTAP SAN IPv6 DataLIFs now work if specified with brackets.
- Fixed issue where attempting to import an already imported volume returns EOF leaving PVC in pending state. (GitHub issue #489)
- Fixed issue when Trident performance slows down when > 32 snapshots are created on a SolidFire volume.
- Replaced SHA-1 with SHA-256 in SSL certificate creation.
- Fixed Azure NetApp Files driver to allow duplicate resource names and limit operations to a single location.
- Fixed Azure NetApp Files driver to allow duplicate resource names and limit operations to a single location.

Enhancements

- Kubernetes enhancements:
  - Added support for Kubernetes 1.23.
  - Add scheduling options for Trident pods when installed via Trident Operator or Helm. (GitHub issue #651)
- Allow cross-region volumes in GCP driver. (GitHub issue #633)
- Added support for 'unixPermissions' option to Azure NetApp Files volumes. (GitHub issue #666)

Deprecations

Trident REST interface can listen and serve only at 127.0.0.1 or [::1] addresses

Changes in 21.10.1

The v21.10.0 release has an issue that can put the Trident controller into a CrashLoopBackOff state when a node is removed and then added back to the Kubernetes cluster. This issue is fixed in v21.10.1 (GitHub issue 669).

Fixes

- Fixed potential race condition when importing a volume on a GCP CVS backend resulting in failure to import.
- Fixed an issue that can put the Trident controller into a CrashLoopBackOff state when a node is removed and then added back to the Kubernetes cluster (GitHub issue 669).
- Fixed issue where SVMs were no longer discovered if no SVM name was specified (GitHub issue 612).

Changes in 21.10.0

Fixes

- Fixed issue where clones of XFS volumes could not be mounted on the same node as the source volume (GitHub issue 514).
- Fixed issue where Trident logged a fatal error on shutdown (GitHub issue 597).
- Kubernetes-related fixes:
  - Return a volume's used space as the minimum restoreSize when creating snapshots with `ontap-nas` and `ontap-nas-flexgroup` drivers (GitHub issue 645).
  - Fixed issue where `Failed to expand filesystem` error was logged after volume resize (GitHub issue 560).
  - Fixed issue where a pod could get stuck in `Terminating` state (GitHub issue 572).
  - Fixed the case where an `ontap-san-economy` FlexVol might be full of snapshot LUNs (GitHub issue 533).
  - Fixed custom YAML installer issue with different image (GitHub issue 613).
  - Fixed snapshot size calculation (GitHub issue 611).
  - Fixed issue where all Trident installers could identify plain Kubernetes as OpenShift (GitHub issue 639).
  - Fixed the Trident operator to stop reconciliation if the Kubernetes API server is unreachable (GitHub issue 599).

Enhancements

- Added support for `unixPermissions` option to GCP-CVS Performance volumes.
- Added support for scale-optimized CVS volumes in GCP in the range 600 GiB to 1 TiB.
- Kubernetes-related enhancements:

- Added support for Kubernetes 1.22.
- Enabled the Trident operator and Helm chart to work with Kubernetes 1.22 (GitHub issue 628).
- Added operator image to `tridentctl` images command (GitHub issue 570).

Experimental enhancements

- Added support for volume replication in the `ontap-san` driver.
- Added **tech preview** REST support for the `ontap-nas-flexgroup`, `ontap-san`, and `ontap-nas-economy` drivers.

Known issues

Known issues identify problems that might prevent you from using the product successfully.

- When upgrading a Kubernetes cluster from 1.24 to 1.25 or later that has Trident installed, you must update values.yaml to set `excludePodSecurityPolicy` to `true` or add `--set excludePodSecurityPolicy=true` to the `helm upgrade` command before you can upgrade the cluster.
- Trident now enforces a blank `fsType` (`fsType=""`) for volumes that do not have the `fsType` specified in their StorageClass. When working with Kubernetes 1.17 or later, Trident supports providing a blank `fsType` for NFS volumes. For iSCSI volumes, you are required to set the `fsType` on your StorageClass when enforcing an `fsGroup` using a Security Context.
- When using a backend across multiple Trident instances, each backend configuration file should have a different `storagePrefix` value for ONTAP backends or use a different `TenantName` for SolidFire backends. Trident cannot detect volumes that other instances of Trident have created. Attempting to create an existing volume on either ONTAP or SolidFire backends succeeds, because Trident treats volume creation as an idempotent operation. If `storagePrefix` or `TenantName` do not differ, there might be name collisions for volumes created on the same backend.
- When installing Trident (using `tridentctl` or the Trident Operator) and using `tridentctl` to manage Trident, you should ensure the `KUBECONFIG` environment variable is set. This is necessary to indicate the Kubernetes cluster that `tridentctl` should work against. When working with multiple Kubernetes environments, you should ensure that the `KUBECONFIG` file is sourced accurately.
- To perform online space reclamation for iSCSI PVs, the underlying OS on the worker node might require mount options to be passed to the volume. This is true for RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) instances, which require the `discard` mount option; ensure that the discard mountOption is included in your `StorageClass` to support online block discard.
- If you have more than one instance of Trident per Kubernetes cluster, Trident cannot communicate with other instances and cannot discover other volumes that they have created, which leads to unexpected and incorrect behavior if more than one instance runs within a cluster. There should be only one instance of Trident per Kubernetes cluster.
- If Trident-based `StorageClass` objects are deleted from Kubernetes while Trident is offline, Trident does not remove the corresponding storage classes from its database when it comes back online. You should delete these storage classes using `tridentctl` or the REST API.
- If a user deletes a PV provisioned by Trident before deleting the corresponding PVC, Trident does not automatically delete the backing volume. You should remove the volume via `tridentctl` or the REST API.
- ONTAP cannot concurrently provision more than one FlexGroup at a time unless the set of aggregates are unique to each provisioning request.

- When using Trident over IPv6, you should specify `managementLIF` and `dataLIF` in the backend definition within square brackets. For example, `[fd20:8b1e:b258:2000:f816:3eff:feec:0]`.

  > You cannot specify `dataLIF` on an ONTAP SAN backend. Trident discovers all available iSCSI LIFs and uses them to establish the multipath session.

- If using the `solidfire-san` driver with OpenShift 4.5, ensure that the underlying worker nodes use MD5 as the CHAP authentication algorithm. Secure FIPS-compliant CHAP algorithms SHA1, SHA-256, and SHA3-256 are available with Element 12.7.

Find more information

- Trident GitHub
- Trident blogs

Earlier versions of documentation

If you aren't running Trident 26.02, the documentation for previous releases is available based on the Trident support lifecycle.

- Trident 25.10
- Trident 25.06
- Trident 25.02
- Trident 24.10
- Trident 24.06
- Trident 24.02
- Trident 23.10
- Trident 23.07
- Trident 23.04

NetApp Trident support for ONTAP ASA r2 storage systems

NetApp Trident 25.02 and later supports NetApp ASA r2 systems as storage backends. Refer to ASA r2 systems for more information.

ASA r2 systems require the `ontap-san` driver. Trident does not support the `ontap-san-economy` driver for ASA r2 systems.

When you specify `ontap-san` as the `storageDriverName` in the backend configuration, Trident automatically detects the ASA r2 storage system.

Trident provides limited data protection for ASA r2 systems with Trident protect.

Supported SAN protocols depend on your Trident version:

- Trident 25.02 and later supports iSCSI.

- Trident 25.06 and later supports NVMe/TCP in addition to iSCSI.
- Trident 25.10 and later supports FC in addition to iSCSI and NVMe/TCP.

You must assign at least one aggregate to the storage virtual machine (SVM) for ONTAP backend storage. Refer to Assigning aggregates to SVMs in ASA r2 systems for instructions.

Supported operations

- Provisioning persistent volumes (PVs)
- Dynamic volume provisioning
- Creating and deleting volumes
- Cloning volumes
- Expanding volumes
- Managing storage classes

Unsupported operations

- LUKS encryption
- SnapMirror volume replication
- Limiting aggregate usage
- Space reservation mode
- Snapshots
- Tiering

For more information, refer to ONTAP SAN configuration options and examples .

Known issues

Known issues identify problems that might prevent you from using this release of the product successfully.

The following known issues affect the current release:

VolumeSnapshots not reaching ReadyToUse state

There is a known issue with the `csi-snapshotter` sidecar. In all Kubernetes versions, VolumeGroupSnapshots `v1beta1` prevents VolumeSnapshots from reaching the `ReadyToUse` state.

Two workarounds are available:

1. Delete the VolumeGroupSnapshots CRDs to disable VolumeGroupSnapshots, and then reinstall Trident.
2. Install VolumeGroupSnapshots `v1beta2` and `snapshot-controller` version 8.4.0 or later, and then reinstall Trident. VolumeGroupSnapshots do not function on Kubernetes versions earlier than v1.34.

Restoring Restic backups of large files can fail

When restoring 30GB or larger files from an Amazon S3 backup that was made using Restic, the restore operation can fail. As a workaround, back up the data using Kopia as the data mover (Kopia is the default data mover for backups). Refer to Protect applications using Trident Protect for instructions.

Get started

Learn about Trident

Learn about Trident

Trident is a fully-supported open source project maintained by NetApp. It has been designed to help you meet your containerized application's persistence demands using industry-standard interfaces, such as the Container Storage Interface (CSI).

What is Trident?

Netapp Trident enables consumption and management of storage resources across all popular NetApp storage platforms, in the public cloud or on premises, including on-premises ONTAP clusters (AFF, FAS, and ASA), ONTAP Select, Cloud Volumes ONTAP, Element software (NetApp HCI, SolidFire), Azure NetApp Files, Amazon FSx for NetApp ONTAP, and Google Cloud NetApp Volumes.

Trident is a Container Storage Interface (CSI) compliant dynamic storage orchestrator that natively integrates with Kubernetes. Trident runs as a single Controller Pod plus a Node Pod on each worker node in the cluster. Refer to Trident architecture for details.

Trident also provides direct integration with the Docker ecosystem for NetApp storage platforms. The NetApp Docker Volume Plugin (nDVP) supports the provisioning and management of storage resources from the storage platform to Docker hosts. Refer to Deploy Trident for Docker for details.

If this is your first time using Kubernetes, you should familiarize yourself with the Kubernetes concepts and tools.

Supported Kubernetes platforms

Trident is supported on a range of Kubernetes distributions and platforms.

Supported platforms include:
* Upstream Kubernetes
* Red Hat OpenShift
* SUSE Harvester 1.7.0 (ONTAP iSCSI)

Kubernetes integration with NetApp products

The NetApp portfolio of storage products integrates with many aspects of a Kubernetes cluster, providing advanced data management capabilities, which enhance the functionality, capability, performance, and availability of the Kubernetes deployment.

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP is a fully managed AWS service that lets you launch and run file systems powered by the NetApp ONTAP storage operating system.

Azure NetApp Files

Azure NetApp Files is an enterprise-grade Azure file share service, powered by NetApp. You can run your most demanding file-based workloads in Azure natively, with the performance and rich data management you expect from NetApp.

Cloud Volumes ONTAP

Cloud Volumes ONTAP is a software-only storage appliance that runs the ONTAP data management software in the cloud.

Google Cloud NetApp Volumes

Google Cloud NetApp Volumes is a fully managed file storage service in Google Cloud that provides high-performance, enterprise-grade file storage.

Element software

Element enables the storage administrator to consolidate workloads by guaranteeing performance and enabling a simplified and streamlined storage footprint.

NetApp HCI

NetApp HCI simplifies the management and scale of the datacenter by automating routine tasks and enabling infrastructure administrators to focus on more important functions.

Trident can provision and manage storage devices for containerized applications directly against the underlying NetApp HCI storage platform.

NetApp ONTAP

NetApp ONTAP is the NetApp multiprotocol, unified storage operating system that provides advanced data management capabilities for any application.

ONTAP systems have all-flash, hybrid, or all-HDD configurations and offer many different deployment models: on-premises FAS, AFA, and ASA clusters, ONTAP Select, and Cloud Volumes ONTAP. Trident supports these ONTAP deployment models.

Trident architecture

Trident runs as a single Controller Pod plus a Node Pod on each worker node in the cluster. The node pod must be running on any host where you want to potentially mount a Trident volume.

Understanding controller pods and node pods

Trident deploys as a single Trident Controller Pod and one or more Trident Node Pods on the Kubernetes

cluster and uses standard Kubernetes *CSI Sidecar Containers* to simplify the deployment of CSI plugins. Kubernetes CSI Sidecar Containers are maintained by the Kubernetes Storage community.

Kubernetes node selectors and tolerations and taints are used to constrain a pod to run on a specific or preferred node. You can configure node selectors and tolerations for controller and node pods during Trident installation.

- The controller plugin handles volume provisioning and management, such as snapshots and resizing.
- The node plugin handles attaching the storage to the node.

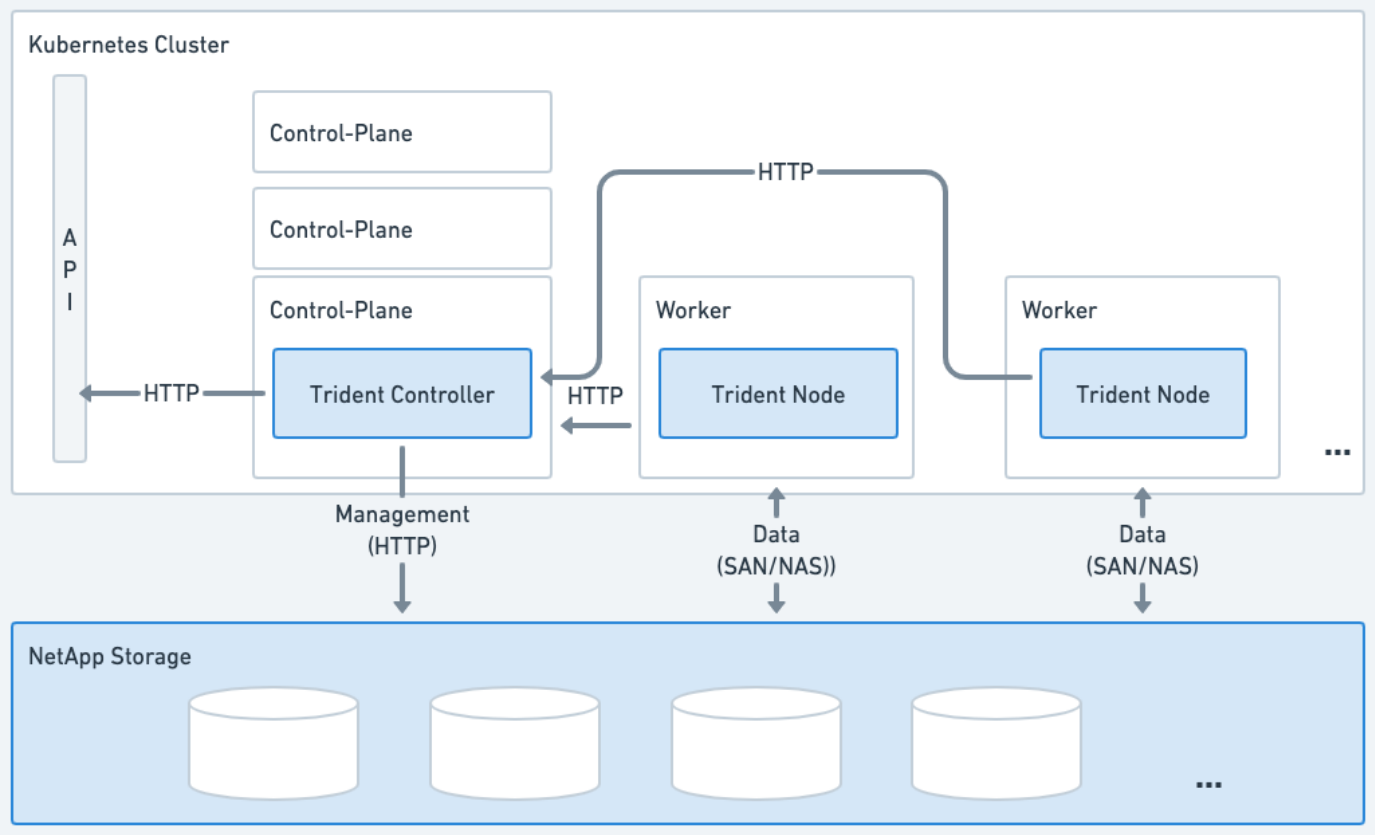


Figure 1. Trident deployed on the Kubernetes cluster

Trident Controller Pod

The Trident Controller Pod is a single Pod running the CSI Controller plugin.

- Responsible for provisioning and managing volumes in NetApp storage
- Managed by a Kubernetes Deployment
- Can run on the control-plane or worker nodes, depending on installation parameters.

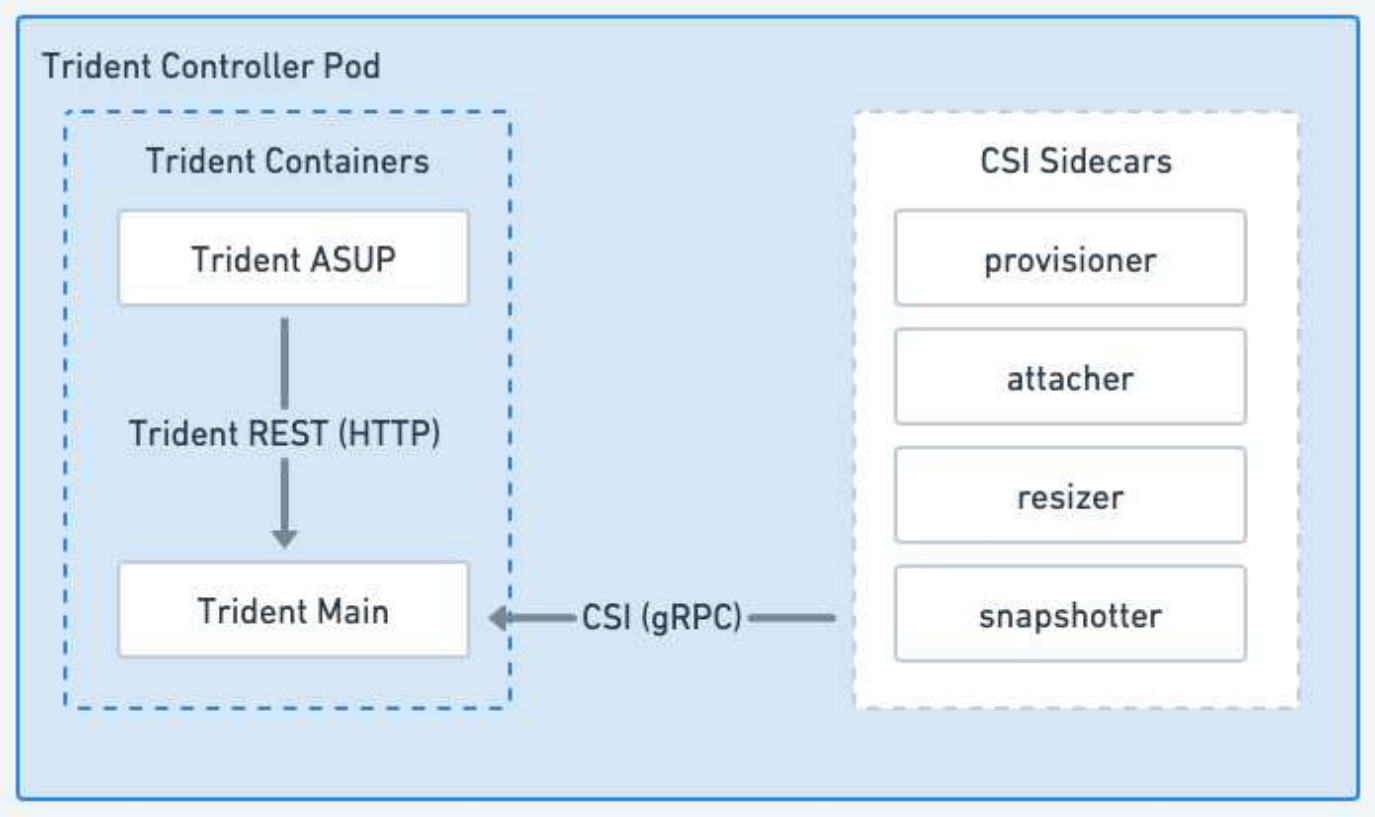


Figure 2. Trident Controller Pod diagram

Trident Node Pods

Trident Node Pods are privileged Pods running the CSI Node plugin.

- Responsible for mounting and unmounting storage for Pods running on the host
- Managed by a Kubernetes DaemonSet
- Must run on any node that will mount NetApp storage

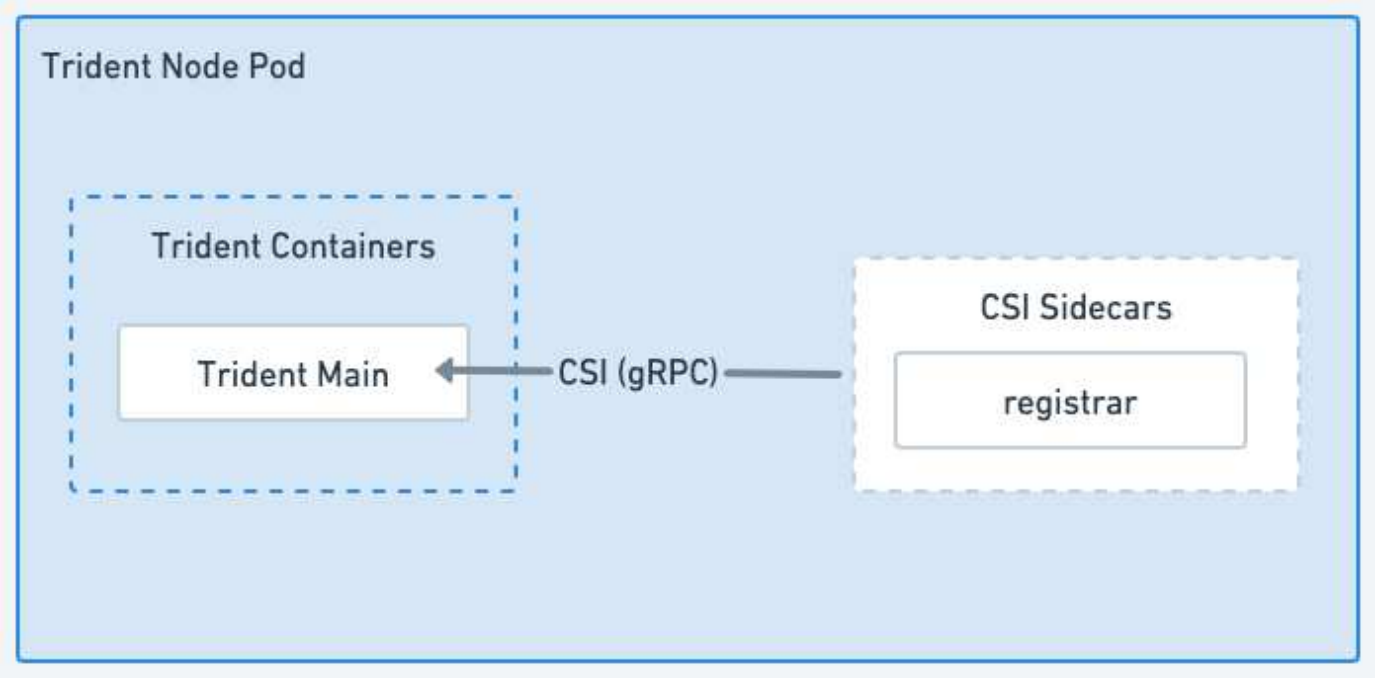


Figure 3. Trident Node Pod diagram

Supported Kubernetes cluster architectures

Trident is supported with the following Kubernetes architectures:

Kubernetes cluster architectures	Supported	Default install
Single master, compute	Yes	Yes
Multiple master, compute	Yes	Yes
Master, `etcd`, compute	Yes	Yes
Master, infrastructure, compute	Yes	Yes

Concepts

Provisioning

Provisioning in Trident has two primary phases. The first phase associates a storage class with the set of suitable backend storage pools and occurs as a necessary preparation before provisioning. The second phase includes the volume creation itself and requires choosing a storage pool from those associated with the pending volume's storage class.

Storage class association

Associating backend storage pools with a storage class relies on both the storage class's requested attributes and its `storagePools`, `additionalStoragePools`, and `excludeStoragePools` lists. When you create a storage class, Trident compares the attributes and pools offered by each of its backends to those requested by

the storage class. If a storage pool's attributes and name match all of the requested attributes and pool names, Trident adds that storage pool to the set of suitable storage pools for that storage class. In addition, Trident adds all storage pools listed in the `additionalStoragePools` list to that set, even if their attributes do not fulfill all or any of the storage class's requested attributes. You should use the `excludeStoragePools` list to override and remove storage pools from use for a storage class. Trident performs a similar process every time you add a new backend, checking whether its storage pools satisfy those of the existing storage classes and removing any that have been marked as excluded.

Volume creation

Trident then uses the associations between storage classes and storage pools to determine where to provision volumes. When you create a volume, Trident first gets the set of storage pools for that volume's storage class, and, if you specify a protocol for the volume, Trident removes those storage pools that cannot provide the requested protocol (for example, a NetApp HCI/SolidFire backend cannot provide a file-based volume while an ONTAP NAS backend cannot provide a block-based volume). Trident randomizes the order of this resulting set, to facilitate an even distribution of volumes, and then iterates through it, attempting to provision the volume on each storage pool in turn. If it succeeds on one, it returns successfully, logging any failures encountered in the process. Trident returns a failure **only if** it fails to provision on **all** the storage pools available for the requested storage class and protocol.

Volume snapshots

Learn more about how Trident handles the creation of volume snapshots for its drivers.

Learn about volume snapshot creation

- For the `ontap-nas`, `ontap-san`, and `azure-netapp-files` drivers, each Persistent Volume (PV) maps to a FlexVol volume As a result, volume snapshots are created as NetApp snapshots. NetApp snapshot technology delivers more stability, scalability, recoverability, and performance than competing snapshot technologies. These snapshot copies are extremely efficient both in the time needed to create them and in storage space.
- For the `ontap-nas-flexgroup` driver, each Persistent Volume (PV) maps to a FlexGroup. As a result, volume snapshots are created as NetApp FlexGroup snapshots. NetApp snapshot technology delivers more stability, scalability, recoverability, and performance than competing snapshot technologies. These snapshot copies are extremely efficient both in the time needed to create them and in storage space.
- For the `ontap-san-economy` driver, PVs map to LUNs created on shared FlexVol volumes VolumeSnapshots of PVs are achieved by performing FlexClones of the associated LUN. ONTAP FlexClone technology makes it possible to create copies of even the largest datasets almost instantaneously. Copies share data blocks with their parents, consuming no storage except what is required for metadata.
- For the `solidfire-san` driver, each PV maps to a LUN created on the NetApp Element software/NetApp HCI cluster. VolumeSnapshots are represented by Element snapshots of the underlying LUN. These snapshots are point-in-time copies and only take up a small amount of system resources and space.
- When working with the `ontap-nas` and `ontap-san` drivers, ONTAP snapshots are point-in-time copies of the FlexVol and consume space on the FlexVol itself. This can result in the amount of writable space in the volume to reduce with time as snapshots are created/scheduled. One simple way of addressing this is to grow the volume by resizing through Kubernetes. Another option is to delete snapshots that are no longer required. When a VolumeSnapshot created through Kubernetes is deleted, Trident will delete the associated ONTAP snapshot. ONTAP snapshots that were not created through Kubernetes can also be deleted.

With Trident, you can use VolumeSnapshots to create new PVs from them. Creating PVs from these snapshots

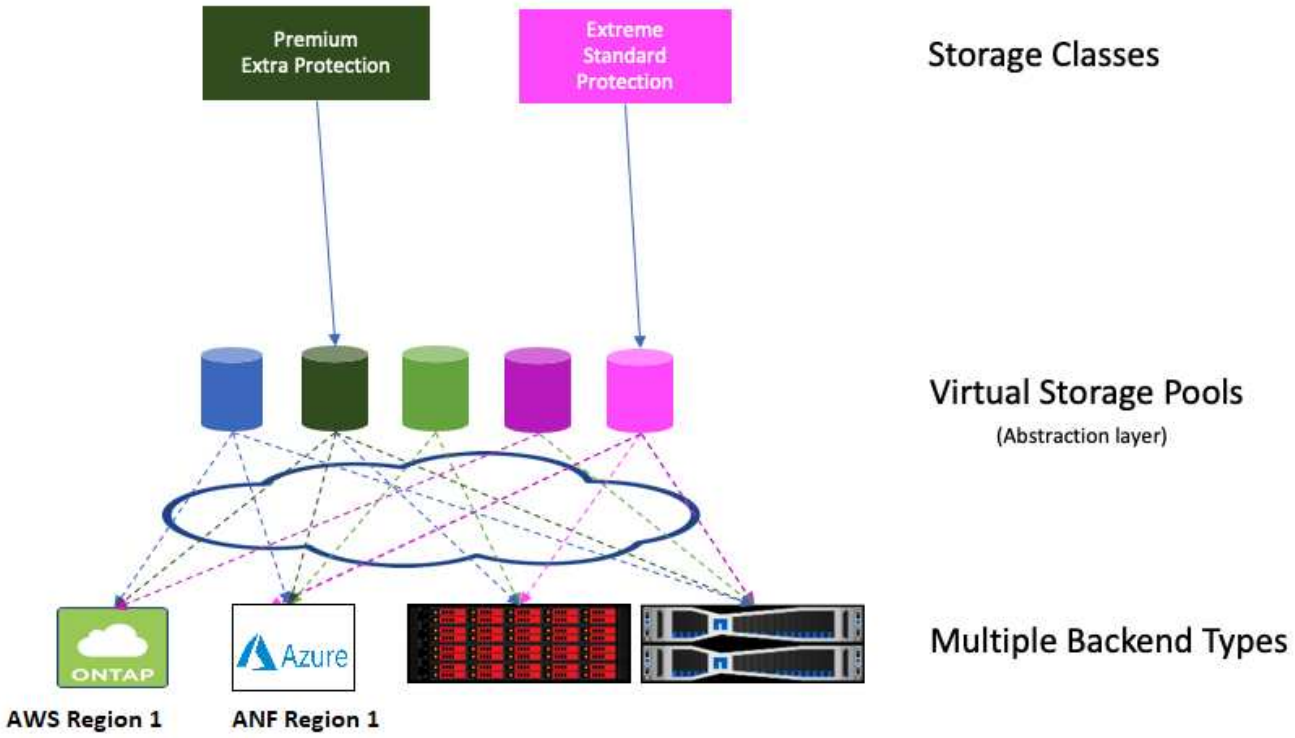
is performed by using the FlexClone technology for supported ONTAP backends. When creating a PV from a snapshot, the backing volume is a FlexClone of the snapshot's parent volume. The `solidfire-san` driver uses Element software volume clones to create PVs from snapshots. Here it creates a clone from the Element snapshot.

Virtual pools

Virtual pools provide a layer of abstraction between Trident storage backends and Kubernetes `StorageClasses`. They allow an administrator to define aspects, such as location, performance, and protection for each backend in a common, backend-agnostic way without making a `StorageClass` specify which physical backend, backend pool, or backend type to use to meet desired criteria.

Learn about virtual pools

The storage administrator can define virtual pools on any of the Trident backends in a JSON or YAML definition file.



Any aspect specified outside the virtual pools list is global to the backend and will apply to all the virtual pools, while each virtual pool might specify one or more aspects individually (overriding any backend-global aspects).

- When defining virtual pools, do not attempt to rearrange the order of existing virtual pools in a backend definition.
- We advise against modifying attributes for an existing virtual pool. You should define a new virtual pool to make changes.

Most aspects are specified in backend-specific terms. Crucially, the aspect values are not exposed outside the

backend's driver and are not available for matching in `StorageClasses`. Instead, the administrator defines one or more labels for each virtual pool. Each label is a key:value pair, and labels might be common across unique backends. Like aspects, labels can be specified per-pool or global to the backend. Unlike aspects, which have predefined names and values, the administrator has full discretion to define label keys and values as needed. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

The virtual pool labels can be defied using these characters:

- uppercase letters `A-Z`
- lowercase letters `a-z`
- numbers `0-9`
- underscores _
- hyphens -

A `StorageClass` identifies which virtual pool to use by referencing the labels within a selector parameter. Virtual pool selectors support the following operators:

Operator	Example	A pool's label value must:
`=`	performance=premium	Match
`!=`	performance!=extreme	Not match
`in`	location in (east, west)	Be in the set of values
`notin`	performance notin (silver, bronze)	Not be in the set of values
`<key>`	protection	Exist with any value
`!<key>`	!protection	Not exist

Volume access groups

Learn more about how Trident uses volume access groups.

> Ignore this section if you are using CHAP, which is recommended to simplify management and avoid the scaling limit described below. In addition, if you are using Trident in CSI mode, you can ignore this section. Trident uses CHAP when installed as an enhanced CSI provisioner.

Learn about volume access groups

Trident can use volume access groups to control access to the volumes that it provisions. If CHAP is disabled, it expects to find an access group called `trident` unless you specify one or more access group IDs in the configuration.

While Trident associates new volumes with the configured access groups, it does not create or otherwise manage access groups themselves. The access groups must exist before the storage backend is added to Trident, and they need to contain the iSCSI IQNs from every node in the Kubernetes cluster that could potentially mount the volumes provisioned by that backend. In most installations, that includes every worker node in the cluster.

For Kubernetes clusters with more than 64 nodes, you should use multiple access groups. Each access group

may contain up to 64 IQNs, and each volume can belong to four access groups. With the maximum four access groups configured, any node in a cluster up to 256 nodes in size will be able to access any volume. For latest limits on volume access groups, refer to here.

If you're modifying the configuration from one that is using the default `trident` access group to one that uses others as well, include the ID for the `trident` access group in the list.

Quick start for Trident

You can install Trident and start managing storage resources in a few steps. Before getting started, review Trident requirements.

For Docker, refer to Trident for Docker.

1 Prepare the worker node

All worker nodes in the Kubernetes cluster must be able to mount the volumes you have provisioned for your pods.

Prepare the worker node

2 Install Trident

Trident offers several installation methods and modes optimized for a variety of environments and organizations.

Install Trident

3 Create a backend

A backend defines the relationship between Trident and a storage system. It tells Trident how to communicate with that storage system and how Trident should provision volumes from it.

Configure a backend for your storage system

4 Create a Kubernetes StorageClass

The Kubernetes StorageClass object specifies Trident as the provisioner and allows you to create a storage class to provision volumes with customizable attributes. Trident creates a matching storage class for Kubernetes objects that specify the Trident provisioner.

Create a storage class

5 Provision a volume

A *PersistentVolume* (PV) is a physical storage resource provisioned by the cluster administrator on a Kubernetes cluster. The *PersistentVolumeClaim* (PVC) is a request for access to the PersistentVolume on the cluster.

Create a PersistentVolume (PV) and a PersistentVolumeClaim (PVC) that uses the configured Kubernetes StorageClass to request access to the PV. You can then mount the PV to a pod.

Provision a volume

What's next?

You can now add additional backends, manage storage classes, manage backends, and perform volume operations.

Requirements

Before installing Trident you should review these general system requirements. Specific backends might have additional requirements.

Critical information about Trident

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.36 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Supported frontends (orchestrators)

Trident supports multiple container engines and orchestrators, including the following:

- Anthos On-Prem (VMware) and Anthos on bare metal 1.16
- Kubernetes 1.27 - 1.36
- OpenShift 4.12, 4.14 - 4.21 (If you plan to use iSCSI node preparation with OpenShift 4.19, the minimum Trident version supported is 25.06.1.)

  Trident continues to support older OpenShift versions in alignment with the Red Hat Extended Update Support (EUS) release lifecycle, even if they rely on Kubernetes versions that are no longer officially supported upstream. While installing Trident in such cases, you can safely ignore any warning messages about the Kubernetes version.

- Rancher Kubernetes Engine 2 (RKE2) v1.28.x - 1.36.x

Trident also works with a host of other fully-managed and self-managed Kubernetes offerings, including Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Services (EKS), Azure Kubernetes Service

(AKS), Mirantis Kubernetes Engine (MKE), and VMWare Tanzu Portfolio.

Trident and ONTAP can be used as a storage provider for KubeVirt.

> Before upgrading a Kubernetes cluster from 1.25 to 1.26 or later that has Trident installed, refer to Upgrade a Helm installation.

Supported backends (storage)

To use Trident, you need one or more of the following supported backends:

- Amazon FSx for NetApp ONTAP
- Azure NetApp Files
- Cloud Volumes ONTAP
- Google Cloud NetApp Volumes
- NetApp All SAN Array (ASA)
- On-premises FAS, AFF, or ASA r2 (iSCSI, NVMe/TCP, and FC) running ONTAP versions under NetApp full or limited support. See Software Version Support.
- NetApp HCI/Element software 11 or above

Trident support for KubeVirt and OpenShift Virtualization

Storage drivers supported:

Trident supports the following ONTAP drivers for KubeVirt and OpenShift Virtualization:

- ontap-nas
- ontap-san (iSCSI, FCP, NVMe over TCP)
- ontap-san-economy (iSCSI Only)

Points to consider:

- Update storage class to have the `fsType` parameter (for example: `fsType: "ext4"`) in OpenShift Virtualization environment. If needed, set the volume mode to block explicitly using the `volumeMode=Block` parameter in the `dataVolumeTemplates` to notify CDI to create Block data volumes.
- *RWX access mode for block-storage drivers*: ontap-san (iSCSI, NVMe/TCP, FC) and ontap-san-economy (iSCSI) drivers are supported only with "volumeMode: Block" (raw device). For these drivers, the `fstype` parameter cannot be used because the volumes are provided in raw device mode.
- For live-migration workflow/s where RWX access mode is required, these combinations are supported:
  - NFS + `volumeMode=Filesystem`
  - iSCSI + `volumeMode=Block` (raw device)
  - NVMe/TCP + `volumeMode=Block` (raw device)
  - FC + `volumeMode=Block` (raw device)

Feature requirements

The table below summarizes the features available with this release of Trident and the versions of Kubernetes it supports.

Feature	Kubernetes version	Feature gates required?
Trident	1.27 - 1.36	No
Volume Snapshots	1.27 - 1.36	No
PVC from Volume Snapshots	1.27 - 1.36	No
iSCSI PV resize	1.27 - 1.36	No
ONTAP Bidirectional CHAP	1.27 - 1.36	No
Dynamic Export Policies	1.27 - 1.36	No
Trident Operator	1.27 - 1.36	No
CSI Topology	1.27 - 1.36	No

Tested host operating systems

Though Trident does not officially support specific operating systems, the following are known to work:

- Red Hat Enterprise Linux CoreOS (RHCOS) versions supported by OpenShift Container Platform on AMD64 and ARM64
- Red Hat Enterprise Linux (RHEL) 8 or later on AMD64 and ARM64

  NVMe/TCP requires RHEL 9 or later.

- Ubuntu 22.04 LTS or later on AMD64 and ARM64
- Windows Server 2022
- SUSE Linux Enterprise Server (SLES) 15 or later

By default, Trident runs in a container and will, therefore, run on any Linux worker. However, those workers need to be able to mount the volumes that Trident provides using the standard NFS client or iSCSI initiator, depending on the backends you are using.

The `tridentctl` utility also runs on any of these distributions of Linux.

Host configuration

All worker nodes in the Kubernetes cluster must be able to mount the volumes you have provisioned for your pods. To prepare the worker nodes, you must install NFS, iSCSI, or NVMe tools based on your driver selection.

Prepare the worker node

Storage system configuration

Trident might require changes to a storage system before a backend configuration can use it.

Configure backends

Trident ports

Trident requires access to specific ports for communication.

Trident ports

Container images and corresponding Kubernetes versions

For air-gapped installations, the following list is a reference of container images needed to install Trident. Use the `tridentctl images` command to verify the list of needed container images.

Container images required for Trident 26.02

Kubernetes versions	Container image
v1.27.0, v1.28.0, v1.29.0, v1.30.0, v1.31.0, v1.32.0, v1.33.0, v1.34.0, v1.36.0	• docker.io/netapp/trident:26.02.0 • docker.io/netapp/trident-autosupport:26.02 • registry.k8s.io/sig-storage/csi-provisioner:v6.1.0 • registry.k8s.io/sig-storage/csi-attacher:v4.10.0 • registry.k8s.io/sig-storage/csi-resizer:v2.0.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.5.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.15.0 • docker.io/netapp/trident-operator:26.02.0 (optional)

Install Trident

Learn about Trident installation

To ensure Trident can be installed in a wide variety of environments and organizations, NetApp offers multiple installation options. You can install Trident using the Trident operator (manually or using Helm) or with `tridentctl`. This topic provides important information for selecting the right installation process for you.

Critical information about Trident 26.02

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.35 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Before you begin

Regardless of your installation path, you must have:

- Full privileges to a supported Kubernetes cluster running a supported version of Kubernetes and feature requirements enabled. Review the requirements for details.
- Access to a supported NetApp storage system.
- Capability to mount volumes from all of the Kubernetes worker nodes.
- A Linux host with `kubectl` (or `oc`, if you are using OpenShift) installed and configured to manage the Kubernetes cluster that you want to use.
- The `KUBECONFIG` environment variable set to point to your Kubernetes cluster configuration.
- If you are using Kubernetes with Docker Enterprise, follow their steps to enable CLI access.
- The cluster must support privileged workloads.

If you have not familiarized yourself with the basic concepts, now is a great time to do that.

Choose your installation method

Select the installation method that's right for you. You should also review the considerations for moving between methods before making your decision.

Using the Trident operator

Whether deploying manually or using Helm, the Trident operator is a great way to simplify installation and dynamically manage Trident resources. You can even customize your Trident operator deployment using the attributes in the `TridentOrchestrator` custom resource (CR).

The benefits of using the Trident operator include:

Trident object creation

The Trident operator automatically creates the following objects for your Kubernetes version.

- ServiceAccount for the operator
- ClusterRole and ClusterRoleBinding to the ServiceAccount
- Dedicated PodSecurityPolicy (for Kubernetes 1.25 and earlier)
- The operator itself

Resource accountability

The cluster-scoped Trident operator manages resources associated with a Trident installation at the cluster level. This mitigates errors that might be caused when maintaining cluster-scoped resources using a namespace-scoped operator. This is essential for self-healing and patching.

Self-healing capability

The operator monitors Trident installation and actively takes measures to address issues, such as when the deployment is deleted or if it is accidentally modified. A `trident-operator-<generated-id>` pod is created that associates a `TridentOrchestrator` CR with a Trident installation. This ensures there is only one instance of Trident in the cluster and controls its setup, making sure the installation is idempotent. When changes are made to the installation (such as, deleting the deployment or node daemonset), the operator identifies them and fixes them individually.

Easy updates to existing installations

You can easily update an existing deployment with the operator. You only need to edit the `TridentOrchestrator` CR to make updates to an installation.

For example, consider a scenario where you need to enable Trident to generate debug logs. To do this, patch your `TridentOrchestrator` to set `spec.debug` to `true`:

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

After `TridentOrchestrator` is updated, the operator processes the updates and patches the existing installation. This might trigger the creation of new pods to modify the installation accordingly.

Clean reinstallation

The cluster-scoped Trident operator enables clean removal of cluster-scoped resources. Users can completely uninstall Trident and easily reinstall.

Automatic Kubernetes upgrade handling

When the Kubernetes version of the cluster is upgraded to a supported version, the operator updates an existing Trident installation automatically and changes it to ensure that it meets the requirements of the Kubernetes version.

> If the cluster is upgraded to an unsupported version, the operator prevents installing Trident. If Trident has already been installed with the operator, a warning is displayed to indicate that Trident is installed on an unsupported Kubernetes version.

Using `tridentctl`

If you have an existing deployment that must be upgraded or if you are looking to highly customize your deployment, you should consider installing using `tridentctl`. This is the conventional method of deploying Trident.

You can customize your `tridentctl` installation to generate the manifests for Trident resources. This includes the deployment, daemonset, service account, and the cluster role that Trident creates as part of its installation.

> Beginning with the 22.04 release, AES keys will no longer be regenerated every time Trident is installed. With this release, Trident will install a new secret object that persists across installations. This means, `tridentctl` in 22.04 can uninstall previous versions of Trident, but earlier versions cannot uninstall 22.04 installations.
> Select the appropriate installation *method*.

Choose your installation mode

Determine your deployment process based on the *installation mode* (Standard, Offline, or Remote) required by your organization.

Standard installation

This is the easiest way to install Trident and works for most environments that do not impose network restrictions. Standard installation mode uses default registries to store required Trident (`docker.io`) and CSI (`registry.k8s.io`) images.

When you use standard mode, the Trident installer:

- Fetches the container images over the Internet
- Creates a deployment or node daemonset, which spins up Trident pods on all the eligible nodes in the Kubernetes cluster

Offline installation

Offline installation mode might be required in an air-gapped or secure location. In this scenario, you can create a single private, mirrored registry or two mirrored registries to store required Trident and CSI images.

Regardless of your registry configuration, CSI images must reside in one registry.

Remote installation

Here is a high-level overview of the remote installation process:

- Deploy the appropriate version of `kubectl` on the remote machine from where you want to deploy Trident.
- Copy the configuration files from the Kubernetes cluster and set the `KUBECONFIG` environment variable on the remote machine.
- Initiate a `kubectl get nodes` command to verify that you can connect to the required Kubernetes cluster.
- Complete the deployment from the remote machine by using the standard installation steps.

Select the process based on your method and mode

After you've made your decisions, select the appropriate process.

Method	Installation mode
Trident operator (manually)	Standard installation Offline installation
Trident operator (Helm)	Standard installation Offline installation
`tridentctl`	Standard or offline installation

Moving between installation methods

You can decide to change your installation method. Before doing so, consider the following:

- Always use the same method for installing and uninstalling Trident. If you have deployed with `tridentctl`, you should use the appropriate version of the `tridentctl` binary to uninstall Trident. Similarly, if you are deploying with the operator, you should edit the `TridentOrchestrator` CR and set `spec.uninstall=true` to uninstall Trident.
- If you have an operator-based deployment that you want to remove and use instead `tridentctl` to deploy Trident, you should first edit `TridentOrchestrator` and set `spec.uninstall=true` to uninstall Trident. Then delete `TridentOrchestrator` and the operator deployment. You can then install using `tridentctl`.
- If you have a manual operator-based deployment, and you want to use Helm-based Trident operator deployment, you should manually uninstall the operator first, and then perform the Helm install. This enables Helm to deploy the Trident operator with the required labels and annotations. If you do not do this, your Helm-based Trident operator deployment will fail with label validation error and annotation validation error.
- If you have a `tridentctl`-based deployment, you can perform Helm-based or Operator-based deployment without uninstalling Trident.

Other known configuration options

When installing Trident on VMWare Tanzu Portfolio products:

- The `--kubelet-dir` flag should be set to the location of kubelet directory. By default, this is `/var/vcap/data/kubelet`.

  Specifying the kubelet location using `--kubelet-dir` is known to work for Trident Operator, Helm, and `tridentctl` deployments.

Install using Trident operator

Manually deploy the Trident operator (Standard mode)

You can manually deploy the Trident operator to install Trident. This process applies to installations where the container images required by Trident are not stored in a private registry. If you do have a private image registry, use the process for offline deployment.

Critical information about Trident 26.02

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.35 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Manually deploy the Trident operator and install Trident

Review the installation overview to ensure you've met installation prerequisites and selected the correct installation option for your environment.

Before you begin

Before you begin installation, log in to the Linux host and verify it is managing a working, supported Kubernetes cluster and that you have the necessary privileges.

With OpenShift, use `oc` instead of `kubectl` in all of the examples that follow, and log in as **system:admin** first by running `oc login -u system:admin` or `oc login -u kube-admin`.

1. Verify your Kubernetes version:

```
kubectl version
```

2. Verify cluster administrator privileges:

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

3. Verify you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network:

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

Step 1: Download the Trident installer package

The Trident installer package contains everything you need to deploy the Trident operator and install Trident. Download and extract the latest version of the Trident installer from the *Assets* section on GitHub.

```
wget https://github.com/NetApp/trident/releases/download/v26.02.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

Step 2: Create the `TridentOrchestrator` CRD

Create the `TridentOrchestrator` Custom Resource Definition (CRD). You will create a `TridentOrchestrator` Custom Resource later. Use the appropriate CRD YAML version in `deploy/crds` to create the `TridentOrchestrator` CRD.

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

Step 3: Deploy the Trident operator

The Trident installer provides a bundle file that can be used to install the operator and create associated objects. The bundle file is an easy way to deploy the operator and install Trident using a default configuration.

- For clusters running Kubernetes 1.24, use `bundle_pre_1_25.yaml`.
- For clusters running Kubernetes 1.25 or later, use `bundle_post_1_25.yaml`.

Before you begin

- By default, the Trident installer deploys the operator in the `trident` namespace. If the `trident` namespace does not exist, create it using:

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

- To deploy the operator in a namespace other than the `trident` namespace, update `serviceaccount.yaml`, `clusterrolebinding.yaml` and `operator.yaml` and generate your bundle file using the `kustomization.yaml`.

  1. Create the `kustomization.yaml` using the following command where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version.

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

  2. Compile the bundle using using the following command where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version.

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

Steps

1. Create the resources and deploy the operator:

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

2. Verify the operator, deployment, and replicasets were created.

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

> There should only be **one instance** of the operator in a Kubernetes cluster. Do not create multiple deployments of the Trident operator.

Step 4: Create the `TridentOrchestrator` and install Trident

You can now create the `TridentOrchestrator` and install Trident. Optionally, you can customize your Trident installation using the attributes in the `TridentOrchestrator` spec.

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
  nodePrep:
  - iscsi
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:26.02
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:26.02.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v26.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

Verify the installation

There are several ways to verify your installation.

Using `TridentOrchestrator` status

The status of `TridentOrchestrator` indicates if the installation was successful and displays the version of Trident installed. During the installation, the status of `TridentOrchestrator` changes from `Installing` to `Installed`. If you observe the `Failed` status and the operator is unable to recover by itself, check the logs.

Status	Description
Installing	The operator is installing Trident using this `TridentOrchestrator` CR.
Installed	Trident has successfully installed.
Uninstalling	The operator is uninstalling Trident, because `spec.uninstall=true`.
Uninstalled	Trident is uninstalled.
Failed	The operator could not install, patch, update or uninstall Trident; the operator will automatically try to recover from this state. If this state persists you will require troubleshooting.
Updating	The operator is updating an existing installation.
Error	The `TridentOrchestrator` is not used. Another one already exists.

Using pod creation status

You can confirm if the Trident installation completed by reviewing the status of the created pods:

```
kubectl get pods -n trident

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

Using `tridentctl`

You can use `tridentctl` to check the version of Trident installed.

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.02.0        | 26.02.0        |
+----------------+----------------+
```

Manually deploy the Trident operator (Offline mode)

You can manually deploy the Trident operator to install Trident. This process applies to installations where the container images required by Trident are stored in a private registry. If you do not have a private image registry, use the process for standard deployment.

Critical information about Trident

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.35 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Manually deploy the Trident operator and install Trident

Review the installation overview to ensure you've met installation prerequisites and selected the correct installation option for your environment.

Before you begin

Log in to the Linux host and verify it is managing a working and supported Kubernetes cluster and that you have the necessary privileges.

With OpenShift, use `oc` instead of `kubectl` in all of the examples that follow, and log in as **system:admin** first by running `oc login -u system:admin` or `oc login -u kube-admin`.

1. Verify your Kubernetes version:

```
kubectl version
```

2. Verify cluster administrator privileges:

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

3. Verify you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network:

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

Step 1: Download the Trident installer package

The Trident installer package contains everything you need to deploy the Trident operator and install Trident. Download and extract the latest version of the Trident installer from the *Assets* section on GitHub.

```
wget https://github.com/NetApp/trident/releases/download/v6.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

Step 2: Create the `TridentOrchestrator` CRD

Create the `TridentOrchestrator` Custom Resource Definition (CRD). You will create a `TridentOrchestrator` Custom Resources later. Use the appropriate CRD YAML version in `deploy/crds` to create the `TridentOrchestrator` CRD:

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

Step 3: Update the registry location in the operator

In `/deploy/operator.yaml`, update `image: docker.io/netapp/trident-operator:26.02.0` to reflect the location of your image registry. Your Trident and CSI images can be located in one registry or different registries, but all CSI images must be located in the same registry. For example:

- `image: <your-registry>/trident-operator:26.02.0` if your images are all located in one registry.
- `image: <your-registry>/netapp/trident-operator:26.02.0` if your Trident image is located in a different registry from your CSI images.

Step 4: Deploy the Trident operator

The Trident installer provides a bundle file that can be used to install the operator and create associated objects. The bundle file is an easy way to deploy the operator and install Trident using a default configuration.

- For clusters running Kubernetes 1.24, use `bundle_pre_1_25.yaml`.
- For clusters running Kubernetes 1.25 or later, use `bundle_post_1_25.yaml`.

Before you begin

- By default, the Trident installer deploys the operator in the `trident` namespace. If the `trident` namespace does not exist, create it using:

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

- To deploy the operator in a namespace other than the `trident` namespace, update `serviceaccount.yaml`, `clusterrolebinding.yaml` and `operator.yaml` and generate your bundle file using the `kustomization.yaml`.

  1. Create the `kustomization.yaml` using the following command where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version.

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

  2. Compile the bundle using using the following command where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version.

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

Steps

1. Create the resources and deploy the operator:

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

2. Verify the operator, deployment, and replicasets were created.

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

> There should only be **one instance** of the operator in a Kubernetes cluster. Do not create multiple deployments of the Trident operator.

Step 5: Update the image registry location in the `TridentOrchestrator`

Your Trident and CSI images can be located in one registry or different registries, but all CSI images must be

located in the same registry. Update `deploy/crds/tridentorchestrator_cr.yaml` to add the additional location specs based on your registry configuration.

Images in one registry

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.02"
tridentImage: "<your-registry>/trident:26.02.0"
```

Images in different registries

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.02"
tridentImage: "<your-registry>/trident:26.02.0"
```

Step 6: Create the `TridentOrchestrator` and install Trident

You can now create the `TridentOrchestrator` and install Trident. Optionally, you can further customize your Trident installation using the attributes in the `TridentOrchestrator` spec. The following example shows an installation where Trident and CSI images are located in different registries.

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/trident-autosupport:26.02
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:26.02.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/trident-autosupport:26.02
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Http Request Timeout:       90s
    Image Pull Secrets:
    Image Registry:       <your-registry>
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
    Trident Image:        <your-registry>/trident:26.02.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v26.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

Verify the installation

There are several ways to verify your installation.

Using `TridentOrchestrator` status

The status of `TridentOrchestrator` indicates if the installation was successful and displays the version of Trident installed. During the installation, the status of `TridentOrchestrator` changes from `Installing` to `Installed`. If you observe the `Failed` status and the operator is unable to recover by itself, check the logs.

Status	Description
Installing	The operator is installing Trident using this `TridentOrchestrator` CR.
Installed	Trident has successfully installed.
Uninstalling	The operator is uninstalling Trident, because `spec.uninstall=true`.
Uninstalled	Trident is uninstalled.
Failed	The operator could not install, patch, update or uninstall Trident; the operator will automatically try to recover from this state. If this state persists you will require troubleshooting.
Updating	The operator is updating an existing installation.
Error	The `TridentOrchestrator` is not used. Another one already exists.

Using pod creation status

You can confirm if the Trident installation completed by reviewing the status of the created pods:

```
kubectl get pods -n trident

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

Using `tridentctl`

You can use `tridentctl` to check the version of Trident installed.

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.02.0        | 26.02.0        |
+----------------+----------------+
```

Deploy Trident operator using Helm (Standard mode)

You can deploy the Trident operator and install Trident using Helm. This process applies to installations where the container images required by Trident are not stored in a private registry. If you do have a private image registry, use the process for offline deployment.

Critical information about Trident 25.10

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.35 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Deploy the Trident operator and install Trident using Helm

Using the Trident Helm Chart you can deploy the Trident operator and install Trident in one step.

Review the installation overview to ensure you've met installation prerequisites and selected the correct installation option for your environment.

Before you begin

In addition to the deployment prerequisites you need Helm version 3.

Steps

1. Add the Trident Helm repository:

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

2. Use `helm install` and specify a name for your deployment as in the following example where `100..0` is the version of Trident you are installing.

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace <trident-namespace>
```

> If you already created a namespace for Trident, the `--create-namespace` parameter will not create an additional namespace.

You can use `helm list` to review installation details such as name, namespace, chart, status, app version, and revision number.

Pass configuration data during install

There are two ways to pass configuration data during the install:

Option	Description
`--values` (or `-f`)	Specify a YAML file with overrides. This can be specified multiple times and the rightmost file will take precedence.
`--set`	Specify overrides on the command line.

For example, to change the default value of `debug`, run the following command where `100.2602.0` is the version of Trident you are installing:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set tridentDebug=true
```

Configuration options

This table and the `values.yaml` file, which is part of the Helm chart, provide the list of keys and their default values.

Option	Description	Default
`nodeSelector`	Node labels for pod assignment	
`podAnnotations`	Pod annotations	
`deploymentAnnotations`	Deployment annotations	

Option	Description	Default
`tolerations`	Tolerations for pod assignment	

`affinity`	Affinity for pod assignment	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedulingIgnoredDuringExecution:`
		`  nodeSelectorTerms:`
		`  - matchExpressions:`
		`  - key: kubernetes.io/arch`
		`  operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key: kubernetes.io/os`
		`  operator: In`
		`  values:`
		`  - linux`
		
		Do not remove the default affinity from the values.yaml file. When you want to provide a custom affinity, extend the default affinity.

`tridentControllerPluginNodeSelector`	Additional node selectors for pods. Refer to Understanding controller pods and node pods for details.	
`tridentControllerPluginTolerations`	Overrides Kubernetes tolerations for pods. Refer to Understanding controller pods and node pods for details.	
`tridentNodePluginNodeSelector`	Additional node selectors for pods. Refer to Understanding controller pods and node pods for details.	
`tridentNodePluginTolerations`	Overrides Kubernetes tolerations for pods. Refer to Understanding controller pods and node pods for details.	

Option	Description	Default
`imageRegistry`	Identifies the registry for the `trident-operator`, `trident`, and other images. Leave empty to accept the default. IMPORTANT: When installing Trident in a private repository, if you are using the `imageRegistry` switch to specify the repository location, do not use `/netapp/` in the repository path.	`""`
`imagePullPolicy`	Sets the image pull policy for the `trident-operator`.	`IfNotPresent`
`imagePullSecrets`	Sets the image pull secrets for the `trident-operator`, `trident`, and other images.	
`kubeletDir`	Allows overriding the host location of kubelet's internal state.	`"/var/lib/kubelet"`
`operatorLogLevel`	Allows the log level of the Trident operator to be set to: `trace`, `debug`, `info`, `warn`, `error`, or `fatal`.	`"info"`
`operatorDebug`	Allows the log level of the Trident operator to be set to debug.	`true`
`operatorImage`	Allows the complete override of the image for `trident-operator`.	`""`
`operatorImageTag`	Allows overriding the tag of the `trident-operator` image.	`""`
`tridentIPv6`	Allows enabling Trident to work in IPv6 clusters.	`false`
`tridentK8sTimeout`	Overrides the default 30-second timeout for most Kubernetes API operations (if non-zero, in seconds).	`0`
`tridentHttpRequestTimeout`	Overrides the default 90-second timeout for the HTTP requests, with `0s` being an infinite duration for the timeout. Negative values are not allowed.	`"90s"`
`tridentSilenceAutosupport`	Allows disabling Trident periodic AutoSupport reporting.	`false`

Option	Description	Default
`tridentAutosupportImageTag`	Allows overriding the tag of the image for Trident AutoSupport container.	`<version>`
`tridentAutosupportProxy`	Enables Trident AutoSupport container to phone home via an HTTP proxy.	`""`
`tridentLogFormat`	Sets the Trident logging format (`text` or `json`).	`"text"`
`tridentDisableAuditLog`	Disables Trident audit logger.	`true`
`tridentLogLevel`	Allows the log level of Trident to be set to: `trace`, `debug`, `info`, `warn`, `error`, or `fatal`.	`"info"`
`tridentDebug`	Allows the log level of Trident to be set to `debug`. You can automate the force-detach process through integration with node health check (NHC) operator. For information, see Automating the failover of stateful applications with Trident.	`false`
`tridentLogWorkflows`	Allows specific Trident workflows to be enabled for trace logging or log suppression.	`""`
`tridentLogLayers`	Allows specific Trident layers to be enabled for trace logging or log suppression.	`""`
`tridentImage`	Allows the complete override of the image for Trident.	`""`
`tridentImageTag`	Allows overriding the tag of the image for Trident.	`""`
`tridentProbePort`	Allows overriding the default port used for Kubernetes liveness/readiness probes.	`""`
`windows`	Enables Trident to be installed on Windows worker node.	`false`
`enableForceDetach`	Allows enabling the force detach feature.	`false`
`excludePodSecurityPolicy`	Excludes the operator pod security policy from creation.	`false`

Option	Description	Default
`cloudProvider`	Set to `"Azure"` when using managed identities or a cloud identity on an AKS cluster. Set to "AWS" when using a cloud identity on an EKS cluster.	`""`
`cloudIdentity`	Set to workload identity ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx") when using cloud identity on an AKS cluster. Set to AWS IAM role ("'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'") when using cloud identity on an EKS cluster.	`""`
`iscsiSelfHealingInterval`	The interval at which the iSCSI self-healing is invoked.	`5m0s`
`iscsiSelfHealingWaitTime`	The duration after which iSCSI self-healing initiates an attempt to resolve a stale session by performing a logout and subsequent login.	`7m0s`
`nodePrep`	Enables Trident to prepare the nodes of the Kubernetes cluster to manage volumes using the specified data storage protocol. **Currently, `iscsi` is the only value supported.** NOTE: Beginning with OpenShift 4.19, the minimum Trident version supported for this feature is 25.06.1.	

Option	Description	Default
`enableConcurrency`	Enables concurrent Trident controller operations for improved throughput. **Tech Preview**: This feature is experimental and currently supports limited parallel workflows with the ONTAP-NAS (NFS only) and ONTAP-SAN (NVMe for unified ONTAP 9) drivers, in addition to the existing tech preview for the ONTAP-SAN driver (iSCSI and FCP protocols in unified ONTAP 9).	false
`k8sAPIQPS`	The queries per second (QPS) limit used by the controller while communicating with the Kubernetes API server. The Burst value is set automatically based on the QPS value.	`100`; optional

Option	Description	Default

`resources`	Sets Kubernetes resource limits and requests for the Trident controller, node, and operator pods. You can configure CPU and memory for each container and sidecar to manage resource allocation in Kubernetes.
	
	For more information about configuring resource requests and limits, refer to Resource Management for Pods and Containers.
	
	• DO NOT change the names of any containers or fields.
	• DO NOT change the indentation - YAML indentation is critical for proper parsing.
	
	• No limits are applied by default - only requests have default values and are automatically applied if not specified.
	• Container names are listed as they appear in the pod specifications.
	• Sidecars are listed under each main container.
	• Check the TORC's `status.CurrentInstallationParams` field to view the values currently applied.
		`resources:`
		`  controller:`
		`  trident-main:`
		`  requests:`
		`  cpu: 10m`
		`  memory: 80Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-provisioner:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-attacher:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-resizer:`
		`  requests:`
		`  cpu: 3m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-snapshotter:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  trident-autosupport:`
		`  requests:`
		`  cpu: 1m`
		`  memory: 30Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  node:`
		`  linux:`
		`  trident-main:`

Option	Description	Default
`httpsMetrics`	Enable HTTPS for Prometheus metrics endpoint.	false
`hostNetwork`	Enables host networking for the Trident controller. This is useful when you want to separate the frontend and backend traffic in a multi-home network.	false

Understanding controller pods and node pods

Trident runs as a single controller pod, plus a node pod on each worker node in the cluster. The node pod must be running on any host where you want to potentially mount a Trident volume.

Kubernetes node selectors and tolerations and taints are used to constrain a pod to run on a specific or preferred node. Using the`ControllerPlugin` and `NodePlugin`, you can specify constraints and overrides.

- The controller plugin handles volume provisioning and management, such as snapshots and resizing.
- The node plugin handles attaching the storage to the node.

Deploy Trident operator using Helm (Offline mode)

You can deploy the Trident operator and install Trident using Helm. This process applies to installations where the container images required by Trident are stored in a private registry. If you do not have a private image registry, use the process for standard deployment.

Critical information about Trident 26.02

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.35 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Deploy the Trident operator and install Trident using Helm

Using the Trident Helm Chart you can deploy the Trident operator and install Trident in one step.

Review the installation overview to ensure you've met installation prerequisites and selected the correct

installation option for your environment.

Before you begin

In addition to the deployment prerequisites you need Helm version 3.

> When installing Trident in a private repository, if you are using the `imageRegistry` switch to specify the repository location, do not use `/netapp/` in the repository path.

Steps

1. Add the Trident Helm repository:

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

2. Use `helm install` and specify a name for your deployment and image registry location. Your Trident and CSI images can be located in one registry or different registries, but all CSI images must be located in the same registry. In the examples, `100.2602.0` is the version of Trident you are installing.

Images in one registry

```
helm install <name> netapp-trident/trident-operator --version
100.2602.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

Images in different registries

```
helm install <name> netapp-trident/trident-operator --version
100.2602.0 --set imageRegistry=<your-registry> --set operatorImage
=<your-registry>/trident-operator:26.02.0 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:26.02
--set tridentImage=<your-registry>/trident:26.02.0 --create
-namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```

> If you already created a namespace for Trident, the `--create-namespace` parameter will not create an additional namespace.

You can use `helm list` to review installation details such as name, namespace, chart, status, app version, and revision number.

Pass configuration data during install

There are two ways to pass configuration data during the install:

Option	Description
`--values` (or `-f`)	Specify a YAML file with overrides. This can be specified multiple times and the rightmost file will take precedence.
`--set`	Specify overrides on the command line.

For example, to change the default value of `debug`, run the following command where `100.2602.0` is the version of Trident you are installing:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set tridentDebug=true
```

To add the nodePrep value, run the following command:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

Configuration options

This table and the `values.yaml` file, which is part of the Helm chart, provide the list of keys and their default values.

Do not remove the default affinity from the values.yaml file. When you want to provide a custom affinity, extend the default affinity.

Option	Description	Default
`nodeSelector`	Node labels for pod assignment	
`podAnnotations`	Pod annotations	
`deploymentAnnotations`	Deployment annotations	
`tolerations`	Tolerations for pod assignment	

Option	Description	Default

`affinity`	Affinity for pod assignment	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedul`
		`ingIgnoredDuringExecu`
		`tion:`
		
		`nodeSelectorTerms:`
		`  -`
		`matchExpressions:`
		`  - key:`
		`kubernetes.io/arch`
		
		`operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key:`
		`kubernetes.io/os`
		
		`operator: In`
		`  values:`
		`  - linux`
		
		Do not remove the default affinity from the values.yaml file. When you want to provide a custom affinity, extend the default affinity.

`tridentControllerPluginNodeSelector`	Additional node selectors for pods. Refer to Understanding controller pods and node pods for details.	
`tridentControllerPluginTolerations`	Overrides Kubernetes tolerations for pods. Refer to Understanding controller pods and node pods for details.	
`tridentNodePluginNodeSelector`	Additional node selectors for pods. Refer to Understanding controller pods and node pods for details.	

Option	Description	Default
`tridentNodePluginTolerations`	Overrides Kubernetes tolerations for pods. Refer to Understanding controller pods and node pods for details.	
`imageRegistry`	Identifies the registry for the `trident-operator`, `trident`, and other images. Leave empty to accept the default. IMPORTANT: When installing Trident in a private repository, if you are using the `imageRegistry` switch to specify the repository location, do not use `/netapp/` in the repository path.	""
`imagePullPolicy`	Sets the image pull policy for the `trident-operator`.	`IfNotPresent`
`imagePullSecrets`	Sets the image pull secrets for the `trident-operator`, `trident`, and other images.	
`kubeletDir`	Allows overriding the host location of kubelet's internal state.	`"/var/lib/kubelet"`
`operatorLogLevel`	Allows the log level of the Trident operator to be set to: `trace`, `debug`, `info`, `warn`, `error`, or `fatal`.	`"info"`
`operatorDebug`	Allows the log level of the Trident operator to be set to debug.	`true`
`operatorImage`	Allows the complete override of the image for `trident-operator`.	""
`operatorImageTag`	Allows overriding the tag of the `trident-operator` image.	""
`tridentIPv6`	Allows enabling Trident to work in IPv6 clusters.	`false`
`tridentK8sTimeout`	Overrides the default 180-second timeout for most Kubernetes API operations (if non-zero, in seconds). The `tridentK8sTimeout` parameter is applicable only for Trident installation.	`180`

Option	Description	Default
`tridentHttpRequestTimeout`	Overrides the default 90-second timeout for the HTTP requests, with `0s` being an infinite duration for the timeout. Negative values are not allowed.	`"90s"`
`tridentSilenceAutosupport`	Allows disabling Trident periodic AutoSupport reporting.	`false`
`tridentAutosupportImageTag`	Allows overriding the tag of the image for Trident AutoSupport container.	`<version>`
`tridentAutosupportProxy`	Enables Trident AutoSupport container to phone home via an HTTP proxy.	""
`tridentLogFormat`	Sets the Trident logging format (`text` or `json`).	`"text"`
`tridentDisableAuditLog`	Disables Trident audit logger.	`true`
`tridentLogLevel`	Allows the log level of Trident to be set to: `trace`, `debug`, `info`, `warn`, `error`, or `fatal`.	`"info"`
`tridentDebug`	Allows the log level of Trident to be set to `debug`.	`false`
`tridentLogWorkflows`	Allows specific Trident workflows to be enabled for trace logging or log suppression.	""
`tridentLogLayers`	Allows specific Trident layers to be enabled for trace logging or log suppression.	""
`tridentImage`	Allows the complete override of the image for Trident.	""
`tridentImageTag`	Allows overriding the tag of the image for Trident.	""
`tridentProbePort`	Allows overriding the default port used for Kubernetes liveness/readiness probes.	""
`windows`	Enables Trident to be installed on Windows worker node.	`false`
`enableForceDetach`	Allows enabling the force detach feature. You can automate the force-detach process through integration with node health check (NHC) operator. For information, see Automating the failover of stateful applications with Trident.	`false`

Option	Description	Default
`excludePodSecurityPolicy`	Excludes the operator pod security policy from creation.	`false`
`nodePrep`	Enables Trident to prepare the nodes of the Kubernetes cluster to manage volumes using the specified data storage protocol. **Currently, `iscsi` is the only value supported.** Beginning with OpenShift 4.19, the minimum Trident version supported for this feature is 25.06.1.	

Option	Description	Default

`resources`	Sets Kubernetes resource limits and requests for the Trident controller, node, and operator pods. You can configure CPU and memory for each container and sidecar to manage resource allocation in Kubernetes.
	
	For more information about configuring resource requests and limits, refer to Resource Management for Pods and Containers.
	
	• DO NOT change the names of any containers or fields.
	• DO NOT change the indentation - YAML indentation is critical for proper parsing.
	
	• No limits are applied by default - only requests have default values.
	• Container names are listed as they appear in the pod specifications.
	• Sidecars are listed under each main container.
	• Check the TORC's `status.CurrentInstallationParams` field to view the values currently applied.
		`resources:`
		`  controller:`
		`    trident-main:`
		`      requests:`
		`        cpu: 10m`
		`        memory: 80Mi`
		`      limits:`
		`          cpu:`
		`          memory:`
		`    csi-provisioner:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-attacher:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-resizer:`
		`      requests:`
		`        cpu: 3m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-snapshotter:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    trident-autosupport:`
		`      requests:`
		`        cpu: 1m`
		`        memory: 30Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`  node:`
		`    linux:`

Customize Trident operator installation

The Trident operator allows you to customize Trident installation using the attributes in the `TridentOrchestrator` spec. If you want to customize the installation beyond what `TridentOrchestrator` arguments allow, consider using `tridentctl` to generate custom YAML manifests to modify as needed.

Understanding controller pods and node pods

Trident runs as a single controller pod and a node pod on each worker node in the cluster. The node pod must be running on any host where you want to potentially mount a Trident volume.

Kubernetes node selectors and tolerations and taints are used to constrain a pod to run on a specific or preferred node. Using the`ControllerPlugin` and `NodePlugin`, you can specify constraints and overrides.

- The controller plugin handles volume provisioning and management, such as snapshots and resizing.
- The node plugin handles attaching the storage to the node.

Configuration options

> `spec.namespace` is specified in `TridentOrchestrator` to signify the namespace where Trident is installed. This parameter **cannot be updated after Trident is installed**. Attempting to do so causes the `TridentOrchestrator` status to change to `Failed`. Trident is not intended to be migrated across namespaces.

This table details `TridentOrchestrator` attributes.

Parameter	Description	Default
`namespace`	Namespace to install Trident in	`"default"`
`debug`	Enable debugging for Trident	`false`
`enableForceDetach`	`ontap-san`, `ontap-san-economy`, `ontap-nas`, and `ontap-nas-economy` only. Works with Kubernetes Non-Graceful Node Shutdown (NGNS) to grant cluster administrators ability to safely migrate workloads with mounted volumes to new nodes should a node become unhealthy. For information, see Automating the failover of stateful applications with Trident.	`false`
`windows`	Setting to `true` enables installation on Windows worker nodes.	`false`

Parameter	Description	Default
`cloudProvider`	Set to `"Azure"` when using managed identities or a cloud identity on an AKS cluster. Set to `"AWS"` when using a cloud identity on an EKS cluster. Set to `"GCP"` when using a cloud identity on a GKE cluster.	`""`
`cloudIdentity`	Set to workload identity ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx") when using cloud identity on an AKS cluster. Set to AWS IAM role ("'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'") when using cloud identity on an EKS cluster. Set to cloud identity ("'iam.gke.io/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com'") when using cloud identity on a GKE cluster.	`""`
`IPv6`	Install Trident over IPv6	false
`k8sTimeout`	Timeout for Kubernetes operations. The `k8sTimeout` parameter is applicable only for Trident installation.	`180sec`
`silenceAutosupport`	Don't send autosupport bundles to NetApp automatically	`false`
`autosupportImage`	The container image for Autosupport Telemetry	`"netapp/trident-autosupport10"`
`autosupportProxy`	The address/port of a proxy for sending Autosupport Telemetry	`"http://proxy.example.com:8888"`
`uninstall`	A flag used to uninstall Trident	`false`
`logFormat`	Trident logging format to be used [text,json]	`"text"`
`tridentImage`	Trident image to install	`"netapp/trident:26.02"`
`imageRegistry`	Path to internal registry, of the format `<registry FQDN>[:port][/subpath]`	`"registry.k8s.io"`
`kubeletDir`	Path to the kubelet directory on the host	`"/var/lib/kubelet"`
`wipeout`	A list of resources to delete to perform a complete removal of Trident	
`imagePullSecrets`	Secrets to pull images from an internal registry	

Parameter	Description	Default
`imagePullPolicy`	Sets the image pull policy for the the Trident operator. Valid values are: `Always` to always pull the image. `IfNotPresent` to pull the image only if it does not already exist on the node. `Never` to never pull the image.	`IfNotPresent`
`controllerPluginNodeSelector`	Additional node selectors for pods. Follows same format as `pod.spec.nodeSelector`.	No default; optional
`controllerPluginTolerations`	Overrides Kubernetes tolerations for pods. Follows the same format as `pod.spec.Tolerations`.	No default; optional
`nodePluginNodeSelector`	Additional node selectors for pods. Follows same format as `pod.spec.nodeSelector`.	No default; optional
`nodePluginTolerations`	Overrides Kubernetes tolerations for pods. Follows the same format as `pod.spec.Tolerations`.	No default; optional
`nodePrep`	Enables Trident to prepare the nodes of the Kubernetes cluster to manage volumes using the specified data storage protocol. **Currently, `iscsi` is the only value supported.** Beginning with OpenShift 4.19, the minimum Trident version supported for this feature is 25.06.1.	
`k8sAPIQPS`	The queries per second (QPS) limit used by the controller while communicating with the Kubernetes API server. The Burst value is set automatically based on the QPS value.	`100`; optional
`enableConcurrency`	Enables concurrent Trident controller operations for improved throughput. **Tech Preview**: This feature is experimental and currently supports limited parallel workflows with the ONTAP-NAS (NFS only) and ONTAP-SAN (NVMe for unified ONTAP 9) drivers, in addition to the existing tech preview for the ONTAP-SAN driver (iSCSI and FCP protocols in unified ONTAP 9).	false

Parameter	Description	Default
`resources`	Sets Kubernetes resource limits and requests for the Trident controller and node pods. You can configure CPU and memory for each container and sidecar to manage resource allocation in Kubernetes. For more information about configuring resource requests and limits, refer to Resource Management for Pods and Containers. • DO NOT change the names of any containers or fields. • DO NOT change the indentation - YAML indentation is critical for proper parsing. • No limits are applied by default - only requests have default values and are applied automatically if not specified. • Container names are listed as they appear in the pod specifications. • Sidecars are listed under each main container. • Check the TORC's `status.CurrentInstallationParams` field to view the values currently applied.	`resources: controller: trident-main: requests: cpu: 10m memory: 80Mi limits: cpu: memory: csi-provisioner: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-attacher: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-resizer: requests: cpu: 3m memory: 20Mi limits: cpu: memory: csi-snapshotter: requests: cpu: 2m memory: 20Mi limits:`

Parameter	Description	Default
`httpsMetrics`	Enable HTTPS for Prometheus metrics endpoint.	false
`hostNetwork`	Enables host networking for the Trident controller. This is useful when you want to separate the frontend and backend traffic in a multi-home network.	false

For more information on formatting pod parameters, refer to Assigning Pods to Nodes.

Sample configurations

You can use the attributes in Configuration options when defining `TridentOrchestrator` to customize your installation.

Basic custom configuration

This example, created after running the `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml` command, represents a basic custom installation:

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

Node selectors

This example installs Trident with node selectors.

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

Windows worker nodes

This example, created after running the `cat deploy/crds/tridentorchestrator_cr.yaml` command, installs Trident on a Windows worker node.

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

Managed identities on an AKS cluster

This example installs Trident to enable managed identities on an AKS cluster.

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

Cloud identity on an AKS cluster

This example installs Trident for use with a cloud identity on an AKS cluster.

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

Cloud identity on an EKS cluster

This example installs Trident for use with a cloud identity on an AKS cluster.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/trident-role'"
```

Cloud identity for GKE

This example installs Trident for use with a cloud identity on a GKE cluster.

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

Kubernetes resource requests and limits configuration for Trident controller and Trident Linux node pods

This example configures Kubernetes resource requests and limits for Trident controller and Trident Linux node pods.

Disclaimer: The request and limit values provided in this example are for demonstration purposes only. Adjust these values based on your environment and workload requirements.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
        limits:
          cpu: 200m
          memory: 256Mi
      # sidecars
      csi-provisioner:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-attacher:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-resizer:
        requests:
          cpu: 3m
          memory: 20Mi
```

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

Kubernetes resource requests and limits configuration for Trident controller and Trident Windows and Linux node pods

This example configures Kubernetes resource requests and limits for Trident controller and Trident Windows and Linux node pods.

Disclaimer: The request and limit values provided in this example are for demonstration purposes only. Adjust these values based on your environment and workload requirements.

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

```
          cpu: 3m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-snapshotter:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      trident-autosupport:
        requests:
          cpu: 1m
          memory: 30Mi
        limits:
          cpu: 50m
          memory: 128Mi
    node:
      linux:
        trident-main:
          requests:
            cpu: 10m
            memory: 60Mi
          limits:
            cpu: 200m
            memory: 256Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 1m
            memory: 10Mi
          limits:
            cpu: 50m
            memory: 32Mi
      windows:
        trident-main:
          requests:
            cpu: 6m
            memory: 40Mi
          limits:
            cpu: 200m
            memory: 128Mi
        # sidecars
        node-driver-registrar:
```

```
      requests:
        cpu: 6m
        memory: 40Mi
      limits:
        cpu: 100m
        memory: 128Mi
    liveness-probe:
      requests:
        cpu: 2m
        memory: 40Mi
      limits:
        cpu: 50m
        memory: 64Mi
```

Install using tridentctl

Install using tridentctl

You can install Trident using `tridentctl`. This process applies to installations where the container images required by Trident are stored either in a private registry or not. To customize your `tridentctl` deployment, refer to Customize tridentctl deployment.

Critical information about Trident10

You must read the following critical information about Trident.

Critical information about Trident

- Kubernetes 1.27 is now supported in Trident. Upgrade Trident prior to upgrading Kubernetes.
- Trident strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in multipath.conf file.

  Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in multipath.conf file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Install Trident using `tridentctl`

Review the installation overview to ensure you've met installation prerequisites and selected the correct installation option for your environment.

Before you begin

Before you begin installation, log in to the Linux host and verify it is managing a working, supported Kubernetes cluster and that you have the necessary privileges.

With OpenShift, use `oc` instead of `kubectl` in all of the examples that follow, and log in as **system:admin** first by running `oc login -u system:admin` or `oc login -u kube-admin`.

1. Verify your Kubernetes version:

```
kubectl version
```

2. Verify cluster administrator privileges:

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

3. Verify you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network:

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

Step 1: Download the Trident installer package

The Trident installer package creates a Trident pod, configures the CRD objects that are used to maintain its state, and initializes the CSI sidecars to perform actions such as provisioning and attaching volumes to the cluster hosts. Download and extract the latest version of the Trident installer from the *Assets* section on GitHub. Update *<trident-installer-XX.XX.X.tar.gz>* in the example with your selected Trident version.

```
wget https://github.com/NetApp/trident/releases/download/v26.02.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

Step 2: Install Trident

Install Trident in the desired namespace by executing the `tridentctl install` command. You can add additional arguments to specify image registry location.

Standard mode

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

Images in one registry

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:26.02 --trident
-image <your-registry>/trident:26.02.0
```

Images in different registries

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:26.02 --trident
-image <your-registry>/trident:26.02.0
```

Your installation status should look something like this.

```
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=26.10.0
INFO Trident installation succeeded.
....
```

Verify the installation

You can verify your installation using pod creation status or `tridentctl`.

Using pod creation status

You can confirm if the Trident installation completed by reviewing the status of the created pods:

```
kubectl get pods -n trident

NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-679648bd45-cv2mx   6/6     Running   0          5m29s
trident-node-linux-vgc8n              2/2     Running   0          5m29s
```

If the installer does not complete successfully or `trident-controller-<generated id>` (`trident-csi-<generated id>` in versions prior to 23.01) does not have a **Running** status, the platform was not installed. Use `-d` to turn on debug mode and troubleshoot the issue.

Using `tridentctl`

You can use `tridentctl` to check the version of Trident installed.

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.02.0        | 26.02.0        |
+----------------+----------------+
```

Sample configurations

The following examples provide sample configurations for installing Trident using `tridentctl`.

Windows nodes

To enable Trident to run on Windows nodes:

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

Force detach

For information, see Automating the failover of stateful applications with Trident.

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

Enable concurrent Trident controller operations

To enable concurrent Trident controller operations for improved throughput, add the `--enable-concurrency` option during the installation as shown in this example.

> **Tech Preview**: This feature is experimental and currently supports limited parallel workflows with the ONTAP-NAS (NFS only) and ONTAP-SAN (NVMe for unified ONTAP 9) drivers, in addition to the existing tech preview for the ONTAP-SAN driver (iSCSI and FCP protocols in unified ONTAP 9).

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

Customize tridentctl installation

You can use the Trident installer to customize installation.

Learn about the installer

The Trident installer enables you to customize attributes. For example, if you have copied the Trident image to a private repository, you can specify the image name by using `--trident-image`. If you have copied the Trident image as well as the needed CSI sidecar images to a private repository, it might be preferable to specify the location of that repository by using the `--image-registry` switch, which takes the form `<registry FQDN>[:port]`.

> When installing Trident in a private repository, if you are using the `--image-registry` switch to specify the repository location, do not use `/netapp/` in the repository path. For example: `./tridentctl install --image-registry <image-registry> -n <namespace>`

If you are using a distribution of Kubernetes, where `kubelet` keeps its data on a path other than the usual `/var/lib/kubelet`, you can specify the alternate path by using `--kubelet-dir`.

If you need to customize the installation beyond what the installer's arguments allow, you can also customize the deployment files. Using the `--generate-custom-yaml` parameter creates the following YAML files in the installer's `setup` directory:

- `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`

*

After you have generated these files, you can modify them according to your needs and then use `--use -custom-yaml` to install your custom deployment.

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

Install using OpenShift certified operator

Install Trident using OpenShift OperatorHub

If you use Red Hat OpenShift, you can install NetApp Trident using the Red Hat certified operator. Use this procedure to install Trident from the Red Hat OpenShift Container Platform.

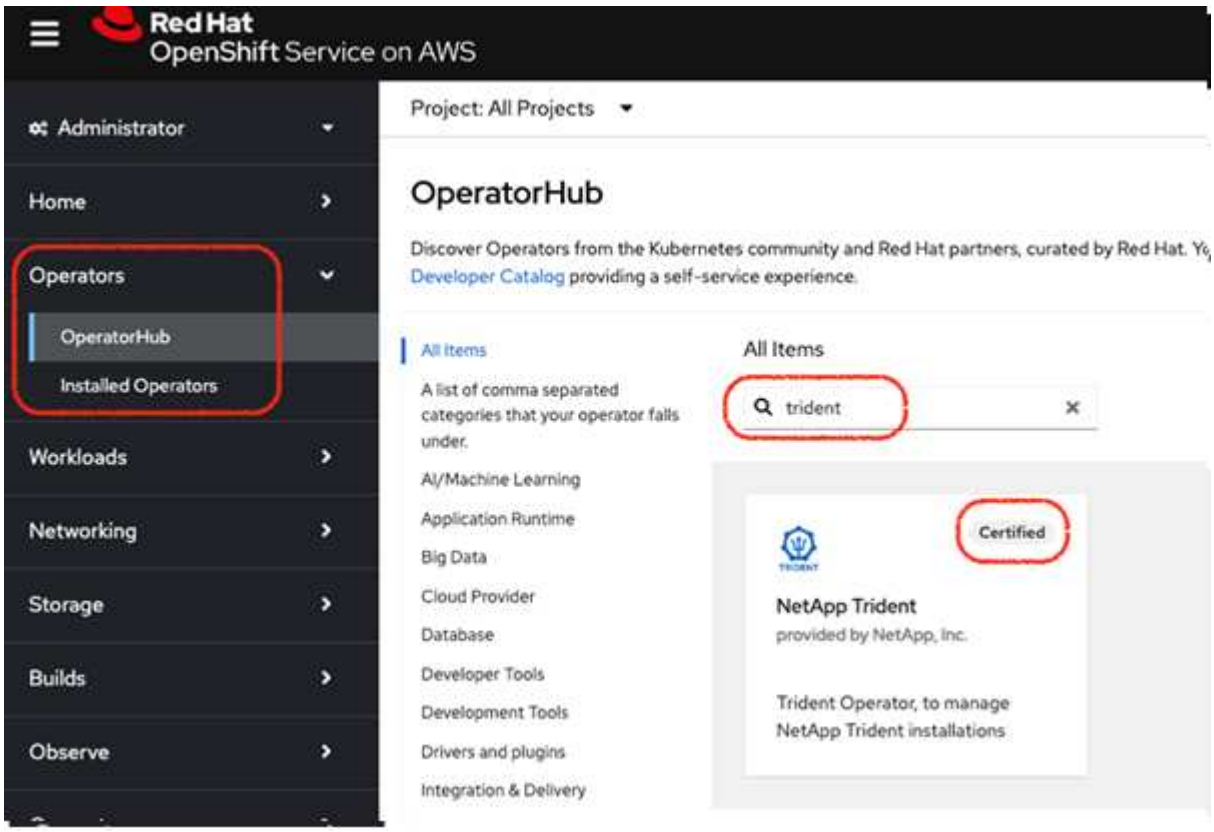
Before you begin

Before you begin the installation, prepare your environment for Trident installation.

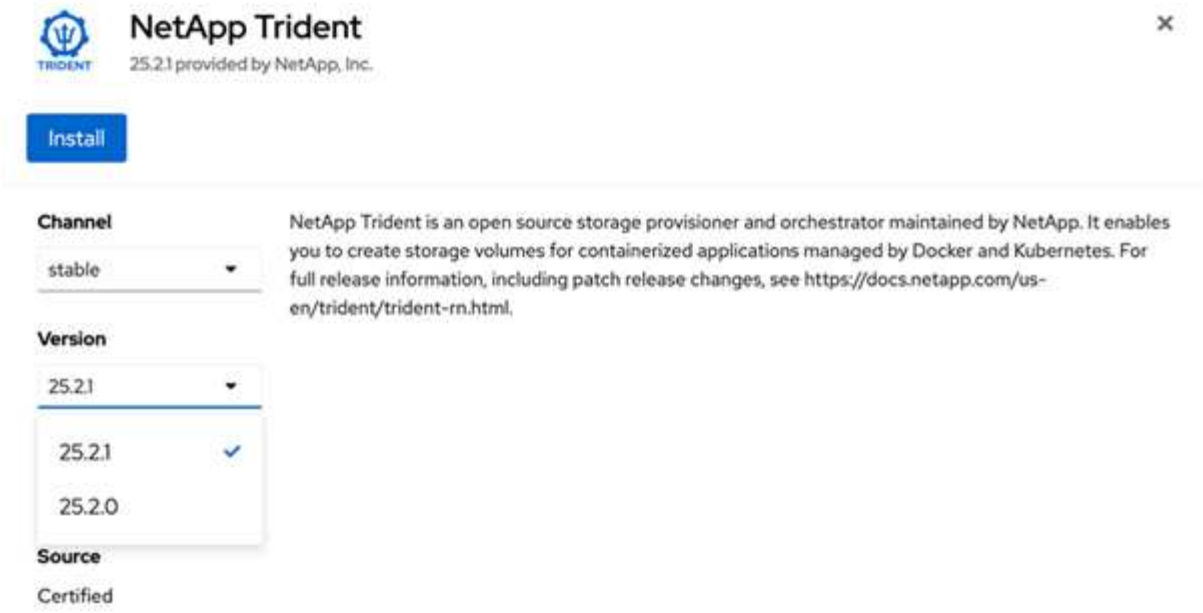
Find and install the Trident operator

Steps

1. Navigate to OpenShift OperatorHub and search for NetApp Trident.

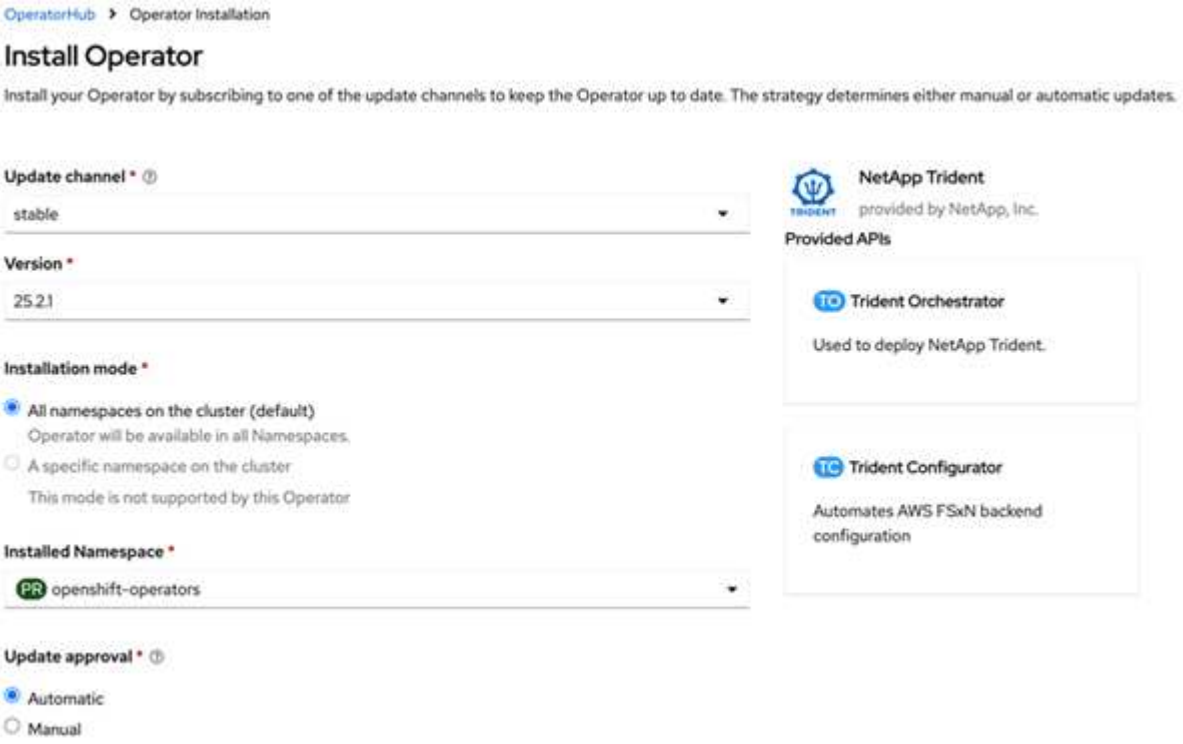


2. Click **NetApp Trident** to open the installation settings.
3. Select the required options and click **Install** to open the Operator configuration.

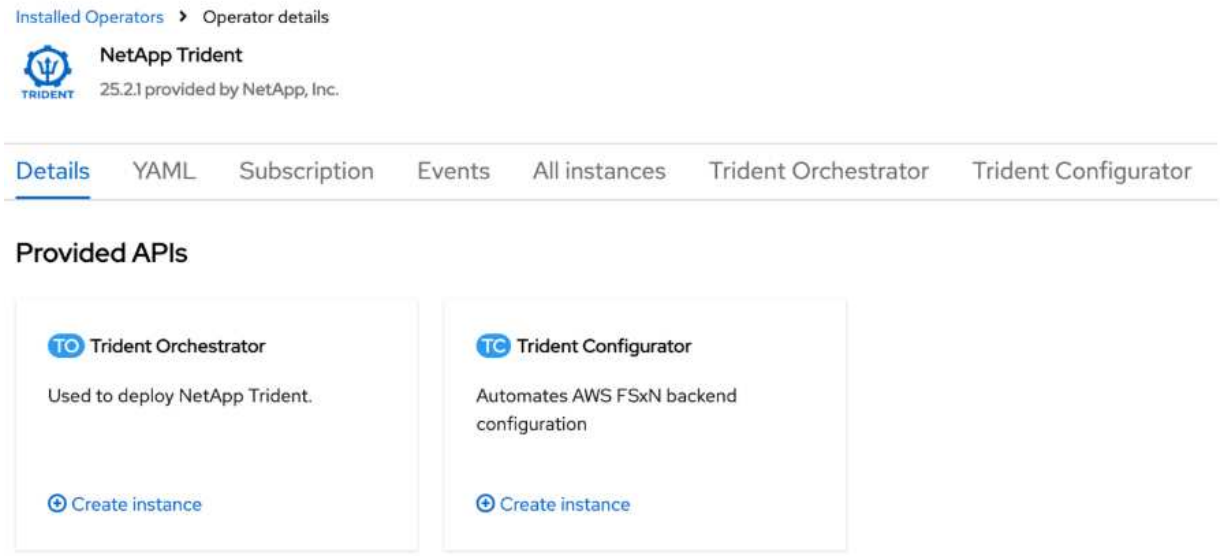


Make sure that you select the most recent Operator version.

4. Retain all the parameters as they are and click **Install**.



5. Click **View Operator** to view the details of the Operator.



6. Click **YAML view** and paste the following in the form:

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

The UI provides a default example. You can edit it directly instead of copying a full configuration.

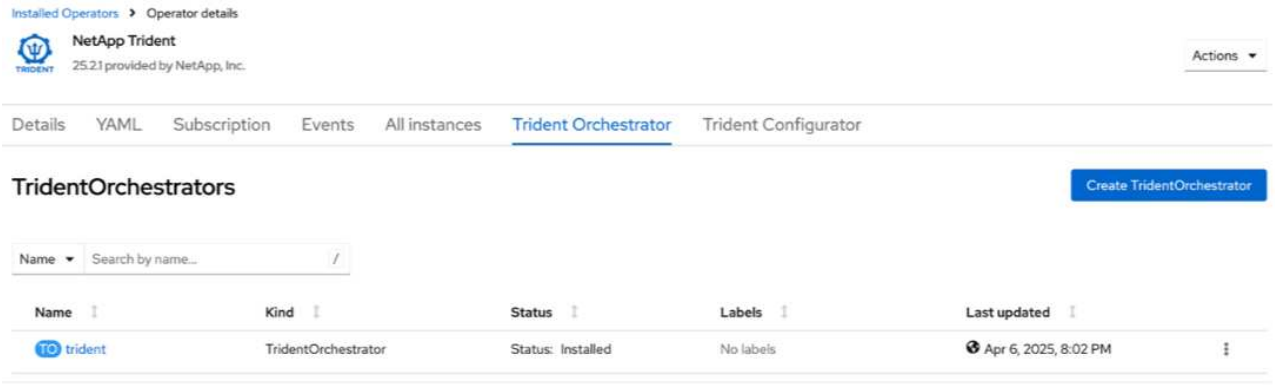
Optional: Enable concurrency

To enable concurrency, add the following field to the spec:

```
enableConcurrency: true
```

- Red Hat Enterprise Linux CoreOS (RHCOS) does not have iSCSI enabled and configured.
- You can add the `nodePrep` parameter to configure and enable both iSCSI and Multipath services on all OpenShift worker nodes.
- Beginning with OpenShift 4.19, the minimum Trident version supported for this feature is 25.06.1.

1. Click **Create**; the Trident Orchestrator will be fully installed.



Uninstall Trident operator

Steps

1. Select the Trident operator from the list of installed operator.
2. Select if you want to delete all the operand instance from the operator.

If you do not select the **Delete all operand instances from this operator** checkbox, Trident will not be uninstalled.

3. Click **Uninstall**.

Switch to the OpenShift certified Trident operator

You can switch to the Red Hat OpenShift certified Trident operator from the community operator, a Helm-based installation, or a manually deployed operator. The process for each method involves uninstalling the existing operator and then installing the certified operator using the OperatorHub.

Before you begin

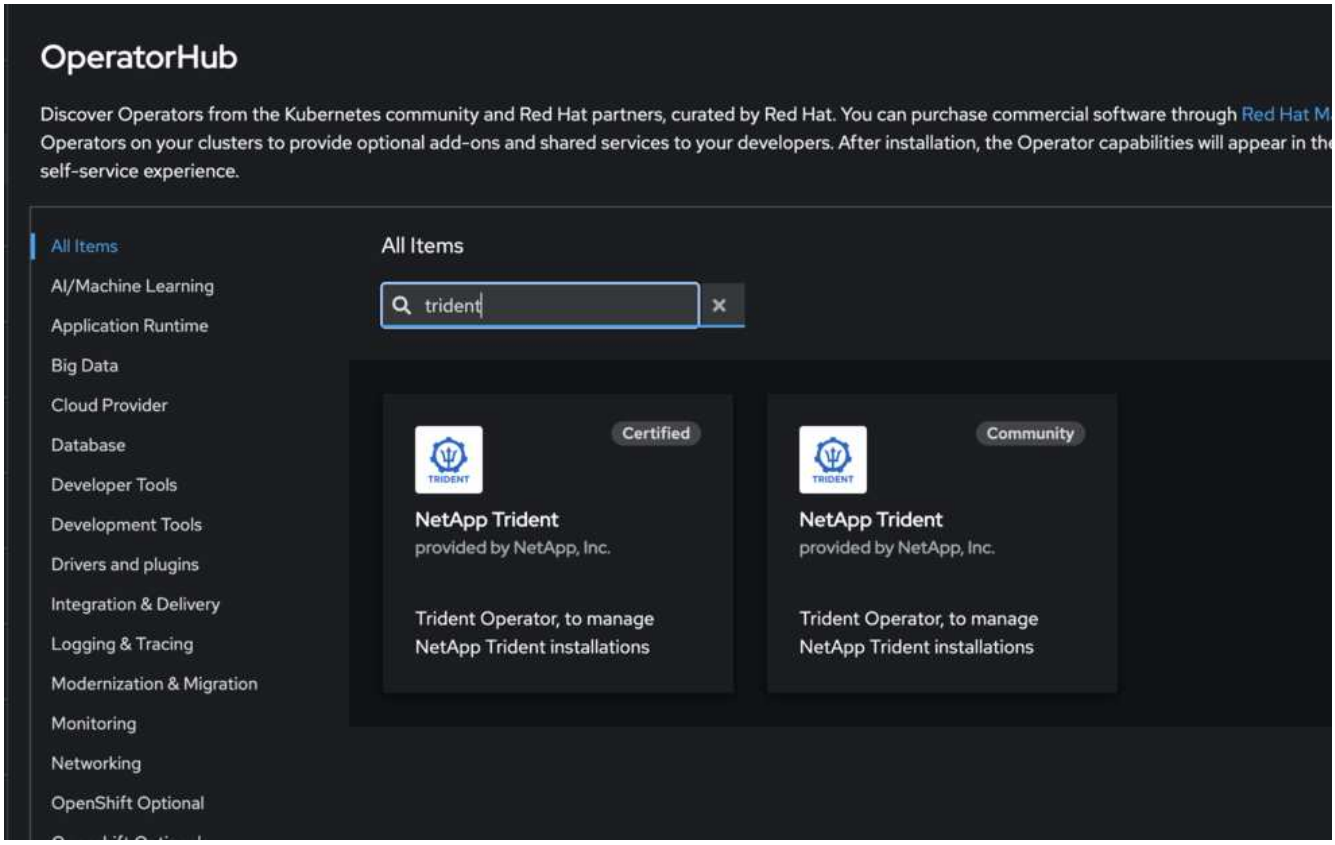
Before you begin the installation, prepare your environment for Trident installation.

Do not delete the `TridentOrchestrator` custom resource (CR) during the uninstall process. The `TridentOrchestrator` CR preserves your backend and storage class configuration, which is required after you install the certified operator.

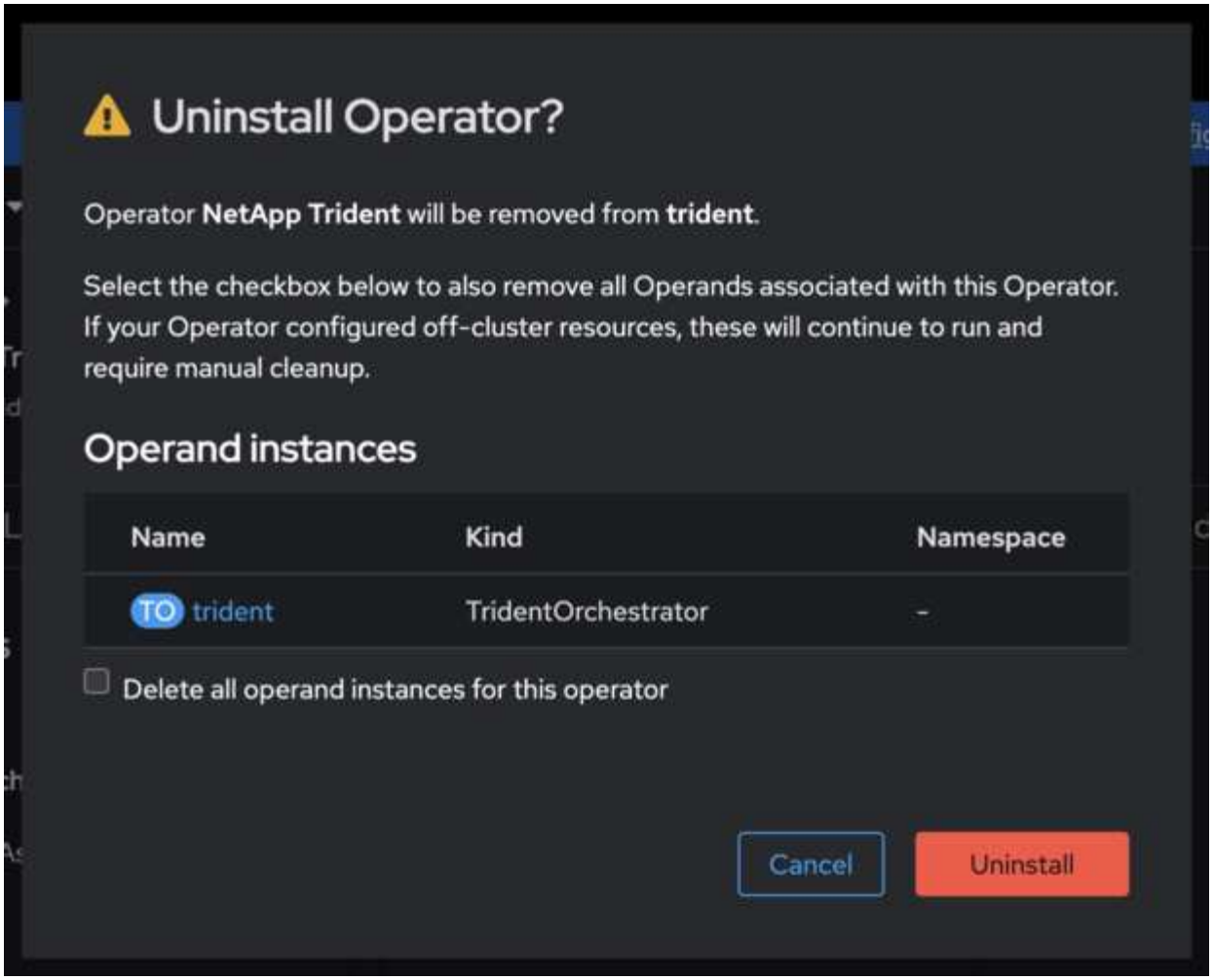
Switch from the community operator

Steps

1. Use the OpenShift console to navigate to the OperatorHub.



2. Find the NetApp Trident community operator.



Do not select **Delete all operand instances from this operator**.

3. Click **Uninstall**.
4. After the uninstall completes, proceed to Install the OpenShift certified operator.

Switch from a Helm-based operator installation

Steps

1. List the Helm release for your Trident installation:

```
helm ls -n trident
```

2. Uninstall the Helm release:

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

3. After the uninstall completes, proceed to Install the OpenShift certified operator.

Switch from a manually deployed operator

If you installed Trident by manually deploying the operator using a `bundle.yaml` from the installer package, remove it by deleting the same manifest.

Steps

1. Delete the operator deployment using the bundle manifest:

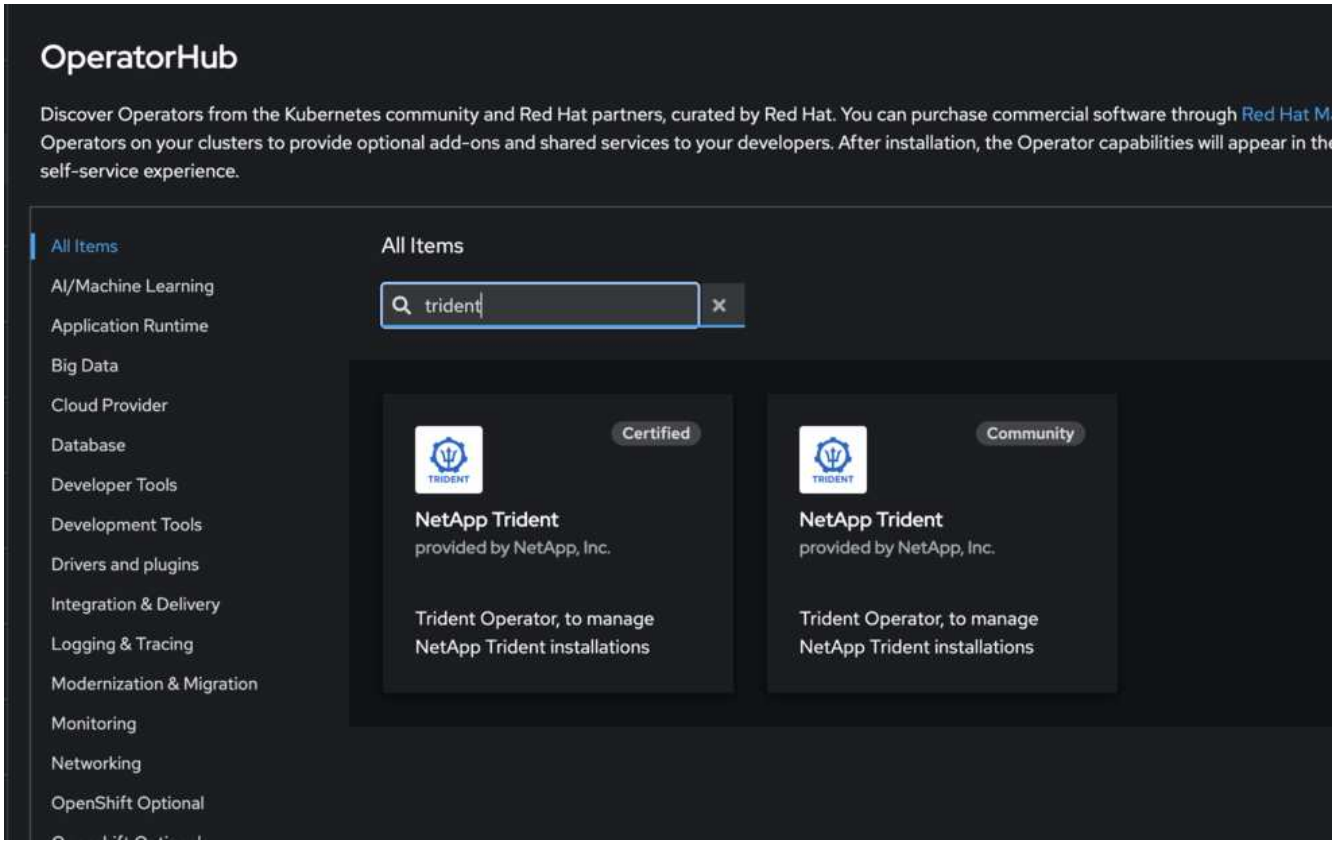
```
kubectl delete -f deploy/bundle.yaml -n trident
```

2. After the uninstall completes, proceed to Install the OpenShift certified operator.

Install the OpenShift certified operator

Steps

1. Navigate to the Red Hat OperatorHub.
2. Search for and select the NetApp Trident Operator.



3. Follow the on-screen instructions to install the operator.

Verification

- Check the OperatorHub in the console to ensure the new certified operator has been installed successfully.

Use Trident

Prepare the worker node

All worker nodes in the Kubernetes cluster must be able to mount the volumes you have provisioned for your pods. To prepare the worker nodes, you must install NFS, iSCSI, NVMe/TCP, or FC tools based on your driver selection.

Selecting the right tools

If you are using a combination of drivers, you should install all required tools for your drivers. Recent versions of Red Hat Enterprise Linux CoreOS (RHCOS) have the tools installed by default.

NFS tools

Install the NFS tools if you are using: `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, or `azure-netapp-files`.

iSCSI tools

Install the iSCSI tools if you are using: `ontap-san`, `ontap-san-economy`, `solidfire-san`.

NVMe tools

Install the NVMe tools if you are using `ontap-san` for nonvolatile memory express (NVMe) over TCP (NVMe/TCP) protocol.

> NetApp recommends ONTAP 9.12 or later for NVMe/TCP.

SCSI over FC tools

Refer to Ways to configure FC & FC-NVMe SAN hosts for more information about configuring your FC and FC-NVMe SAN hosts.

Install the FC tools if you are using `ontap-san` with sanType `fcp` (SCSI over FC).

Points to consider:
* SCSI over FC is supported on OpenShift and KubeVirt environments.
* SCSI over FC is not supported on Docker.
* iSCSI self healing is not applicable to SCSI over FC.

SMB tools

Prepare to provision SMB volumes if you are using: `ontap-nas` to provision SMB volumes.

Node service discovery

Trident attempts to automatically detect if the node can run iSCSI or NFS services.

> Node service discovery identifies discovered services but does not guarantee services are properly configured. Conversely, the absence of a discovered service does not guarantee the volume mount will fail.

Review events

Trident creates events for the node to identify the discovered services. To review these events, run:

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

Review discovered services

Trident identifies services enabled for each node on the Trident node CR. To view the discovered services, run:

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

NFS volumes

Install the NFS tools using the commands for your operating system. Ensure the NFS service is started up during boot time.

RHEL 8+

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

Reboot your worker nodes after installing the NFS tools to prevent failure when attaching volumes to containers.

iSCSI volumes

Trident can automatically establish an iSCSI session, scan LUNs, and discover multipath devices, format them, and mount them to a pod.

iSCSI self-healing capabilities

For ONTAP systems, Trident runs iSCSI self-healing every five minutes to:

1. **Identify** the desired iSCSI session state and the current iSCSI session state.
2. **Compare** the desired state to the current state to identify needed repairs. Trident determines repair priorities and when to preempt repairs.
3. **Perform repairs** required to return the current iSCSI session state to the desired iSCSI session state.

Logs of self-healing activity are located in the `trident-main` container on the respective Daemonset pod. To view logs, you must have set `debug` to "true" during Trident installation.

Trident iSCSI self-healing capabilities can help prevent:

- Stale or unhealthy iSCSI sessions that could occur after a network connectivity issue. In the case of a stale session, Trident waits seven minutes before logging out to reestablish the connection with a portal.

> For example, if CHAP secrets were rotated on the storage controller and the network loses connectivity, the old (*stale*) CHAP secrets could persist. Self-healing can recognize this and automatically reestablish the session to apply the updated CHAP secrets.

- Missing iSCSI sessions
- Missing LUNs

Points to consider before upgrading Trident

- If only per-node igroups (introduced in 23.04+) are in use, iSCSI self-healing will initiate SCSI rescans for all devices in the SCSI bus.
- If only backend-scoped igroups (deprecated as of 23.04) are in use, iSCSI self-healing will initiate SCSI rescans for exact LUN IDs in the SCSI bus.
- If a mix of per-node igroups and backend-scoped igroups are in use, iSCSI self-healing will initiate SCSI rescans for exact LUN IDs in the SCSI bus.

Install the iSCSI tools

Install the iSCSI tools using the commands for your operating system.

Before you begin

- Each node in the Kubernetes cluster must have a unique IQN. **This is a necessary prerequisite**.
- If using RHCOS version 4.5 or later, or other RHEL-compatible Linux distribution, with the `solidfire-san` driver and Element OS 12.5 or earlier, ensure that the CHAP authentication algorithm is set to MD5 in `/etc/iscsi/iscsid.conf`. Secure FIPS-compliant CHAP algorithms SHA1, SHA-256, and SHA3-256 are available with Element 12.7.

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

- When using worker nodes that run RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) with iSCSI PVs, specify the `discard` mountOption in the StorageClass to perform inline space reclamation. Refer to Red Hat documentation.
- Ensure that you have upgraded to the latest version of the `multipath-tools`.

RHEL 8+

1. Install the following system packages:

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

2. Check that iscsi-initiator-utils version is 6.2.0.874-2.el7 or later:

```
rpm -q iscsi-initiator-utils
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `iscsid` and `multipathd` are running:

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

6. Enable and start `iscsi`:

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. Install the following system packages:

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

2. Check that open-iscsi version is 2.0.874-5ubuntu2.10 or later (for bionic) or 2.0.874-7.1ubuntu6.1 or later (for focal):

```
dpkg -l open-iscsi
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
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
```

Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `open-iscsi` and `multipath-tools` are enabled and running:

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

For Ubuntu 18.04, you must discover target ports with `iscsiadm` before starting `open-iscsi` for the iSCSI daemon to start. You can alternatively modify the `iscsi` service to start `iscsid` automatically.

Configure or disable iSCSI self healing

You can configure the following Trident iSCSI self-healing settings to fix stale sessions:

- **iSCSI self-healing interval**: Determines the frequency at which iSCSI self-healing is invoked (default: 5 minutes). You can configure it to run more frequently by setting a smaller number or less frequently by setting a larger number.

Setting the iSCSI self-healing interval to 0 stops iSCSI self-healing completely. We do not recommend disabling iSCSI Self-healing; it should only be disabled in certain scenarios when iSCSI self-healing is not working as intended or for debugging purposes.

- **iSCSI Self-Healing Wait Time**: Determines the duration iSCSI self-healing waits before logging out of an unhealthy session and trying to log in again (default: 7 minutes). You can configure it to a larger number so that sessions that are identified as unhealthy have to wait longer before being logged out and then an attempt is made to log back in, or a smaller number to log out and log in earlier.

Helm

To configure or change iSCSI self-healing settings, pass the `iscsiSelfHealingInterval` and `iscsiSelfHealingWaitTime` parameters during the helm installation or helm update.

The following example sets the iSCSI self-healing interval to 3 minutes and self-healing wait time to 6 minutes:

```
helm install trident trident-operator-100.2602.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

tridentctl

To configure or change iSCSI self-healing settings, pass the `iscsi-self-healing-interval` and `iscsi-self-healing-wait-time` parameters during the tridentctl installation or update.

The following example sets the iSCSI self-healing interval to 3 minutes and self-healing wait time to 6 minutes:

```
tridentctl install --iscsi-self-healing-interval=3m0s --iscsi-self
-healing-wait-time=6m0s -n trident
```

NVMe/TCP volumes

Install the NVMe tools using the commands for your operating system.

- NVMe requires RHEL 9 or later.
- If the kernel version of your Kubernetes node is too old or if the NVMe package is not available for your kernel version, you might have to update the kernel version of your node to one with the NVMe package.

RHEL 9

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

Ubuntu

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

Verify installation

After installation, verify that each node in the Kubernetes cluster has a unique NQN using the command:

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

Trident modifies the `ctrl_device_tmo` value to ensure NVMe doesn't give up on the path if it goes down. Do not change this setting.

SCSI over FC volumes

You can now use the Fibre Channel (FC) protocol with Trident to provision and manage storage resources on ONTAP system.

Prerequisites

Configure the required network and node settings for FC.

Network settings

1. Get the WWPN of the target interfaces. Refer to network interface show for more information.
2. Get the WWPN for the interfaces on initiator (Host).

   Refer to the corresponding host operating system utilities.

3. Configure zoning on the FC switch using WWPNs of the Host and target.

   Refer to the respecive switch vendor documentation for information.

   Refer to the following ONTAP documentation for details:

   - Fibre Channel and FCoE zoning overview
   - Ways to configure FC & FC-NVMe SAN hosts

Install the FC tools

Install the FC tools using the commands for your operating system.

- When using worker nodes that run RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) with FC PVs, specify the `discard` mountOption in the StorageClass to perform inline space reclamation. Refer to Red Hat documentation.

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipathd` is running:

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

Ubuntu

1. Install the following system packages:

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

2. Enable multipathing:

```
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
```

> Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipath-tools` is enabled and running:

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

Prepare to provision SMB volumes

You can provision SMB volumes using `ontap-nas` drivers.

> You must configure both NFS and SMB/CIFS protocols on the SVM to create an `ontap-nas-economy` SMB volume for ONTAP on-premises clusters. Failure to configure either of these protocols will cause SMB volume creation to fail.

> `autoExportPolicy` is not supported for SMB volumes.

Before you begin

Before you can provision SMB volumes, you must have the following.

- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials. To generate secret `smbcreds`:

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

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Steps

1. For on-premises ONTAP, you can optionally create an SMB share or Trident can create one for you.

   > SMB shares are required for Amazon FSx for ONTAP.

   You can create the SMB admin shares in one of two ways either using the Microsoft Management Console Shared Folders snap-in or using the ONTAP CLI. To create the SMB shares using the ONTAP CLI:

   a. If necessary, create the directory path structure for the share.

      The `vserver cifs share create` command checks the path specified in the -path option during share creation. If the specified path does not exist, the command fails.

   b. Create an SMB share associated with the specified SVM:

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

   c. Verify that the share was created:

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

Refer to Create an SMB share for full details.

2. When creating the backend, you must configure the following to specify SMB volumes. For all FSx for ONTAP backend configuration options, refer to FSx for ONTAP configuration options and examples.

Parameter	Description	Example
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`
`nasType`	**Must set to `smb`.** If null, defaults to `nfs`.	`smb`
`securityStyle`	Security style for new volumes. **Must be set to `ntfs` or `mixed` for SMB volumes.**	`ntfs` or `mixed` for SMB volumes
`unixPermissions`	Mode for new volumes. **Must be left empty for SMB volumes.**	""

Configure and manage backends

Configure backends

A backend defines the relationship between Trident and a storage system. It tells Trident how to communicate with that storage system and how Trident should provision volumes from it.

Trident automatically offers up storage pools from backends that match the requirements defined by a storage class. Learn how to configure the backend for your storage system.

- Configure an Azure NetApp Files backend
- Configure a Google Cloud NetApp Volumes backend
- Configure a NetApp HCI or SolidFire backend
- Configure a backend with ONTAP or Cloud Volumes ONTAP NAS drivers
- Configure a backend with ONTAP or Cloud Volumes ONTAP SAN drivers
- Use Trident with Amazon FSx for NetApp ONTAP

Azure NetApp Files

Configure an Azure NetApp Files backend

Use Azure NetApp Files as a backend for Trident.
This backend supports NFS and SMB volumes.
Trident supports managed identities and workload identity for Azure Kubernetes Service (AKS) clusters.

Supported Azure cloud environments

Trident supports Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When you deploy Trident or configure an Azure NetApp Files backend, ensure that Azure Resource Manager and authentication endpoints match your Azure cloud environment.

Review Azure NetApp Files driver support

Trident provides the following Azure NetApp Files storage driver.

Supported access modes include *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), and *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`azure-netapp-files`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	`nfs`, `smb`

Review considerations

- Azure NetApp Files does not support volumes smaller than 50 GiB.
  Trident creates a 50-GiB volume when a smaller volume is requested.
- Trident supports SMB volumes mounted to pods running on Windows nodes only.
- Azure NetApp Files deployments in non-Commercial Azure clouds require cloud-specific Azure Resource Manager and authentication endpoints.
  Ensure that Trident and any backend configuration use the endpoints appropriate for your Azure cloud environment.

Use managed identities for AKS

Trident supports managed identities for AKS clusters.

If you use `tridentctl` to create or manage Azure NetApp Files backends, ensure that it is configured for the correct Azure cloud environment.

To use managed identities, you must have:

- A Kubernetes cluster deployed using AKS

- Managed identities configured on the AKS Kubernetes cluster
- Trident installed with `cloudProvider` set to `"Azure"`

Trident operator

Edit `tridentorchestrator_cr.yaml` and set `cloudProvider` to `"Azure"`.

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

Helm

The following example installs Trident and sets `cloudProvider` using the environment variable `$CP`:

```
helm install trident trident-operator-100.2602.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

`tridentctl`

The following example installs Trident and sets the `cloud-provider` flag to `Azure`:

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

Use workload identity for AKS

Workload identity enables Kubernetes pods to access Azure resources by authenticating as a workload identity.

If you use `tridentctl` to create or manage Azure NetApp Files backends, ensure that it is configured for the correct Azure cloud environment.

To use workload identity, you must have:

- A Kubernetes cluster deployed using AKS
- Workload identity and oidc-issuer configured on the AKS Kubernetes cluster
- Trident installed with `cloudProvider` set to `"Azure"` and `cloudIdentity` set to the workload identity value

Trident operator

Edit `tridentorchestrator_cr.yaml` and set `cloudProvider` to `"Azure"`.
Set `cloudIdentity` to `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`.

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

Helm

Set the values for the **cloud-provider (CP)** and **cloud-identity (CI)** flags using the following environment variables:

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

The following example installs Trident and sets `cloudProvider` using `$CP` and sets `cloudIdentity` using `$CI`:

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

`tridentctl`

Set the values for the **cloud provider** and **cloud identity** flags using the following environment variables:

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

The following example installs Trident and sets `cloud-provider` to `$CP` and `cloud-identity` to `$CI`:

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

Prepare to configure an Azure NetApp Files backend

Before you can configure your Azure NetApp Files backend, you need to ensure the following requirements are met.

Supported Azure cloud environments

Trident supports Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When preparing your environment, ensure that your Azure subscription, identity configuration, and Azure NetApp Files resources are created in the appropriate Azure cloud environment.

Prerequisites for NFS and SMB volumes

If you are using Azure NetApp Files for the first time or in a new location, some initial configuration is required to set up Azure NetApp Files and create an NFS volume. Refer to Azure: Set up Azure NetApp Files and create an NFS volume.

To configure and use an Azure NetApp Files backend, you need the following:

- `subscriptionID`, `tenantID`, `clientID`, `location`, and `clientSecret` are optional when using managed identities on an AKS cluster.
- `tenantID`, `clientID`, and `clientSecret` are optional when using a cloud identity on an AKS cluster.
- Azure NetApp Files deployments in non-Commercial Azure clouds require cloud-specific Azure Resource Manager and authentication endpoints. Ensure that Trident and any backend configuration use the endpoints appropriate for your Azure cloud environment.

- A capacity pool. Refer to Microsoft: Create a capacity pool for Azure NetApp Files.
- A subnet delegated to Azure NetApp Files. Refer to Microsoft: Delegate a subnet to Azure NetApp Files.
- `subscriptionID` from an Azure subscription with Azure NetApp Files enabled.
- `tenantID`, `clientID`, and `clientSecret` from an App Registration in Azure Active Directory with sufficient permissions to the Azure NetApp Files service. The App Registration should use either:
  - The Owner or Contributor role predefined by Azure.
  - A custom Contributor role at the subscription level (`assignableScopes`) with the following permissions that are limited to only what Trident requires. After creating the custom role, assign the role using the Azure portal.

Custom contributor role

```
{
  "id": "/subscriptions/<subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-
definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",

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

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

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

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

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

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

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

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

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

```
            "Microsoft.Features/features/read",
            "Microsoft.Features/operations/read",
            "Microsoft.Features/providers/features/read",

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

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

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

- The Azure `location` that contains at least one delegated subnet. As of Trident 22.01, the `location` parameter is a required field at the top level of the backend configuration file. Location values specified in virtual pools are ignored.
- To use `Cloud Identity`, get the `client ID` from a user-assigned managed identity and specify that ID in `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`.

Additional requirements for SMB volumes

To create an SMB volume, you must have:

- Active Directory configured and connected to Azure NetApp Files. Refer to Microsoft: Create and manage Active Directory connections for Azure NetApp Files.
- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials so Azure NetApp Files can authenticate to Active Directory. To generate secret `smbcreds`:

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

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Azure NetApp Files backend configuration options and examples

Learn about NFS and SMB backend configuration options for Azure NetApp Files and review configuration examples.

Backend configuration options

Trident uses your backend configuration (subnet, virtual network, service level, and location) to create Azure NetApp Files volumes on capacity pools that are available in the requested location and match the requested service level and subnet.

Azure NetApp Files backends provide these configuration options.

Parameter	Description	Default
`version`	Backend configuration version.	Always 1
`storageDriverName`	Name of the storage driver	"azure-netapp-files"
`backendName`	Custom name for the storage backend	Driver name + "_" + random characters
`subscriptionID`	The subscription ID from your Azure subscription Optional when managed identities is enabled on an AKS cluster.	
`tenantID`	The tenant ID from an App Registration Optional when managed identities or cloud identity is used on an AKS cluster.	
`clientID`	The client ID from an App Registration Optional when managed identities or cloud identity is used on an AKS cluster.	
`clientSecret`	The client secret from an App Registration Optional when managed identities or cloud identity is used on an AKS cluster.	
`serviceLevel`	One of `Standard`, `Premium`, or `Ultra`	"" (random)
`location`	Name of the Azure location where the new volumes will be created Optional when managed identities is enabled on an AKS cluster.	
`resourceGroups`	List of resource groups for filtering discovered resources	"[]" (no filter)
`netappAccounts`	List of NetApp accounts for filtering discovered resources	"[]" (no filter)

Parameter	Description	Default
`capacityPools`	List of capacity pools for filtering discovered resources	"[]" (no filter, random)
`virtualNetwork`	Name of a virtual network with a delegated subnet	""
`subnet`	Name of a subnet delegated to `Microsoft.Netapp/volumes`	""
`networkFeatures`	Set of VNet features for a volume, may be `Basic` or `Standard`. Network Features is not available in all regions and might have to be enabled in a subscription. Specifying `networkFeatures` when the functionality is not enabled causes volume provisioning to fail.	""
`nfsMountOptions`	Fine-grained control of NFS mount options. Ignored for SMB volumes. To mount volumes using NFS version 4.1, include `nfsvers=4` in the comma-delimited mount options list to choose NFS v4.1. Mount options set in a storage class definition override mount options set in backend configuration.	"nfsvers=3"
`limitVolumeSize`	Fail provisioning if the requested volume size is above this value	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `\{"api": false, "method": true, "discovery": true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes.	`nfs`

Parameter	Description	Default
`supportedTopologies`	Represents a list of regions and zones that are supported by this backend. For more information, refer to Use CSI Topology.	
`qosType`	Represents the QoS type: Auto or Manual.	Auto
`maxThroughput`	Sets the maximum throughput allowed in MiB/sec. Supported only for manual QoS capacity pools.	`4 MiB/sec`

For more information on Network Features, refer to Configure network features for an Azure NetApp Files volume.

Consider Azure cloud environments (26.02)

Starting with the 26.02 release, Trident supports creating and managing Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When you deploy Trident or create an Azure NetApp Files backend, ensure that Azure Resource Manager and authentication endpoints match your Azure cloud environment.
If the endpoints do not match, `tridentctl` cannot authenticate and backend creation fails.

Required permissions and resources

If you receive a "No capacity pools found" error when creating a PVC, it is likely your app registration doesn't have the required permissions and resources (subnet, virtual network, capacity pool) associated.
If debug is enabled, Trident logs the Azure resources discovered when the backend is created.
Verify an appropriate role is being used.

The values for `resourceGroups`, `netappAccounts`, `capacityPools`, `virtualNetwork`, and `subnet` can be specified using short or fully-qualified names.
Fully-qualified names are recommended in most situations as short names can match multiple resources with the same name.

If the vNet is located in a different resource group from the Azure NetApp Files (ANF) storage account, specify the resource group for the virtual network while configuring the resourceGroups list for the backend.

The `resourceGroups`, `netappAccounts`, and `capacityPools` values are filters that restrict the set of discovered resources to those available to this storage backend and may be specified in any combination.
Fully-qualified names follow this format:

Type	Format
Resource group	<resource group>
NetApp account	<resource group>/<netapp account>
Capacity pool	<resource group>/<netapp account>/<capacity pool>
Virtual network	<resource group>/<virtual network>
Subnet	<resource group>/<virtual network>/<subnet>

Volume provisioning

You can control default volume provisioning by specifying the following options in a special section of the configuration file.
Refer to Example configurations for details.

Parameter	Description	Default
`exportRule`	Export rules for new volumes. `exportRule` must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation. Ignored for SMB volumes.	"0.0.0.0/0"
`snapshotDir`	Access to the `.snapshot` directory	`true`, `false` (Set explicitly).
`size`	The default size of new volumes	"100G"
`unixPermissions`	The unix permissions of new volumes (4 octal digits). Ignored for SMB volumes.	"" (preview feature, requires whitelisting in subscription)

Example configurations

The following examples show basic configurations that leave most parameters to default.
This is the easiest way to define a backend.

Minimal configuration

This is the absolute minimum backend configuration.
With this configuration, Trident discovers all of your NetApp accounts, capacity pools, and subnets delegated to Azure NetApp Files in the configured location, and places new volumes on one of those pools and subnets randomly.
Because `nasType` is omitted, the `nfs` default applies and the backend will provision for NFS volumes.

This configuration is ideal when you are just getting started with Azure NetApp Files and trying things out, but in practice you are going to want to provide additional scoping for the volumes you provision.

```
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
  tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
  clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
  clientSecret: SECRET
  location: eastus
```

Managed identities for AKS

This backend configuration omits `subscriptionID`, `tenantID`, `clientID`, and `clientSecret`, which are optional when using managed identities.

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - resource-group-1/netapp-account-1/ultra-pool
  resourceGroups:
    - resource-group-1
  netappAccounts:
    - resource-group-1/netapp-account-1
  virtualNetwork: resource-group-1/eastus-prod-vnet
  subnet: resource-group-1/eastus-prod-vnet/eastus-anf-subnet
```

Cloud identity for AKS

This backend configuration omits `tenantID`, `clientID`, and `clientSecret`, which are optional when using a cloud identity.

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
  location: eastus
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
```

Specific service level configuration with capacity pool filters

This backend configuration places volumes in Azure's `eastus` location in an `Ultra` capacity pool. Trident automatically discovers all of the subnets delegated to Azure NetApp Files in that location and places a new volume on one of them randomly.

```
---
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
```

Backend example with manual QoS capacity pools

This backend configuration places volumes in Azure's `eastus` location with manual QoS capacity pools.

```
---
version: 1
storageDriverName: azure-netapp-files
backendName: anf1
location: eastus
labels:
  clusterName: test-cluster-1
  cloud: anf
  nasType: nfs
defaults:
  qosType: Manual
storage:
  - serviceLevel: Ultra
    labels:
      performance: gold
    defaults:
      maxThroughput: 10
  - serviceLevel: Premium
    labels:
      performance: silver
    defaults:
      maxThroughput: 5
  - serviceLevel: Standard
    labels:
      performance: bronze
    defaults:
      maxThroughput: 3
```

Advanced configuration

This backend configuration further reduces the scope of volume placement to a single subnet, and also modifies some volume provisioning defaults.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
virtualNetwork: application-group-1/eastus-prod-vnet
subnet: application-group-1/eastus-prod-vnet/my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: "true"
  size: 200Gi
  unixPermissions: "0777"
```

Virtual pool configuration

This backend configuration defines multiple storage pools in a single file.
This is useful when you have multiple capacity pools supporting different service levels and you want to create storage classes in Kubernetes that represent those.
Virtual pool labels were used to differentiate the pools based on `performance`.

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

Supported topologies configuration

Trident facilitates provisioning of volumes for workloads based on regions and availability zones.
The `supportedTopologies` block in this backend configuration is used to provide a list of regions and zones per backend.
The region and zone values specified here must match the region and zone values from the labels on each Kubernetes cluster node.
These regions and zones represent the list of permissible values that can be provided in a storage class.
For storage classes that contain a subset of the regions and zones provided in a backend, Trident creates volumes in the mentioned region and zone.
For more information, refer to Use CSI Topology.

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

Storage class definitions

The following `StorageClass` definitions refer to the storage pools above.

Example definitions using `parameter.selector` field

Using `parameter.selector` you can specify for each `StorageClass` the virtual pool that is used to host a volume.
The volume will have the aspects defined in the chosen pool.

```
---
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
```

Example definitions for SMB volumes

Using `nasType`, `node-stage-secret-name`, and `node-stage-secret-namespace`, you can specify an SMB volume and provide the required Active Directory credentials.

Basic configuration on default namespace

```
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"
```

Using different secrets per namespace

```
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}
```

Using different secrets per volume

```
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` filters for pools which support SMB volumes.
`nasType: nfs` or `nasType: null` filters for NFS pools.

Create the backend

After you create the backend configuration file, run the following command:

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

If you use a non-Commercial Azure cloud, ensure that `tridentctl` is configured to use the Azure Resource Manager and authentication endpoints for your Azure cloud environment.
If the backend creation fails, check your backend configuration and view the logs to determine the cause:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can run the create command again.

Google Cloud NetApp Volumes

Configure Google Cloud NetApp Volumes

You can configure Google Cloud NetApp Volumes as a backend for Trident to provision storage for Kubernetes workloads.

Overview

Trident supports Google Cloud NetApp Volumes for both NAS (NFS and SMB) and block (iSCSI) workloads.

- NAS workloads use the `google-cloud-netapp-volumes` backend
- Block (iSCSI) workloads use the `google-cloud-netapp-volumes-san` backend

NAS volumes provide file-based storage and are accessed using NFS or SMB protocols. These volumes support shared access across multiple pods or nodes.

Block volumes provide raw block storage and are accessed as iSCSI devices attached to Kubernetes nodes. These volumes are used when applications require block-level access.

This applies to the following environments:

- Trident 26.02 and later
- Google Kubernetes Engine (GKE) or Red Hat OpenShift
- Google Cloud NetApp Volumes storage pools

To configure block (iSCSI) storage, see
Configure block storage (iSCSI).

Prepare to configure

Cloud identity enables Kubernetes workloads to access Google Cloud resources by authenticating as a workload identity instead of using static credentials.

To use cloud identity with Google Cloud NetApp Volumes, you must have:

- A Kubernetes cluster deployed using Google Kubernetes Engine (GKE)
- Workload identity enabled on the GKE cluster and the metadata server enabled on the node pools
- A Google Cloud service account with the Google Cloud NetApp Volumes Admin role (`roles/netapp.admin`) or an equivalent custom role
- Trident installed with the cloud provider set to `GCP` and the cloud identity annotation configured

Trident operator

To install Trident using the Trident operator, edit `tridentorchestrator_cr.yaml`:

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

Helm

Set the cloud provider and cloud identity when installing Trident with Helm:

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

tridentctl

Install Trident by specifying the cloud provider and cloud identity:

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

Configure NAS storage

For Google Cloud NetApp Volumes UNIFIED storage pools, Trident applies UNIFIED-specific naming and validation rules during volume operations.

When locating a volume, Trident can evaluate multiple compatible volume name variants (for example, hyphen and underscore formats) to improve import and discovery reliability.

Driver details

Trident provides the `google-cloud-netapp-volumes` driver to provision NAS storage from Google Cloud NetApp Volumes.

The driver supports the following access modes:

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

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`google-cloud-netapp-volumes`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	`nfs`, `smb`

Configure a Trident NAS backend

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

Provision NAS volumes

NAS volumes are provisioned using the `google-cloud-netapp-volumes` backend and support NFS and

SMB protocols.

StorageClass for NFS volumes

To provision NFS volumes, set `nasType` to `nfs`.

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

StorageClass for SMB volumes

To provision SMB volumes, set `nasType` to `smb` and provide credentials.

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

PersistentVolumeClaim example (RWX)

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

PersistentVolumeClaim example (RWO)

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

NAS volumes use `volumeMode: Filesystem`.

Configure Google Cloud NetApp Volumes for SAN workloads

You can configure Trident to provision block storage volumes using the iSCSI protocol from Google Cloud NetApp Volumes. SAN volumes are provisioned from Flex Unified storage pools by using the `google-cloud-netapp-volumes-san` storage driver.

This driver is dedicated to block workloads and does not support NAS protocols.

The `google-cloud-netapp-volumes-san` backend is required to provision iSCSI block volumes. The `google-cloud-netapp-volumes` backend supports NAS protocols only and cannot be used for SAN workloads.

Overview

Trident supports Google Cloud NetApp Volumes SAN (iSCSI) workloads using the `google-cloud-netapp-volumes-san` driver.

SAN volumes are provisioned from Flex Unified storage pools and presented to Kubernetes nodes as iSCSI block devices.

This applies to the following environments:

- Trident 26.02 and later
- Google Kubernetes Engine (GKE) or Red Hat OpenShift
- Google Cloud NetApp Volumes Flex Unified storage pools
- iSCSI-based workloads

Flex Unified storage pools

Flex Unified storage pools provide block storage using the iSCSI protocol and are required for SAN provisioning:

- Flex Unified REGIONAL pools are supported.
- Flex Unified ZONAL pools are supported starting with Trident 26.02.1.
- Only the **Flex** service level is supported for SAN workloads.

Configure a Trident SAN backend

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

Create a StorageClass

After configuring the SAN backend, create a StorageClass that references the `google-cloud-netapp-volumes-san` driver.

The filesystem type is defined in the StorageClass, not in the backend.

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

Supported filesystem types:

- `ext4` (default)
- `ext3`
- `xfs`

> The SAN driver supports only the Flex service level and does not use NAS-specific backend parameters such as `exportRule`, `unixPermissions`, `nasType`, `snapshotDir`, `nfsMountOptions`, or tiering-related settings.

Provision block volumes

ReadWriteOnce (RWO)

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

ReadWriteOncePod (RWOP)

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

ReadOnlyMany (ROX)

A common pattern for ROX is to clone an existing ReadWriteOnce volume and mount the clone as read-only.

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

ReadWriteMany (RWX) — raw block only

ReadWriteMany is supported only when `volumeMode: Block`.

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

Block volume behavior

Block volumes are provisioned as iSCSI LUNs and presented to Kubernetes nodes as block devices.

Block volumes:

- Use the iSCSI protocol
- Support filesystem and raw block presentation
- Are attached and managed by Trident
- Support multiple Kubernetes access modes

Access modes

Block volumes provisioned by Trident support the following access modes:

- `ReadWriteOnce` (RWO)
- `ReadOnlyMany` (ROX)
- `ReadWriteOncePod` (RWOP)
- `ReadWriteMany` (RWX), supported only when `volumeMode: Block`

volumeMode behavior

The `volumeMode` field controls how a block volume is exposed:

- `Filesystem`
  Trident formats and mounts the volume.
- `Block`
  Trident attaches the device and exposes it as a raw block device.

Supported operations

Block volumes provisioned using the `google-cloud-netapp-volumes-san` driver support:

- Create

- Delete
- Clone
- Snapshot
- Resize
- Import

Extra GiB overprovisioning behavior

Google Cloud NetApp Volumes block volumes include internal metadata overhead. This overhead reduces the kernel-visible device size compared to the provisioned capacity.

Testing shows:

- Approximately 300 KiB overhead on initial creation
- Up to approximately 107 MiB overhead after a resize

Because Google Cloud NetApp Volumes accepts only whole-GiB allocations, Trident ensures that the usable device size always meets or exceeds the PVC request by:

- Rounding the requested size up to the next whole GiB
- Adding an additional 1 GiB buffer

Example:

- PVC request: 100 GiB
- Provisioned size in Google Cloud NetApp Volumes: 101 GiB
- Usable space visible to the application: at least 100 GiB

Pod examples

Filesystem-mounted block volume (RWO)

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

Raw block device (RWX)

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

Attach and mount behavior

For SAN volumes provisioned from Google Cloud NetApp Volumes:

- Trident creates a Logical Unit Number (LUN) in a Flex Unified storage pool.
- During publish, Trident maps the LUN to a per-node host group.
- During node staging, Trident:
  - Logs in to the iSCSI target

- Discovers the LUN
- Configures multipath

- If `volumeMode: Filesystem`, Trident formats the device if required and mounts it.
- If `volumeMode: Block`, Trident attaches the device and exposes it directly to the pod without formatting or mounting.

> SAN block volumes do not provide distributed locking or write coordination. When a block volume is accessed by multiple nodes (ReadWriteMany with `volumeMode: Block`), the application or filesystem must manage concurrency.

Prepare to configure a Google Cloud NetApp Volumes backend

Before you can configure your Google Cloud NetApp Volumes backend, you need to ensure the following requirements are met.

Prerequisites for NFS or SMB volumes

If you are using Google Cloud NetApp Volumes for the first time or in a new location, some initial configuration is required to set up Google Cloud NetApp Volumes and create an NFS or SMB volume. Refer to Before you begin.

Ensure that you have the following before configuring Google Cloud NetApp Volumes backend:

- A Google Cloud account configured with Google Cloud NetApp Volumes service. Refer to Google Cloud NetApp Volumes.
- Project number of your Google Cloud account. Refer to Identifying projects.
- A Google Cloud service account with the NetApp Volumes Admin (`roles/netapp.admin`) role. Refer to Identity and Access Management roles and permissions.
- API key file for your GCNV account. Refer to Create a service account key
- A storage pool. Refer to Storage pools overview .

For more information about how to set up access to Google Cloud NetApp Volumes, refer to Set up access to Google Cloud NetApp Volumes.

Google Cloud NetApp Volumes backend configuration options and examples

Learn about backend configuration options for Google Cloud NetApp Volumes and review configuration examples.

Backend configuration options

Each backend provisions volumes in a single Google Cloud region. To create volumes in other regions, you can define additional backends.

Parameter	Description	Default
`version`		Always 1

Parameter	Description	Default
`storageDriverName`	Name of the storage driver	The value of `storageDriverName` must be specified as "google-cloud-netapp-volumes".
`backendName`	(Optional) Custom name of the storage backend	Driver name + "_" + part of API key
`storagePools`	Optional parameter used to specify storage pools for volume creation.	
`projectNumber`	Google Cloud account project number. The value is found on the Google Cloud portal home page.	
`location`	The Google Cloud location where Trident creates GCNV volumes. When creating cross-region Kubernetes clusters, volumes created in a `location` can be used in workloads scheduled on nodes across multiple Google Cloud regions. Cross-region traffic incurs an additional cost.	
`apiKey`	API key for the Google Cloud service account with the `netapp.admin` role. It includes the JSON-formatted contents of a Google Cloud service account's private key file (copied verbatim into the backend configuration file). The `apiKey` must include key-value pairs for the following keys: `type`, `project_id`, `client_email`, `client_id`, `auth_uri`, `token_uri`, `auth_provider_x509_cert_url`, and `client_x509_cert_url`.	
`nfsMountOptions`	Fine-grained control of NFS mount options.	"nfsvers=3"
`limitVolumeSize`	Fail provisioning if the requested volume size is above this value.	"" (not enforced by default)
`serviceLevel`	The service level of a storage pool and its volumes. The values are `flex`, `standard`, `premium`, or `extreme`.	
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`network`	Google Cloud network used for GCNV volumes.	
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `{"api":false, "method":true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null

Parameter	Description	Default
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes.	`nfs`

`supportedTopologies`	Represents a list of regions and zones that are supported by this backend.
	
	For more information, refer to Use CSI Topology.
	For example:
	`supportedTopologies:`
	`- topology.kubernetes.io/region: asia-east1`
	`topology.kubernetes.io/zone: asia-east1-a`	

Volume provisioning options

You can control default volume provisioning in the `defaults` section of the configuration file.

Parameter	Description	Default
`exportRule`	The export rules for new volumes. Must be a comma-separated list of any combination of IPv4 addresses.	"0.0.0.0/0"
`snapshotDir`	Access to the `.snapshot` directory	`true`, `false` (default behavior might vary. Set explicitly) "false" for NFSv3
`snapshotReserve`	Percentage of volume reserved for snapshots	"" (accept default of 0)
`unixPermissions`	The unix permissions of new volumes (4 octal digits).	""

Example configurations

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

Minimal configuration

This is the absolute minimum backend configuration. With this configuration, Trident discovers all of your storage pools delegated to Google Cloud NetApp Volumes in the configured location, and places new volumes on one of those pools randomly. Because `nasType` is omitted, the `nfs` default applies and the backend will provision for NFS volumes.

This configuration is ideal when you are just getting started with Google Cloud NetApp Volumes and trying things out, but in practice you might need to provide additional scoping for the volumes you provision.

Replace `<id_value>` and `<key_value>` with your service account credentials.

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

Configuration for SMB volumes

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

Configuration with StoragePools filter

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

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

Virtual pool configuration

This backend configuration defines multiple virtual pools in a single file. Virtual pools are defined in the `storage` section. They are useful when you have multiple storage pools supporting different service levels and you want to create storage classes in Kubernetes that represent those. Virtual pool labels are used to differentiate the pools. For instance, in the example below `performance` label and `serviceLevel` type is used to differentiate virtual pools.

You can also set some default values to be applicable to all virtual pools, and overwrite the default values for individual virtual pools. In the following example, `snapshotReserve` and `exportRule` serve as defaults for all virtual pools.

For more information, refer to Virtual pools.

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

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

```
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

Cloud identity for GKE

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

Supported topologies configuration

Trident facilitates provisioning of volumes for workloads based on regions and availability zones. The `supportedTopologies` block in this backend configuration is used to provide a list of regions and zones per backend. The region and zone values specified here must match the region and zone values from the labels on each Kubernetes cluster node. These regions and zones represent the list of permissible values that can be provided in a storage class. For storage classes that contain a subset of the regions and zones provided in a backend, Trident creates volumes in the mentioned region and zone. For more information, refer to Use CSI Topology.

```
---
version: 1
storageDriverName: google-cloud-netapp-volumes
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: asia-east1
serviceLevel: flex
supportedTopologies:
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-a
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-b
```

What's next?

After you create the backend configuration file, run the following command:

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

To verify that the backend is successfully created, run the following command:

```
kubectl get tridentbackendconfig

NAME                BACKEND NAME        BACKEND UUID
PHASE   STATUS
backend-tbc-gcnv    backend-tbc-gcnv    b2fd1ff9-b234-477e-88fd-713913294f65
Bound   Success
```

If the backend creation fails, something is wrong with the backend configuration. You can describe the backend using the `kubectl get tridentbackendconfig <backend-name>` command or view the logs to determine the cause by running the following command:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can delete the backend and run the create command again.

Storage class definitions

The following is a basic `StorageClass` definition that refers to the backend above.

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

Example definitions using the `parameter.selector` field:

Using `parameter.selector` you can specify for each `StorageClass` the virtual pool that is used to host a volume. The volume will have the aspects defined in the chosen pool.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: extreme-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=extreme
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: premium-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=standard
  backendType: google-cloud-netapp-volumes
```

For more details on storage classes, refer to Create a storage class.

Example definitions for SMB volumes

Using `nasType`, `node-stage-secret-name`, and `node-stage-secret-namespace`, you can specify an SMB volume and provide the required Active Directory credentials. Any Active Directory user/password with any/no permissions can be used for the node stage secret.

Basic configuration on default namespace

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

Using different secrets per namespace

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

Using different secrets per volume

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

`nasType: smb` filters for pools which support SMB volumes. `nasType: nfs` or `nasType: null` filters for NFS pools.

PVC definition example

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

To verify if the PVC is bound, run the following command:

```
kubectl get pvc gcnv-nfs-pvc

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

Configure auto-tiering for Google Cloud NetApp Volumes

Auto-tiering is configured through Trident backend parameters and PersistentVolumeClaim annotations during volume provisioning. You can configure auto-tiering for Google Cloud NetApp Volumes using Trident.

Overview

Auto-tiering allows Trident to provision volumes that automatically move inactive data from a performance tier to a capacity tier.
This reduces storage cost while preserving performance for frequently accessed data.

Trident applies auto-tiering settings only at volume creation time.
Post-provisioning changes are not supported in Trident 26.02.

Concepts

Auto-tiering

Auto-tiering moves infrequently accessed data from a performance tier to a capacity tier based on access patterns.

Data movement occurs asynchronously and is not immediate.

Tiering policy

The tiering policy determines whether auto-tiering is enabled for a volume.

The following policies are supported:
* `auto`: Enables automatic tiering based on access patterns
* `none`: Disables auto-tiering

Cooling days

Cooling days specify the minimum number of days a block of data must remain inactive before it becomes eligible for tiering.
Cooling days apply only when the tiering policy is set to `auto`.

Configuration model

Configuration scopes

Auto-tiering can be configured at multiple scopes:

- **Storage pool scope**
  Applies to all volumes provisioned from the pool.
- **Volume scope**
  Applies to a single volume through PersistentVolumeClaim annotations.

Trident determines the effective configuration based on where each setting is defined.

Configuration precedence

When the same setting is defined at multiple scopes, Trident applies the following precedence order:

1. PersistentVolumeClaim annotations
2. Trident backend configuration
3. Storage pool defaults

Settings defined at a higher precedence override lower-level values.

Supported functionality in Trident 26.02

Trident 26.02 supports the following auto-tiering capabilities for Google Cloud NetApp Volumes:

- Enabling or disabling auto-tiering during volume provisioning
- Defining a tiering policy in the Trident backend configuration
- Overriding the tiering policy and cooling days per volume using PVC annotations
- Configuring cooling days for volumes with auto-tiering enabled

Unsupported functionality in Trident 26.02

The following operations are not supported:

- Modifying auto-tiering settings after volume creation
- Changing tiering policies on existing volumes using Kubernetes updates
- Applying auto-tiering settings outside of Trident-managed provisioning workflows

Backend configuration parameters

The following parameters control auto-tiering behavior when defined in the Trident backend configuration:

Parameter	Required	Description
tieringPolicy	No	Tiering policy for volumes (`auto` or `none`)
tieringMinimumCoolingDays	No	Number of inactive days before data is tiered (range: 2–183, default: 31)

Volume-level overrides using PersistentVolumeClaim annotations

Supported annotations

PersistentVolumeClaim annotations allow per-volume overrides of auto-tiering settings.

Annotation	Description
`trident.netapp.io/tieringPolicy`	Overrides the tiering policy for the volume
`trident.netapp.io/tieringMinimumCoolingDays`	Overrides the cooling days value for the volume

Example: PersistentVolumeClaim with auto-tiering overrides

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-tiering-pvc
  annotations:
    trident.netapp.io/tieringPolicy: auto
    trident.netapp.io/tieringMinimumCoolingDays: "45"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: google-cloud-netapp-volumes-auto-tiering
  resources:
    requests:
      storage: 500Gi
```

Behavior and limitations

Provisioning behavior

- Auto-tiering settings are evaluated and applied only at volume creation time.
- Trident does not reconcile tiering configuration after provisioning.
- Cooling days are ignored when the tiering policy is set to `none`.

Platform limitations

- Auto-tiering is supported only for NAS volumes (NFS and SMB).
- Block volumes (iSCSI) do not support auto-tiering.
- The Google Cloud NetApp Volumes storage pool must have auto-tiering enabled in Google Cloud.

Supported values

- Valid range for `tieringMinimumCoolingDays`: 2 to 183
- Default value: 31

Configure a NetApp HCI or SolidFire backend

Learn how to create and use an Element backend with your Trident installation.

Element driver details

Trident provides the `solidfire-san` storage driver to communicate with the cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

The `solidfire-san` storage driver supports *file* and *block* volume modes. For the `Filesystem` volumeMode, Trident creates a volume and creates a filesystem. The filesystem type is specified by the StorageClass.

Driver	Protocol	VolumeMode	Access modes supported	File systems supported
`solidfire-san`	iSCSI	Block	RWO, ROX, RWX, RWOP	No Filesystem. Raw block device.
`solidfire-san`	iSCSI	Filesystem	RWO, RWOP	`xfs`, `ext3`, `ext4`

Before you begin

You'll need the following before creating an Element backend.

- A supported storage system that runs Element software.
- Credentials to a NetApp HCI/SolidFire cluster admin or tenant user that can manage volumes.
- All of your Kubernetes worker nodes should have the appropriate iSCSI tools installed. Refer to worker node preparation information.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	Always "solidfire-san"
`backendName`	Custom name or the storage backend	"solidfire_" + storage (iSCSI) IP address
`Endpoint`	MVIP for the SolidFire cluster with tenant credentials	
`SVIP`	Storage (iSCSI) IP address and port	
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes.	""
`TenantName`	Tenant name to use (created if not found)	
`InitiatorIFace`	Restrict iSCSI traffic to a specific host interface	"default"
`UseCHAP`	Use CHAP to authenticate iSCSI. Trident uses CHAP.	true
`AccessGroups`	List of Access Group IDs to use	Finds the ID of an access group named "trident"
`Types`	QoS specifications	
`limitVolumeSize`	Fail provisioning if requested volume size is above this value	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true}	null

Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.

Example 1: Backend configuration for `solidfire-san` driver with three volume types

This example shows a backend file using CHAP authentication and modeling three volume types with specific QoS guarantees. Most likely you would then define storage classes to consume each of these using the `IOPS` storage class parameter.

```
---
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
```

Example 2: Backend and storage class configuration for `solidfire-san` driver with virtual pools

This example shows the backend definition file configured with virtual pools along with StorageClasses that refer back to them.

Trident copies labels present on a storage pool to the backend storage LUN at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the `type` at Silver. The virtual pools are defined in the `storage` section. In this example, some of the storage pools set their own type, and some pools override the default values set above.

```
---
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
```

The following StorageClass definitions refer to the above virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pool(s) can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

The first StorageClass (`solidfire-gold-four`) will map to the first virtual pool. This is the only pool offering gold performance with a `Volume Type QoS` of Gold. The last StorageClass (`solidfire-silver`) calls out any storage pool which offers a silver performance. Trident will decide which virtual pool is selected and ensures the storage requirement is met.

```
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
```

Find more information

- Volume access groups

ONTAP SAN drivers

ONTAP SAN driver overview

Learn about configuring an ONTAP backend with ONTAP and Cloud Volumes ONTAP SAN drivers.

ONTAP SAN driver details

Trident provides the following SAN storage drivers to communicate with the ONTAP cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-san`	iSCSI SCSI over FC	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device
`ontap-san`	iSCSI SCSI over FC	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`
`ontap-san`	NVMe/TCP Refer to Additional considerations for NVMe/TCP.	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-san`	NVMe/TCP Refer to Additional considerations for NVMe/TCP.	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`
`ontap-san-economy`	iSCSI	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device
`ontap-san-economy`	iSCSI	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`

- Use `ontap-san-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits.
- Use `ontap-nas-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits and the `ontap-san-economy` driver cannot be used.
- Do not use use `ontap-nas-economy` if you anticipate the need for data protection, disaster recovery, or mobility.
- NetApp does not recommend using Flexvol autogrow in all ONTAP drivers, except ontap-san. As a workaround, Trident supports the use of snapshot reserve and scales Flexvol volumes accordingly.

User permissions

Trident expects to be run as either an ONTAP or SVM administrator, typically using the `admin` cluster user or a `vsadmin` SVM user, or a user with a different name that has the same role. For Amazon FSx for NetApp ONTAP deployments, Trident expects to be run as either an ONTAP or SVM administrator, using the cluster `fsxadmin` user or a `vsadmin` SVM user, or a user with a different name that has the same role. The `fsxadmin` user is a limited replacement for the cluster admin user.

If you use the `limitAggregateUsage` parameter, cluster admin permissions are required. When using Amazon FSx for NetApp ONTAP with Trident, the `limitAggregateUsage` parameter will not work with the `vsadmin` and `fsxadmin` user accounts. The configuration operation will fail if you specify this parameter.

While it is possible to create a more restrictive role within ONTAP that a Trident driver can use, we don't recommend it. Most new releases of Trident will call additional APIs that would have to be accounted for, making upgrades difficult and error-prone.

Additional considerations for NVMe/TCP

Trident supports the non-volatile memory express (NVMe) protocol using the `ontap-san` driver including:

- IPv6
- Snapshots and clones of NVMe volumes
- Resizing an NVMe volume
- Importing an NVMe volume that was created outside of Trident so that its lifecycle can be managed by Trident
- NVMe-native multipathing
- Graceful or ungraceful shutdown of the K8s nodes (24.06)

Trident does not support:

- DH-HMAC-CHAP that is natively supported by NVMe
- Device mapper (DM) multipathing
- LUKS encryption

NVMe is supported only with ONTAP REST APIs and not supported with ONTAPI (ZAPI).

Prepare to configure backend with ONTAP SAN drivers

Understand the requirements and authentication options for configuring an ONTAP backend with ONTAP SAN drivers.

Requirements

For all ONTAP backends, Trident requires at least one aggregate be assigned to the SVM.

ASA r2 systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. In ASA r2 systems, storage availability zones are used instead of aggregates. Refer to this Knowledge Base article on how to assign aggregates to SVMs in ASA r2 systems.

Remember that you can also run more than one driver, and create storage classes that point to one or the other. For example, you could configure a `san-dev` class that uses the `ontap-san` driver and a `san-default` class that uses the `ontap-san-economy` one.

All your Kubernetes worker nodes must have the appropriate iSCSI tools installed. Refer to Prepare the worker node for details.

Authenticate the ONTAP backend

Trident offers two modes of authenticating an ONTAP backend.

- Credential-based: The username and password to an ONTAP user with the required permissions. It is recommended to use a pre-defined security login role, such as `admin` or `vsadmin` to ensure maximum compatibility with ONTAP versions.
- Certificate-based: Trident can also communicate with an ONTAP cluster using a certificate installed on the backend. Here, the backend definition must contain Base64-encoded values of the client certificate, key,

and the trusted CA certificate if used (recommended).

You can update existing backends to move between credential-based and certificate-based methods. However, only one authentication method is supported at a time. To switch to a different authentication method, you must remove the existing method from the backend configuration.

> If you attempt to provide **both credentials and certificates**, backend creation will fail with an error that more than one authentication method was provided in the configuration file.

Enable credential-based authentication

Trident requires the credentials to an SVM-scoped/cluster-scoped admin to communicate with the ONTAP backend. It is recommended to make use of standard, pre-defined roles such as `admin` or `vsadmin`. This ensures forward compatibility with future ONTAP releases that might expose feature APIs to be used by future Trident releases. A custom security login role can be created and used with Trident, but is not recommended.

A sample backend definition will look like this:

YAML

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

JSON

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

Keep in mind that the backend definition is the only place the credentials are stored in plain text. After the backend is created, usernames/passwords are encoded with Base64 and stored as Kubernetes secrets. The creation or update of a backend is the only step that requires knowledge of the credentials. As such, it is an admin-only operation, to be performed by the Kubernetes/storage administrator.

Enable certificate-based authentication

New and existing backends can use a certificate and communicate with the ONTAP backend. Three parameters are required in the backend definition.

- clientCertificate: Base64-encoded value of client certificate.
- clientPrivateKey: Base64-encoded value of associated private key.
- trustedCACertificate: Base64-encoded value of trusted CA certificate. If using a trusted CA, this parameter must be provided. This can be ignored if no trusted CA is used.

A typical workflow involves the following steps.

Steps

1. Generate a client certificate and key. When generating, set Common Name (CN) to the ONTAP user to authenticate as.

```
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. Add trusted CA certificate to the ONTAP cluster. This might be already handled by the storage administrator. Ignore if no trusted CA is used.

```
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. Install the client certificate and key (from step 1) on the ONTAP cluster.

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

> After running this command, ONTAP prompts for certificate input. Paste the contents of the `k8senv.pem` file generated in step 1, then enter `END` to complete the installation.

4. Confirm the ONTAP security login role supports `cert` authentication method.

```
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. Test authentication using certificate generated. Replace <ONTAP Management LIF> and <vserver name> with Management LIF IP and SVM name.

```
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. Encode certificate, key and trusted CA certificate with Base64.

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

7. Create backend using the values obtained from the previous step.

```
cat cert-backend.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"trustedCACertificate": "QNFinfO...SiqOyN",
"storagePrefix": "myPrefix_"
}

tridentctl create backend -f cert-backend.json -n trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       0 |
+------------+----------------+--------------------------------------
+--------+---------+
```

Update authentication methods or rotate credentials

You can update an existing backend to use a different authentication method or to rotate their credentials. This works both ways: backends that make use of username/password can be updated to use certificates; backends that utilize certificates can be updated to username/password based. To do this, you must remove the existing authentication method and add the new authentication method. Then use the updated backend.json file containing the required parameters to execute `tridentctl backend update`.

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

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

When rotating passwords, the storage administrator must first update the password for the user on ONTAP. This is followed by a backend update. When rotating certificates, multiple certificates can be added to the user. The backend is then updated to use the new certificate, following which the old certificate can be deleted from the ONTAP cluster.

Updating a backend does not disrupt access to volumes that have already been created, nor impact volume connections made after. A successful backend update indicates that Trident can communicate with the ONTAP backend and handle future volume operations.

Create custom ONTAP role for Trident

You can create an ONTAP cluster role with minimum privileges so that you do not have to use the ONTAP admin role to perform operations in Trident. When you include the username in a Trident backend configuration, Trident uses the ONTAP cluster role you created to perform the operations.

Refer to Trident custom-role generator for more information about creating Trident custom roles.

Using ONTAP CLI

1. Create a new role using the following command:

```
security login role create <role_name\> -cmddirname "command" -access all
–vserver <svm_name\>
```

2. Create a usename for the Trident user:

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> –vserver
<svm_name\> -comment "user_description"
```

3. Map the role to the user:

```
security login modify username <user_name\> –vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

Using System Manager

Perform the following steps in ONTAP System Manager:

1. **Create a custom role**:
   a. To create a custom role at the cluster-level, select **Cluster > Settings**.

      (Or) To create a custom role at the SVM level, select **Storage > Storage VMs > `required SVM`> Settings > Users and Roles**.

   b. Select the arrow icon (→) next to **Users and Roles**.
   c. Select **+Add** under **Roles**.
   d. Define the rules for the role and click **Save**.
2. **Map the role to the Trident user**:
   + Perform the following steps on the **Users and Roles** page:
   a. Select Add icon **+** under **Users**.
   b. Select the required username, and select a role in the drop-down menu for **Role**.
   c. Click **Save**.

Refer to the following pages for more information:

- Custom roles for administration of ONTAP or Define custom roles
- Work with roles and users

Authenticate connections with bidirectional CHAP

Trident can authenticate iSCSI sessions with bidirectional CHAP for the `ontap-san` and `ontap-san-economy` drivers. This requires enabling the `useCHAP` option in your backend definition. When set to `true`, Trident configures the SVM's default initiator security to bidirectional CHAP and set the username and secrets

from the backend file. NetApp recommends using bidirectional CHAP to authenticate connections. See the following sample configuration:

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
```

The `useCHAP` parameter is a Boolean option that can be configured only once. It is set to false by default. After you set it to true, you cannot set it to false.

In addition to `useCHAP=true`, the `chapInitiatorSecret`, `chapTargetInitiatorSecret`, `chapTargetUsername`, and `chapUsername` fields must be included in the backend definition. The secrets can be changed after a backend is created by running `tridentctl update`.

How it works

By setting `useCHAP` to true, the storage administrator instructs Trident to configure CHAP on the storage backend. This includes the following:

- Setting up CHAP on the SVM:
  - If the SVM's default initiator security type is none (set by default) **and** there are no pre-existing LUNs already present in the volume, Trident will set the default security type to `CHAP` and proceed to configuring the CHAP initiator and target username and secrets.
  - If the SVM contains LUNs, Trident will not enable CHAP on the SVM. This ensures that access to LUNs that are already present on the SVM isn't restricted.
- Configuring the CHAP initiator and target username and secrets; these options must be specified in the backend configuration (as shown above).

After the backend is created, Trident creates a corresponding `tridentbackend` CRD and stores the CHAP secrets and usernames as Kubernetes secrets. All PVs that are created by Trident on this backend will be mounted and attached over CHAP.

Rotate credentials and update backends

You can update the CHAP credentials by updating the CHAP parameters in the `backend.json` file. This will require updating the CHAP secrets and using the `tridentctl update` command to reflect these changes.

When updating the CHAP secrets for a backend, you must use `tridentctl` to update the backend. Do not update the credentials on the storage cluster using the ONTAP CLI or ONTAP System Manager as Trident will not be able to pick up these changes.

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

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

Existing connections will remain unaffected; they will continue to remain active if the credentials are updated by Trident on the SVM. New connections use the updated credentials and existing connections continue to remain active. Disconnecting and reconnecting old PVs will result in them using the updated credentials.

ONTAP SAN configuration options and examples

Learn how to create and use ONTAP SAN drivers with your Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses. ASA r2 systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. These variations impact the usage of certain parameters as notated. Learn more about the differences between ASA r2 systems and other ONTAP systems. In the Trident backend configuration, you need not specify that your system is ASA r2. When you select `ontap-san` as the `storageDriverName`,

Trident detects automatically the ASA r2 or other ONTAP systems. Some backend configuration parameters are not applicable to ASA r2 systems as noted in the table below.

> Only the `ontap-san` driver (with iSCSI, NVMe/TCP, and FC protocols) is supported for ASA r2 systems.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	`ontap-san` or `ontap-san-economy`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF. A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. For seamless MetroCluster switchover, see the MetroCluster example. > If you are using "vsadmin" credentials, `managementLIF` must be that of the SVM; if using "admin" credentials, `managementLIF` must be that of the cluster.	"10.0.0.1", "[2001:1234:abcd::fefe]"

Parameter	Description	Default
`dataLIF`	IP address of protocol LIF. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. **Do not specify for iSCSI.** Trident uses ONTAP Selective LUN Map to discover the iSCSI LIFs needed to establish a multi path session. A warning is generated if `dataLIF` is explicitly defined. **Omit for Metrocluster.** See the MetroCluster example.	Derived by the SVM
`svm`	Storage virtual machine to use **Omit for Metrocluster.** See the MetroCluster example.	Derived if an SVM `managementLIF` is specified
`useCHAP`	Use CHAP to authenticate iSCSI for ONTAP SAN drivers [Boolean]. Set to `true` for Trident to configure and use bidirectional CHAP as the default authentication for the SVM given in the backend. Refer to Prepare to configure backend with ONTAP SAN drivers for details. **Not supported for FCP or NVMe/TCP.**	`false`
`chapInitiatorSecret`	CHAP initiator secret. Required if `useCHAP=true`	""
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`chapTargetInitiatorSecret`	CHAP target initiator secret. Required if `useCHAP=true`	""
`chapUsername`	Inbound username. Required if `useCHAP=true`	""
`chapTargetUsername`	Target username. Required if `useCHAP=true`	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based authentication.	""

Parameter	Description	Default
`username`	Username needed to communicate with the ONTAP cluster. Used for credential-based authentication. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	""
`password`	Password needed to communicate with the ONTAP cluster. Used for credential-based authentication. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	""
`svm`	Storage virtual machine to use	Derived if an SVM `managementLIF` is specified
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be modified later. To update this parameter, you will need to create a new backend.	`trident`
`aggregate`	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the `ontap-nas-flexgroup` driver, this option is ignored. If not assigned, any of the available aggregates can be used to provision a FlexGroup volume. When the aggregate is updated in SVM, it is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate in Trident to provision volumes, if the aggregate is renamed or moved out of the SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate to one that is present on the SVM or remove it altogether to bring the backend back online. **Do not specify for ASA r2 systems**.	""
`limitAggregateUsage`	Fail provisioning if usage is above this percentage. If you are using an Amazon FSx for NetApp ONTAP backend, do not specify `limitAggregateUsage`. The provided `fsxadmin` and `vsadmin` do not contain the permissions required to retrieve aggregate usage and limit it using Trident. **Do not specify for ASA r2 systems**.	"" (not enforced by default)

Parameter	Description	Default
`limitVolumeSize`	Fail provisioning if requested volume size is above this value. Also restricts the maximum size of the volumes it manages for LUNs.	"" (not enforced by default)
`lunsPerFlexvol`	Maximum LUNs per Flexvol, must be in range [50, 200]	`100`
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use unless you are troubleshooting and require a detailed log dump.	`null`
`useREST`	Boolean parameter to use ONTAP REST APIs. `useREST` When set to `true`, Trident uses ONTAP REST APIs to communicate with the backend; when set to `false`, Trident uses ONTAPI (ZAPI) calls to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the `ontapi` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles. Beginning with the Trident 24.06 release and ONTAP 9.15.1 or later, `useREST` is set to `true` by default; change `useREST` to `false` to use ONTAPI (ZAPI) calls. `useREST` is fully qualified for NVMe/TCP. NVMe is supported only with ONTAP REST APIs and not supported with ONTAPI (ZAPI). **If specified, always set to `true` for ASA r2 systems**.	`true` for ONTAP 9.15.1 or later, otherwise `false`.
`sanType`	Use to select `iscsi` for iSCSI, `nvme` for NVMe/TCP or `fcp` for SCSI over Fibre Channel (FC).	`iscsi` if blank

Parameter	Description	Default
`formatOptions`	Use `formatOptions` to specify command line arguments for the `mkfs` command, which will be applied whenever a volume is formatted. This allows you to format the volume according to your preferences. Make sure to specify the formatOptions similar to that of the mkfs command options, excluding the device path. Example: "-E nodiscard" **Supported for `ontap-san` and `ontap-san-economy` drivers with iSCSI protocol. Additionally, supported for ASA r2 systems when using iSCSI and NVMe/TCP protocols.**	
`limitVolumePoolSize`	Maximum requestable FlexVol size when using LUNs in ontap-san-economy backend.	"" (not enforced by default)
`denyNewVolumePools`	Restricts `ontap-san-economy` backends from creating new FlexVol volumes to contain their LUNs. Only preexisting Flexvols are used for provisioning new PVs.	

Recommendations for using formatOptions

Trident recommends the following options to expedite the formatting process:

- **-E nodiscard (ext3, ext4):** Do not attempt to discard blocks at mkfs time (discarding blocks initially is useful on solid state devices and sparse / thin-provisioned storage). This replaces the deprecated option "-K" and it is applicable to ext3, ext4 file systems.
- **-K (xfs):** Do not attempt to discard blocks at mkfs time. This option is applicable to xfs file system.

Authenticate Trident to a backend SVM using Active Directory credentials

You can configure Trident to authenticate to a backend SVM using Active Directory (AD) credentials. Before an AD account can access the SVM, you must configure AD domain controller access to the cluster or SVM. For cluster administration with an AD account, you must create domain tunnel. Refer to Configure Active Directory domain controller access in ONTAP for details.

steps

1. Configure Domain Name System (DNS) settings for a backend SVM:

   ```
   vserver services dns create -vserver <svm_name> -dns-servers
   <dns_server_ip1>,<dns_server_ip2>
   ```

2. Run the following command to create a computer account for the SVM in Active Directory:

   ```
   vserver active-directory create -vserver DataSVM -account-name ADSERVER1
   -domain demo.netapp.com
   ```

3. Use this command to create an AD user or group to manage the cluster or SVM

```
security login create -vserver <svm_name> -user-or-group-name
<ad_user_or_group> -application <application> -authentication-method domain
-role vsadmin
```

4. In the Trident backend configuration file, set the `username` and `password` parameters to the AD user or group name and password, respectively.

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the `defaults` section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
`spaceAllocation`	Space-allocation for LUNs	"true" **If specified, set to `true` for ASA r2 systems**.
`spaceReserve`	Space reservation mode; "none" (thin) or "volume" (thick). **Set to `none` for ASA r2** systems.	"none"
`snapshotPolicy`	Snapshot policy to use. **Set to `none` for ASA r2 systems**.	"none"
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend. Using QoS policy groups with Trident requires ONTAP 9.8 or later. You should use a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group enforces the ceiling for the total throughput of all workloads.	""
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""
`snapshotReserve`	Percentage of volume reserved for snapshots. **Do not specify for ASA r2 systems**.	"0" if `snapshotPolicy` is "none", otherwise ""
`splitOnClone`	Split a clone from its parent upon creation	"false"
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	"false" **If specified, set to `true` for ASA r2 systems**.

Parameter	Description	Default
`luksEncryption`	Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS).	"" **Set to `false` for ASA r2 systems.**
`tieringPolicy`	Tiering policy to use "none" **Do not specify for ASA r2 systems** .	
`nameTemplate`	Template to create custom volume names.	""

Volume provisioning examples

Here's an example with defaults defined:

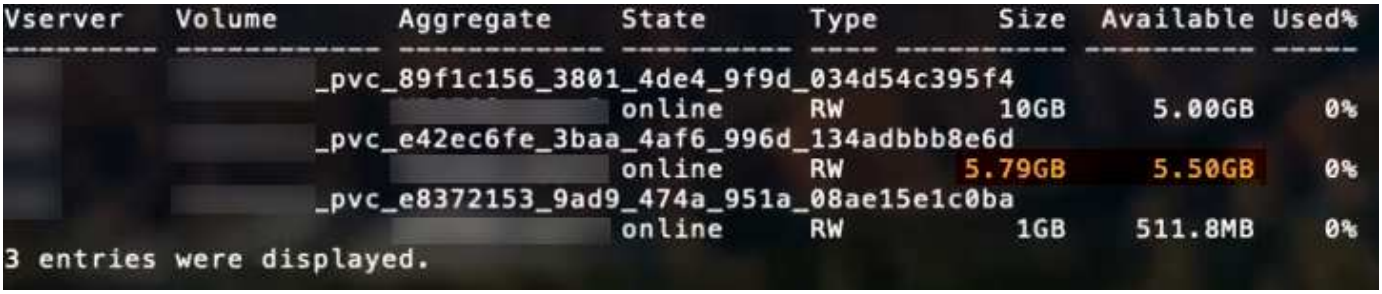
```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: <password>
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: standard
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

For all volumes created using the `ontap-san` driver, Trident adds an extra 10 percent capacity to the FlexVol to accommodate the LUN metadata. The LUN will be provisioned with the exact size that the user requests in the PVC. Trident adds 10 percent to the FlexVol (shows as Available size in ONTAP). Users will now get the amount of usable capacity they requested. This change also prevents LUNs from becoming read-only unless the available space is fully utilized. This does not apply to ontap-san-economy.

For backends that define `snapshotReserve`, Trident calculates the size of volumes as follows:

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

The 1.1 is the extra 10 percent Trident adds to the FlexVol to accommodate the LUN metadata. For `snapshotReserve` = 5%, and PVC request = 5 GiB, the total volume size is 5.79 GiB and the available size is 5.5 GiB. The `volume show` command should show results similar to this example:



Currently, resizing is the only way to use the new calculation for an existing volume.

Minimal configuration examples

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

> If you are using Amazon FSx on NetApp ONTAP with Trident, NetApp recommends that you specify DNS names for LIFs instead of IP addresses.

ONTAP SAN example

This is a basic configuration using the `ontap-san` driver.

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
username: vsadmin
password: <password>
```

MetroCluster example

You can configure the backend to avoid having to manually update the backend definition after switchover and switchback during SVM replication and recovery.

For seamless switchover and switchback, specify the SVM using `managementLIF` and omit the `svm` parameters. For example:

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

ONTAP SAN economy example

```
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
username: vsadmin
password: <password>
```

Certificate-based authentication example

In this basic configuration example `clientCertificate`, `clientPrivateKey`, and `trustedCACertificate` (optional, if using trusted CA) are populated in `backend.json` and take the base64-encoded values of the client certificate, private key, and trusted CA certificate, respectively.

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

Bidirectional CHAP examples

These examples create a backend with `useCHAP` set to `true`.

ONTAP SAN CHAP example

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

ONTAP SAN economy CHAP example

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

NVMe/TCP example

You must have an SVM configured with NVMe on your ONTAP backend. This is a basic backend configuration for NVMe/TCP.

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

SCSI over FC (FCP) example

You must have an SVM configured with FC on your ONTAP backend. This is a basic backend configuration for FC.

```
---
version: 1
backendName: fcp-backend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_fc
username: vsadmin
password: password
sanType: fcp
useREST: true
```

Backend configuration example with nameTemplate

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap-san-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

formatOptions example for ontap-san-economy driver

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: ""
svm: svm1
username: ""
password: "!"
storagePrefix: whelk_
debugTraceFlags:
  method: true
  api: true
defaults:
  formatOptions: -E nodiscard
```

Examples of backends with virtual pools

In these sample backend definition files, specific defaults are set for all storage pools, such as `spaceReserve` at none, `spaceAllocation` at false, and `encryption` at false. The virtual pools are defined in the storage section.

Trident sets provisioning labels in the "Comments" field. Comments are set on the FlexVol volume Trident copies all labels present on a virtual pool to the storage volume at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In these examples, some of the storage pools set their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools override the default values.

ONTAP SAN example

```yaml
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "40000"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
      adaptiveQosPolicy: adaptive-extreme
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
      qosPolicy: premium
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

ONTAP SAN economy example

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
labels:
  store: san_economy_store
region: us_east_1
storage:
  - labels:
      app: oracledb
      cost: "30"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
  - labels:
      app: postgresdb
      cost: "20"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
  - labels:
      app: mysqldb
      cost: "10"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1c
```

```
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

NVMe/TCP example

```
---
version: 1
storageDriverName: ontap-san
sanType: nvme
managementLIF: 10.0.0.1
svm: nvme_svm
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceAllocation: "false"
  encryption: "true"
storage:
  - labels:
      app: testApp
      cost: "20"
    defaults:
      spaceAllocation: "false"
      encryption: "false"
```

Map backends to StorageClasses

The following StorageClass definitions refer to the Examples of backends with virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

- The `protection-gold` StorageClass will map to the first virtual pool in the `ontap-san` backend. This is the only pool offering gold-level protection.

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

- The `protection-not-gold` StorageClass will map to the second and third virtual pool in `ontap-san` backend. These are the only pools offering a protection level other than gold.

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

- The `app-mysqldb` StorageClass will map to the third virtual pool in `ontap-san-economy` backend. This is the only pool offering storage pool configuration for the mysqldb type app.

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

- The `protection-silver-creditpoints-20k` StorageClass will map to the second virtual pool in `ontap-san` backend. This is the only pool offering silver-level protection and 20000 creditpoints.

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

- The `creditpoints-5k` StorageClass will map to the third virtual pool in `ontap-san` backend and the fourth virtual pool in the `ontap-san-economy` backend. These are the only pool offerings with 5000 creditpoints.

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

- The `my-test-app-sc` StorageClass will map to the `testAPP` virtual pool in the `ontap-san` driver with `sanType: nvme`. This is the only pool offering `testApp`.

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

Trident will decide which virtual pool is selected and ensures the storage requirement is met.

ONTAP NAS drivers

ONTAP NAS driver overview

Learn about configuring an ONTAP backend with ONTAP and Cloud Volumes ONTAP NAS drivers.

ONTAP NAS driver details

Trident provides the following NAS storage drivers to communicate with the ONTAP cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-nas`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", `nfs`, `smb`
`ontap-nas-economy`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", `nfs`, `smb`

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-nas-flexgroup`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", `nfs`, `smb`

- Use `ontap-san-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits.
- Use `ontap-nas-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits and the `ontap-san-economy` driver cannot be used.
- Do not use use `ontap-nas-economy` if you anticipate the need for data protection, disaster recovery, or mobility.
- NetApp does not recommend using Flexvol autogrow in all ONTAP drivers, except ontap-san. As a workaround, Trident supports the use of snapshot reserve and scales Flexvol volumes accordingly.

User permissions

Trident expects to be run as either an ONTAP or SVM administrator, typically using the `admin` cluster user or a `vsadmin` SVM user, or a user with a different name that has the same role.

For Amazon FSx for NetApp ONTAP deployments, Trident expects to be run as either an ONTAP or SVM administrator, using the cluster `fsxadmin` user or a `vsadmin` SVM user, or a user with a different name that has the same role. The `fsxadmin` user is a limited replacement for the cluster admin user.

If you use the `limitAggregateUsage` parameter, cluster admin permissions are required. When using Amazon FSx for NetApp ONTAP with Trident, the `limitAggregateUsage` parameter will not work with the `vsadmin` and `fsxadmin` user accounts. The configuration operation will fail if you specify this parameter.

While it is possible to create a more restrictive role within ONTAP that a Trident driver can use, we don't recommend it. Most new releases of Trident will call additional APIs that would have to be accounted for, making upgrades difficult and error-prone.

Prepare to configure a backend with ONTAP NAS drivers

Understand the requirements, authentication options, and export policies for configuring an ONTAP backend with ONTAP NAS drivers. Beginning with the 25.10 release, NetApp Trident supports NetApp AFX storage system. NetApp AFX storage systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. In the Trident backend configuration, you need not specify that your system is AFX. When you select `ontap-nas` as the `storageDriverName`, Trident detects automatically the AFX systems.

Only the `ontap-nas` driver (with NFS protocol) is supported for AFX systems; SMB protocol is not supported.

Requirements

- For all ONTAP backends, Trident requires at least one aggregate be assigned to the SVM.
- You can run more than one driver, and create storage classes that point to one or the other. For example, you could configure a Gold class that uses the `ontap-nas` driver and a Bronze class that uses the `ontap-nas-economy` one.
- All your Kubernetes worker nodes must have the appropriate NFS tools installed. Refer to here for more details.
- Trident supports SMB volumes mounted to pods running on Windows nodes only. Refer to Prepare to provision SMB volumes for details.

Authenticate the ONTAP backend

Trident offers two modes of authenticating an ONTAP backend.

- Credential-based: This mode requires sufficient permissions to the ONTAP backend. It is recommended to use an account associated with a pre-defined security login role, such as `admin` or `vsadmin` to ensure maximum compatibility with ONTAP versions.
- Certificate-based: This mode requires a certificate installed on the backend for Trident to communicate with an ONTAP cluster. Here, the backend definition must contain Base64-encoded values of the client certificate, key, and the trusted CA certificate if used (recommended).

You can update existing backends to move between credential-based and certificate-based methods. However, only one authentication method is supported at a time. To switch to a different authentication method, you must remove the existing method from the backend configuration.

If you attempt to provide **both credentials and certificates**, backend creation will fail with an error that more than one authentication method was provided in the configuration file.

Enable credential-based authentication

Trident requires the credentials to an SVM-scoped/cluster-scoped admin to communicate with the ONTAP backend. It is recommended to make use of standard, pre-defined roles such as `admin` or `vsadmin`. This ensures forward compatibility with future ONTAP releases that might expose feature APIs to be used by future Trident releases. A custom security login role can be created and used with Trident, but is not recommended.

A sample backend definition will look like this:

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
credentials:
  name: secret-backend-creds
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "credentials": {
        "name": "secret-backend-creds"
    }
}
```

Keep in mind that the backend definition is the only place the credentials are stored in plain text. After the backend is created, usernames/passwords are encoded with Base64 and stored as Kubernetes secrets. The creation/updation of a backend is the only step that requires knowledge of the credentials. As such, it is an admin-only operation, to be performed by the Kubernetes/storage administrator.

Enable certificate-based Authentication

New and existing backends can use a certificate and communicate with the ONTAP backend. Three parameters are required in the backend definition.

- clientCertificate: Base64-encoded value of client certificate.
- clientPrivateKey: Base64-encoded value of associated private key.
- trustedCACertificate: Base64-encoded value of trusted CA certificate. If using a trusted CA, this parameter must be provided. This can be ignored if no trusted CA is used.

A typical workflow involves the following steps.

Steps

1. Generate a client certificate and key. When generating, set Common Name (CN) to the ONTAP user to authenticate as.

```
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. Add trusted CA certificate to the ONTAP cluster. This might be already handled by the storage administrator. Ignore if no trusted CA is used.

```
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. Install the client certificate and key (from step 1) on the ONTAP cluster.

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

4. Confirm the ONTAP security login role supports `cert` authentication method.

```
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. Test authentication using certificate generated. Replace <ONTAP Management LIF> and <vserver name> with Management LIF IP and SVM name. You must ensure the LIF has its service policy set to `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. Encode certificate, key and trusted CA certificate with Base64.

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

7. Create backend using the values obtained from the previous step.

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

Update authentication methods or rotate credentials

You can update an existing backend to use a different authentication method or to rotate their credentials. This works both ways: backends that make use of username/password can be updated to use certificates; backends that utilize certificates can be updated to username/password based. To do this, you must remove the existing authentication method and add the new authentication method. Then use the updated backend.json file containing the required parameters to execute `tridentctl update backend`.

```
cat cert-backend-updated.json
```

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

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

When rotating passwords, the storage administrator must first update the password for the user on ONTAP. This is followed by a backend update. When rotating certificates, multiple certificates can be added to the user. The backend is then updated to use the new certificate, following which the old certificate can be deleted from the ONTAP cluster.

Updating a backend does not disrupt access to volumes that have already been created, nor impact volume connections made after. A successful backend update indicates that Trident can communicate with the ONTAP backend and handle future volume operations.

Create custom ONTAP role for Trident

You can create an ONTAP cluster role with minimum privileges so that you do not have to use the ONTAP admin role to perform operations in Trident. When you include the username in a Trident backend configuration, Trident uses the ONTAP cluster role you created to perform the operations.

Refer to Trident custom-role generator for more information about creating Trident custom roles.

Using ONTAP CLI

1. Create a new role using the following command:

```
security login role create <role_name\> -cmddirname "command" -access all
–vserver <svm_name\>
```

2. Create a usename for the Trident user:

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> -vserver
<svm_name\> -comment "user_description"
```

3. Map the role to the user:

```
security login modify username <user_name\> –vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

Using System Manager

Perform the following steps in ONTAP System Manager:

1. **Create a custom role**:
   a. To create a custom role at the cluster-level, select **Cluster > Settings**.

      (Or) To create a custom role at the SVM level, select **Storage > Storage VMs > `required SVM`> Settings > Users and Roles**.

   b. Select the arrow icon (→) next to **Users and Roles**.
   c. Select **+Add** under **Roles**.
   d. Define the rules for the role and click **Save**.
2. **Map the role to the Trident user**:
   + Perform the following steps on the **Users and Roles** page:
   a. Select Add icon **+** under **Users**.
   b. Select the required username, and select a role in the drop-down menu for **Role**.
   c. Click **Save**.

Refer to the following pages for more information:

- Custom roles for administration of ONTAP or Define custom roles
- Work with roles and users

Manage NFS export policies

Trident uses NFS export policies to control access to the volumes that it provisions.

Trident provides two options when working with export policies:

- Trident can dynamically manage the export policy itself; in this mode of operation, the storage administrator

specifies a list of CIDR blocks that represent admissible IP addresses. Trident adds applicable node IPs that fall in these ranges to the export policy automatically at publish time. Alternatively, when no CIDRs are specified, all global-scoped unicast IPs found on the node that the volume being published to will be added to the export policy.

- Storage administrators can create an export policy and add rules manually. Trident uses the default export policy unless a different export policy name is specified in the configuration.

Dynamically manage export policies

Trident provides the ability to dynamically manage export policies for ONTAP backends. This provides the storage administrator the ability to specify a permissible address space for worker node IPs, rather than defining explicit rules manually. It greatly simplifies export policy management; modifications to the export policy no longer require manual intervention on the storage cluster. Moreover, this helps restrict access to the storage cluster only to worker nodes that are mounting volumes and have IPs in the range specified, supporting a fine-grained and automated management.

> Do not use Network Address Translation (NAT) when using dynamic export policies. With NAT, the storage controller sees the frontend NAT address and not the actual IP host address, so access will be denied when no match is found in the export rules.

Example

There are two configuration options that must be used. Here's an example backend definition:

```
---
version: 1
storageDriverName: ontap-nas-economy
backendName: ontap_nas_auto_export
managementLIF: 192.168.0.135
svm: svm1
username: vsadmin
password: password
autoExportCIDRs:
  - 192.168.0.0/24
autoExportPolicy: true
```

> When using this feature, you must ensure that the root junction in your SVM has a previously created export policy with an export rule that permits the node CIDR block (such as the default export policy). Always follow NetApp recommended best practice to dedicate an SVM for Trident.

Here is an explanation of how this feature works using the example above:

- `autoExportPolicy` is set to `true`. This indicates that Trident creates an export policy for each volume provisioned with this backend for the `svm1` SVM and handle the addition and deletion of rules using `autoexportCIDRs` address blocks. Until a volume is attached to a node, the volume uses an empty export policy with no rules to prevent unwanted access to that volume. When a volume is published to a node Trident creates an export policy with the same name as the underlying qtree containing the node IP within the specified CIDR block. These IPs will also be added to the export policy used by the parent

FlexVol volume

- For example:
  - backend UUID 403b5326-8482-40db-96d0-d83fb3f4daec
  - `autoExportPolicy` set to `true`
  - storage prefix `trident`
  - PVC UUID a79bcf5f-7b6d-4a40-9876-e2551f159c1c
  - qtree named trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c creates an export policy for the FlexVol named `trident-403b5326-8482-40db-96d0-d83fb3f4daec` , an export policy for the qtree named `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`, and an empty export policy named `trident_empty` on the SVM. The rules for the FlexVol export policy will be a superset of any rules contained in the qtree export policies. The empty export policy will be reused by any volumes that are not attached.

- `autoExportCIDRs` contains a list of address blocks. This field is optional and it defaults to ["0.0.0.0/0", "::/0"]. If not defined, Trident adds all globally-scoped unicast addresses found on the worker nodes with publications.

In this example, the `192.168.0.0/24` address space is provided. This indicates that Kubernetes node IPs that fall within this address range with publications will be added to the export policy that Trident creates. When Trident registers a node it runs on, it retrieves the IP addresses of the node and checks them against the address blocks provided in `autoExportCIDRs`. At publish time, after filtering the IPs, Trident creates the export policy rules for the client IPs for the node it is publishing to.

You can update `autoExportPolicy` and `autoExportCIDRs` for backends after you create them. You can append new CIDRs for a backend that is automatically managed or delete existing CIDRs. Exercise care when deleting CIDRs to ensure that existing connections are not dropped. You can also choose to disable `autoExportPolicy` for a backend and fall back to a manually created export policy. This will require setting the `exportPolicy` parameter in your backend config.

After Trident creates or updates a backend, you can check the backend using `tridentctl` or the corresponding `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
```

When a node is removed, Trident checks all export policies to remove the access rules corresponding to the node. By removing this node IP from the export policies of managed backends, Trident prevents rogue mounts, unless this IP is reused by a new node in the cluster.

For previously existing backends, updating the backend with `tridentctl update backend` ensures that Trident manages the export policies automatically. This creates two new export policies named after the backend's UUID and qtree name when they are needed. Volumes that are present on the backend will use the newly created export policies after they are unmounted and mounted again.

> Deleting a backend with auto-managed export policies will delete the dynamically created export policy. If the backend is re-created, it is treated as a new backend and will result in the creation of a new export policy.

If the IP address of a live node is updated, you must restart the Trident pod on the node. Trident will then update the export policy for backends it manages to reflect this IP change.

Prepare to provision SMB volumes

With a little additional preparation, you can provision SMB volumes using `ontap-nas` drivers.

> You must configure both NFS and SMB/CIFS protocols on the SVM to create an `ontap-nas-economy` SMB volume for ONTAP on-premises clusters. Failure to configure either of these protocols will cause SMB volume creation to fail.

> `autoExportPolicy` is not supported for SMB volumes.

Before you begin

Before you can provision SMB volumes, you must have the following.

- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials. To generate secret `smbcreds`:

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

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Steps

1. For on-premises ONTAP, you can optionally create an SMB share or Trident can create one for you.

   SMB shares are required for Amazon FSx for ONTAP.

   You can create the SMB admin shares in one of two ways either using the Microsoft Management Console Shared Folders snap-in or using the ONTAP CLI. To create the SMB shares using the ONTAP CLI:

   a. If necessary, create the directory path structure for the share.

      The `vserver cifs share create` command checks the path specified in the -path option during share creation. If the specified path does not exist, the command fails.

   b. Create an SMB share associated with the specified SVM:

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

   c. Verify that the share was created:

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

      Refer to Create an SMB share for full details.

2. When creating the backend, you must configure the following to specify SMB volumes. For all FSx for ONTAP backend configuration options, refer to FSx for ONTAP configuration options and examples.

Parameter	Description	Example
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`
`nasType`	**Must set to `smb`.** If null, defaults to `nfs`.	`smb`
`securityStyle`	Security style for new volumes. **Must be set to `ntfs` or `mixed` for SMB volumes.**	`ntfs` or `mixed` for SMB volumes
`unixPermissions`	Mode for new volumes. **Must be left empty for SMB volumes.**	""

Enable secure SMB

Beginning with the 25.06 release, NetApp Trident supports secure provisioning of SMB volumes created using `ontap-nas` and `ontap-nas-economy` backends. When secure SMB is enabled, you can provide controlled access to SMB the shares for Active Directory (AD) users and user groups using Access Control Lists (ACLs).

Points to remember

- Importing `ontap-nas-economy` volumes is not supported.
- Only read-only clones are supported for `ontap-nas-economy` volumes.
- If Secure SMB is enabled, Trident will ignore the SMB share mentioned in the backend.
- Updating the PVC annotation, storage class annotation, and backend field does not update the SMB share ACL.
- The SMB share ACL specified in the annotation of the clone PVC will take precedence over those in the source PVC.
- Ensure that you provide valid AD users while enabling secure SMB. Invalid users will not be added to the ACL.
- If you provide the same AD user in the backend, storage class, and PVC with different permissions, the permission priority will be: PVC, storage class, and then backend.
- Secure SMB is supported for `ontap-nas` managed volume imports and not applicable to unmanaged volume imports.

Steps

1. Specify adAdminUser in TridentBackendConfig as shown in the following example:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.193.176.x
  svm: svm0
  useREST: true
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

2. Add the annotation in the storage class.

   Add the `trident.netapp.io/smbShareAdUser` annotation to the storage class to enable secure SMB without fail.
   The user value specified for the annotation `trident.netapp.io/smbShareAdUser` should be the same as the username specified in the `smbcreds` secret.
   You can choose one of the following for `smbShareAdUserPermission`: `full_control`, `change`, or `read`. The default permission is `full_control`.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

3. Create a PVC.

   The following example creates a PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/snapshotDirectory: "true"
    trident.netapp.io/smbShareAccessControl: |
      read:
        - tridentADtest
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

ONTAP NAS configuration options and examples

Learn to create and use ONTAP NAS drivers with your Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses. Beginning with the 25.10 release, NetApp Trident supports NetApp AFX storage systems. NetApp AFX storage systems differ from other ONTAP-based systems (ASA, AFF, and FAS) in the implementation of their storage layer.

> Only the `ontap-nas` driver (with NFS protocol) is supported for NetApp AFX systems; SMB protocol is not supported.

Backend configuration options

In the Trident backend configuration, you need not specify that your system is an NetApp AFX storage system. When you select `ontap-nas` as the `storageDriverName`, Trident detects automatically the AFX storage system. Some backend configuration parameters are not applicable to AFX storage systems.

The following table displays backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver For NetApp AFX systems, only `ontap-nas` is supported.	`ontap-nas`, `ontap-nas-economy`, or `ontap-nas-flexgroup`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF

Parameter	Description	Default
`managementLIF`	IP address of a cluster or SVM management LIF A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. For seamless MetroCluster switchover, see the MetroCluster example.	"10.0.0.1", "[2001:1234:abcd::fefe]"
`dataLIF`	IP address of protocol LIF. NetApp recommends specifying `dataLIF`. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. Can be changed after initial setting. Refer to Update `dataLIF` after initial configuration. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. **Omit for Metrocluster.** See the MetroCluster example.	Specified address or derived from SVM, if not specified (not recommended)
`svm`	Storage virtual machine to use **Omit for Metrocluster.** See the MetroCluster example.	Derived if an SVM `managementLIF` is specified
`autoExportPolicy`	Enable automatic export policy creation and updating [Boolean]. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	false
`autoExportCIDRs`	List of CIDRs to filter Kubernetes' node IPs against when `autoExportPolicy` is enabled. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	["0.0.0.0/0", "::/0"]`

Parameter	Description	Default
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based auth	""
`username`	Username to connect to the cluster/SVM. Used for credential-based auth. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	
`password`	Password to connect to the cluster/SVM. Used for credential-based auth. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be updated after you set it When using ontap-nas-economy and a storagePrefix that is 24 or more characters, the qtrees will not have the storage prefix embedded, though it will be in the volume name.	"trident"

Parameter	Description	Default
`aggregate`	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the `ontap-nas-flexgroup` driver, this option is ignored. If not assigned, any of the available aggregates can be used to provision a FlexGroup volume. When the aggregate is updated in SVM, it is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate in Trident to provision volumes, if the aggregate is renamed or moved out of the SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate to one that is present on the SVM or remove it altogether to bring the backend back online. **Do not specify for AFX storage systems.**	""
`limitAggregateUsage`	Fail provisioning if usage is above this percentage. **Does not apply to Amazon FSx for ONTAP**. **Do not specify for AFX storage systems.**	"" (not enforced by default)
flexgroupAggregateList	List of aggregates for provisioning (optional; if set, must be assigned to the SVM). All aggregates assigned to the SVM are used to provision a FlexGroup volume. Supported for the **ontap-nas-flexgroup** storage driver. When the aggregate list is updated in SVM, the list is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate list in Trident to provision volumes, if the aggregate list is renamed or moved out of SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate list to one that is present on the SVM or remove it altogether to bring the backend back online.	""

Parameter	Description	Default
`limitVolumeSize`	Fail provisioning if requested volume size is above this value.	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.	null
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes. **If specified, always set to `nfs` for AFX storage systems**.	`nfs`
`nfsMountOptions`	Comma-separated list of NFS mount options. The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in the storage class or the configuration file, Trident will not set any mount options on an associated persistent volume.	""
`qtreesPerFlexvol`	Maximum Qtrees per FlexVol, must be in range [50, 300]	"200"
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`

Parameter	Description	Default
`useREST`	Boolean parameter to use ONTAP REST APIs. `useREST` When set to `true`, Trident uses ONTAP REST APIs to communicate with the backend; when set to `false`, Trident uses ONTAPI (ZAPI) calls to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the `ontapi` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles. Beginning with the Trident 24.06 release and ONTAP 9.15.1 or later, `useREST` is set to `true` by default; change `useREST` to `false` to use ONTAPI (ZAPI) calls. **If specified, always set to `true` for AFX storage systems**.	`true` for ONTAP 9.15.1 or later, otherwise `false`.
`limitVolumePoolSize`	Maximum requestable FlexVol size when using Qtrees in ontap-nas-economy backend.	"" (not enforced by default)
`denyNewVolumePools`	Restricts `ontap-nas-economy` backends from creating new FlexVol volumes to contain their Qtrees. Only preexisting Flexvols are used for provisioning new PVs.	
`adAdminUser`	Active Directory admin user or user group with full access to SMB shares. Use this parameter to provide admin rights to the SMB share with full control.	

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the `defaults` section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
`spaceAllocation`	Space-allocation for Qtrees	"true"
`spaceReserve`	Space reservation mode; "none" (thin) or "volume" (thick)	"none"
`snapshotPolicy`	Snapshot policy to use	"none"
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""

Parameter	Description	Default
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend. Not supported by ontap-nas-economy.	""
`snapshotReserve`	Percentage of volume reserved for snapshots	"0" if `snapshotPolicy` is "none", otherwise ""
`splitOnClone`	Split a clone from its parent upon creation	"false"
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	"false"
`tieringPolicy`	Tiering policy to use "none"	
`unixPermissions`	Mode for new volumes	"777" for NFS volumes; empty (not applicable) for SMB volumes
`snapshotDir`	Controls access to the `.snapshot` directory	`true`, `false` (Set explicitly).
`exportPolicy`	Export policy to use	"default"
`securityStyle`	Security style for new volumes. NFS supports `mixed` and `unix` security styles. SMB supports `mixed` and `ntfs` security styles.	NFS default is `unix`. SMB default is `ntfs`.
`nameTemplate`	Template to create custom volume names.	""

Using QoS policy groups with Trident requires ONTAP 9.8 or later. You should use a non-shared QoS policy group and ensure the policy group is applied to each constituent individually. A shared QoS policy group enforces the ceiling for the total throughput of all workloads.

Volume provisioning examples

Here's an example with defaults defined:

```
---
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"
```

For `ontap-nas` and `ontap-nas-flexgroups`, Trident now uses a new calculation to ensure that the FlexVol is sized correctly with the snapshotReserve percentage and PVC. When the user requests a PVC, Trident creates the original FlexVol with more space by using the new calculation. This calculation ensures that the user receives the writable space they requested for in the PVC, and not less space than what they requested. Before v21.07, when the user requests a PVC (for example, 5 GiB), with the snapshotReserve to 50 percent, they get only 2.5 GiB of writeable space. This is because what the user requested for is the whole volume and `snapshotReserve` is a percentage of that. With Trident 21.07, what the user requests for is the writeable space and Trident defines the `snapshotReserve` number as the percentage of the whole volume. This does not apply to `ontap-nas-economy`. See the following example to see how this works:

The calculation is as follows:

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

For snapshotReserve = 50%, and PVC request = 5 GiB, the total volume size is 5/.5 = 10 GiB and the available size is 5 GiB, which is what the user requested in the PVC request. The `volume show` command should show results similar to this example:

Existing backends from previous installs will provision volumes as explained above when upgrading Trident. For volumes that you created before upgrading, you should resize their volumes for the change to be observed. For example, a 2 GiB PVC with `snapshotReserve=50` earlier resulted in a volume that provides 1 GiB of writable space. Resizing the volume to 3 GiB, for example, provides the application with 3 GiB of writable space on a 6 GiB volume.

Minimal configuration examples

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

> If you are using Amazon FSx on NetApp ONTAP with Trident, the recommendation is to specify DNS names for LIFs instead of IP addresses.

ONTAP NAS economy example

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

ONTAP NAS Flexgroup example

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

MetroCluster example

You can configure the backend to avoid having to manually update the backend definition after switchover and switchback during SVM replication and recovery.

For seamless switchover and switchback, specify the SVM using `managementLIF` and omit the `dataLIF` and `svm` parameters. For example:

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

SMB volumes example

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

Certificate-based authentication example

This is a minimal backend configuration example. `clientCertificate`, `clientPrivateKey`, and `trustedCACertificate` (optional, if using trusted CA) are populated in `backend.json` and take the base64-encoded values of the client certificate, private key, and trusted CA certificate, respectively.

```
---
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_
```

Auto export policy example

This example shows you how you can instruct Trident to use dynamic export policies to create and manage the export policy automatically. This works the same for the `ontap-nas-economy` and `ontap-nas-flexgroup` drivers.

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-nasbackend
autoExportPolicy: true
autoExportCIDRs:
- 10.0.0.0/24
username: admin
password: password
nfsMountOptions: nfsvers=4
```

IPv6 addresses example

This example shows `managementLIF` using an IPv6 address.

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

Amazon FSx for ONTAP using SMB volumes example

The `smbShare` parameter is required for FSx for ONTAP using SMB volumes.

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

Backend configuration example with nameTemplate

```
---
version: 1
storageDriverName: ontap-nas
backendName: ontap-nas-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

Examples of backends with virtual pools

In the sample backend definition files shown below, specific defaults are set for all storage pools, such as `spaceReserve` at none, `spaceAllocation` at false, and `encryption` at false. The virtual pools are defined in the storage section.

Trident sets provisioning labels in the "Comments" field. Comments are set on FlexVol for `ontap-nas` or FlexGroup for `ontap-nas-flexgroup`. Trident copies all labels present on a virtual pool to the storage volume at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In these examples, some of the storage pools set their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools override the default values.

ONTAP NAS example

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
svm: svm_nfs
username: admin
password: <password>
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: "false"
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      app: msoffice
      cost: "100"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
      adaptiveQosPolicy: adaptive-premium
  - labels:
      app: slack
      cost: "75"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      app: wordpress
```

```
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS FlexGroup example

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "50000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: gold
      creditpoints: "30000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
    defaults:
```

```
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS economy example

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: nas_economy_store
region: us_east_1
storage:
  - labels:
      department: finance
      creditpoints: "6000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: engineering
      creditpoints: "3000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      department: humanresource
      creditpoints: "2000"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
```

```
      encryption: "false"
      unixPermissions: "0775"
```

Map backends to StorageClasses

The following StorageClass definitions refer to Examples of backends with virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

- The `protection-gold` StorageClass will map to the first and second virtual pool in the `ontap-nas-flexgroup` backend. These are the only pools offering gold level protection.

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

- The `protection-not-gold` StorageClass will map to the third and fourth virtual pool in the `ontap-nas-flexgroup` backend. These are the only pools offering protection level other than gold.

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

- The `app-mysqldb` StorageClass will map to the fourth virtual pool in the `ontap-nas` backend. This is the only pool offering storage pool configuration for mysqldb type app.

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

- The `protection-silver-creditpoints-20k` StorageClass will map to the third virtual pool in the `ontap-nas-flexgroup` backend. This is the only pool offering silver-level protection and 20000 creditpoints.

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

- The `creditpoints-5k` StorageClass will map to the third virtual pool in the `ontap-nas` backend and the second virtual pool in the `ontap-nas-economy` backend. These are the only pool offerings with 5000 creditpoints.

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

Trident will decide which virtual pool is selected and ensures the storage requirement is met.

Update `dataLIF` after initial configuration

You can change the dataLIF after initial configuration by running the following command to provide the new backend JSON file with updated dataLIF.

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

If PVCs are attached to one or multiple pods, you must bring down all corresponding pods and then bring them back up in order to for the new dataLIF to take effect.

Secure SMB examples

Backend configuration with ontap-nas driver

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Backend configuration with ontap-nas-economy driver

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas-economy
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Backend configuration with storage pool

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm0
  useREST: false
  storage:
  - labels:
      app: msoffice
    defaults:
      adAdminUser: tridentADuser
  nasType: smb
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Storage class example with ontap-nas driver

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADtest
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Ensure that you add `annotations` to enable secure SMB. Secure SMB does not work without the annotations, irrespective of configurations set in the Backend or PVC.

Storage class example with ontap-nas-economy driver

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser3
parameters:
  backendType: ontap-nas-economy
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

PVC example with a single AD user

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      change:
        - tridentADtest
      read:
        - tridentADuser
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

PVC example with multiple AD users

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-test-pvc
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      full_control:
        - tridentTestuser
        - tridentuser
        - tridentTestuser1
        - tridentuser1
      change:
        - tridentADuser
        - tridentADuser1
        - tridentADuser4
        - tridentTestuser2
      read:
        - tridentTestuser2
        - tridentTestuser3
        - tridentADuser2
        - tridentADuser3
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Amazon FSx for NetApp ONTAP

Use Trident with Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP is a fully managed AWS service that runs file systems powered by the NetApp ONTAP storage operating system. It provides ONTAP features, performance, and administration with the scalability and operational simplicity of AWS. A file system is the primary resource in Amazon FSx and is analogous to an on-premises ONTAP cluster. Each file system contains one or more storage virtual machines (SVMs), and each SVM contains one or more volumes that store files and directories. This integration enables Kubernetes clusters running in Amazon Elastic Kubernetes Service (EKS) to provision ONTAP-backed persistent volumes for block and file workloads.

Requirements

In addition to Trident requirements, to integrate FSx for ONTAP with Trident, you need:

- An existing Amazon EKS cluster or self-managed Kubernetes cluster with `kubectl` installed.

- An existing Amazon FSx for NetApp ONTAP file system and storage virtual machine (SVM) that is reachable from your cluster's worker nodes.
- Worker nodes that are prepared for NFS or iSCSI.

> Ensure you follow the node preparation steps required for Amazon Linux and Ubuntu Amazon Machine Images (AMIs) depending on your EKS AMI type.

Considerations

- SMB volumes:
  - SMB volumes are supported using the `ontap-nas` driver only.
  - SMB volumes are not supported with Trident EKS add-on.
  - Trident supports SMB volumes mounted to pods running on Windows nodes only. Refer to Prepare to provision SMB volumes for details.
- Prior to Trident 24.02, volumes created on Amazon FSx file systems that have automatic backups enabled, could not be deleted by Trident. To prevent this issue in Trident 24.02 or later, specify the `fsxFilesystemID`, AWS `apiRegion`, AWS `apikey`, and AWS `secretKey` in the backend configuration file for AWS FSx for ONTAP.

> If you are specifying an IAM role to Trident, then you can omit specifying the `apiRegion`, `apiKey`, and `secretKey` fields to Trident explicitly. For more information, refer to FSx for ONTAP configuration options and examples.

Simultaneous usage of Trident SAN/iSCSI and EBS-CSI driver

If you plan to use ontap-san drivers (e.g., iSCSI) with AWS (EKS, ROSA, EC2, or any other instance), the multipath configuration required on the nodes might conflict with the Amazon Elastic Block Store (EBS) CSI driver. To ensure that multipathing functions without interfering with EBS disks on the same node, you need to exclude EBS in your multipathing setup. This example shows a `multipath.conf` file that includes the required Trident settings while excluding EBS disks from multipathing:

```
defaults {
    find_multipaths no
}
blacklist {
    device {
        vendor "NVME"
        product "Amazon Elastic Block Store"
    }
}
```

Authentication

Trident offers two modes of authentication.

- Credential-based(Recommended): Stores credentials securely in AWS Secrets Manager. You can use the `fsxadmin` user for your file system or the `vsadmin` user configured for your SVM.

Trident expects to be run as a `vsadmin` SVM user or as a user with a different name that has the same role. Amazon FSx for NetApp ONTAP has an `fsxadmin` user that is a limited replacement of the ONTAP `admin` cluster user. We strongly recommend using `vsadmin` with Trident.

- Certificate-based: Trident will communicate with the SVM on your FSx file system using a certificate installed on your SVM.

For details on enabling authentication, refer to the authentication for your driver type:

- ONTAP NAS authentication
- ONTAP SAN authentication

Tested Amazon Machine Images (AMIs)

EKS cluster supports various operating systems, but AWS has optimized certain Amazon Machine Images (AMIs) for containers and EKS. The following AMIs have been tested with NetApp Trident 25.02.

AMI	NAS	NAS-economy	iSCSI	iSCSI-economy
AL2023_x86_64_STANDARD	Yes	Yes	Yes	Yes
AL2_x86_64	Yes	Yes	Yes*	Yes*
BOTTLEROCKET_x86_64	Yes**	Yes	N/A	N/A
AL2023_ARM_64_STANDARD	Yes	Yes	Yes	Yes
AL2_ARM_64	Yes	Yes	Yes*	Yes*
BOTTLEROCKET_ARM_64	Yes**	Yes	N/A	N/A

- * Unable to delete the PV without restarting the node
- ** Doesn't work with NFSv3 with Trident version 25.02.

If your desired AMI is not listed here, it does not mean that it is not supported; it simply means it has not been tested. This list serves as a guide for AMIs are known to work.

Tests performed with:

- EKS version: 1.32
- Installation Method: Helm 25.06 and as an AWS add-On 25.06
- For NAS both NFSv3 and NFSv4.1 were tested.
- For SAN only iSCSI was tested, not NVMe-oF.

Tests performed:

- Create: Storage Class, pvc, pod
- Delete: pod, pvc (regular, qtree/lun – economy, NAS with AWS backup)

Find more information

- Amazon FSx for NetApp ONTAP documentation
- Blog post on Amazon FSx for NetApp ONTAP

Create an IAM role and AWS Secret

You can configure Kubernetes pods to access AWS resources by authenticating as an AWS IAM role instead of by providing explicit AWS credentials.

To authenticate using an AWS IAM role, you must have a Kubernetes cluster deployed using EKS.

Create AWS Secrets Manager secret

Since Trident will be issuing APIs against an FSx vserver to manage the storage for you, it will need credentials to do so. The secure way to pass those credentials is through an AWS Secrets Manager secret. Therefore, if you don't already have one, you'll need to create an AWS Secrets Manager secret that contains the credentials for the vsadmin account.

This example creates an AWS Secrets Manager secret to store Trident CSI credentials:

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

Create IAM Policy

Trident also needs AWS permissions to run correctly. Therefore, you need to create a policy that gives Trident the permissions it needs.

The following examples creates an IAM policy using the AWS CLI:

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

Policy JSON example:

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

Create Pod Identity or IAM role for Service account association (IRSA)

You can configure a Kubernetes service account to assume an AWS Identity and Access Management (IAM) role with EKS Pod Identity or IAM role for Service account association (IRSA). Any Pods that are configured to use the service account can then access any AWS service that the role has permissions to access.

Pod Identity

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

Install Pod Identity on your EKS cluster:

You can create Pod identity via the AWS console or using the following AWS CLI command:

```
aws eks create-addon --cluster-name <EKS_CLUSTER_NAME> --addon-name
eks-pod-identity-agent
```

For more information refer to Set up the Amazon EKS Pod Identity Agent.

Create trust-relationship.json:

Create trust-relationship.json to enable EKS Service Principal to assume this role for Pod Identity. Then create a role with this trust policy:

```
aws iam create-role \
  --role-name fsxn-csi-role --assume-role-policy-document file://trust-
relationship.json \
  --description "fsxn csi pod identity role"
```

trust-relationship.json file:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

Attach the role policy to the IAM role:

Attach the role policy from the previous step to the IAM role that was created:

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:111122223333:policy/fsxn-csi-policy \
  --role-name fsxn-csi-role
```

Create a pod identity association:

Create a pod identity association between IAM role and the Trident service account(trident-controller)

```
aws eks create-pod-identity-association \
  --cluster-name <EKS_CLUSTER_NAME> \
  --role-arn arn:aws:iam::111122223333:role/fsxn-csi-role \
  --namespace trident --service-account trident-controller
```

IAM role for Service account association (IRSA)

Using the AWS CLI:

```
aws iam create-role --role-name AmazonEKS_FSxN_CSI_DriverRole \
  --assume-role-policy-document file://trust-relationship.json
```

trust-relationship.json file:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<account_id>:oidc-
provider/<oidc_provider>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<oidc_provider>:aud": "sts.amazonaws.com",
          "<oidc_provider>:sub":
"system:serviceaccount:trident:trident-controller"
        }
      }
    }
  ]
}
```

Update the following values in the `trust-relationship.json` file:

- **<account_id>** - Your AWS account ID
- **<oidc_provider>** - The OIDC of your EKS cluster. You can obtain the oidc_provider by running:

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

Attach the IAM role with the IAM policy:

Once the role has been created, attach the policy (that was created in the step above) to the role using this command:

```
aws iam attach-role-policy --role-name my-role --policy-arn <IAM policy
ARN>
```

Verify OICD provider is associated:

Verify that your OIDC provider is associated with your cluster. You can verify it using this command:

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

If the output is empty, use the following command to associate IAM OIDC to your cluster:

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

If you are using eksctl, use the following example to create an IAM role for service account in EKS:

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

Install Trident

Trident streamlines Amazon FSx for NetApp ONTAP storage management in Kubernetes to enable your developers and administrators focus on application deployment. You can install Trident using one of the following methods:

- Helm
- EKS add-on

If you want to make use of the snapshot functionality, install the CSI snapshot controller add-on. Refer to Enable snapshot functionality for CSI volumes for more information.

Install Trident via helm

Pod Identity

1. Add the Trident Helm repository:

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

2. Install Trident using the following example:

```
helm install trident-operator netapp-trident/trident-operator --version 100.2502.1 --namespace trident --create-namespace
```

You can use the `helm list` command to review installation details such as name, namespace, chart, status, app version, and revision number.

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2502.0    25.02.0
```

Service account association (IRSA)

1. Add the Trident Helm repository:

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

2. Set the values for **cloud provider** and **cloud identity**:

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1  \
--set cloudProvider="AWS" \
--set cloudIdentity="'eks.amazonaws.com/role-arn:
arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'" \
--namespace trident \
--create-namespace
```

You can use the `helm list` command to review installation details such as name, namespace, chart, status, app version, and revision number.

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2510.0    25.10.0
```

If you're planning to use iSCSI, make sure iSCSI is enabled on your client machine. If you're using AL2023 Worker node OS, you can automate the installation of the iSCSI client by adding the node prep parameter in the helm installation:

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1 --namespace trident --create-namespace --
set nodePrep={iscsi}
```

Install Trident via the EKS add-on

The Trident EKS add-on includes the latest security patches, bug fixes, and is validated by AWS to work with Amazon EKS. The EKS add-on enables you to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons.

Prerequisites

Ensure that you have the following before configuring the Trident add-on for AWS EKS:

- An Amazon EKS cluster account with add-on subscription
- AWS permissions to the AWS marketplace:
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI type: Amazon Linux 2 (AL2_x86_64) or Amazon Linux 2 Arm(AL2_ARM_64)
- Node type: AMD or ARM
- An existing Amazon FSx for NetApp ONTAP file system

Enable the Trident add-on for AWS

Management console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. On the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to configure the NetApp Trident CSI add-on for.
4. Select **Add-ons** and then select **Get more add-ons**.
5. Follow these steps to select the add-on:
   a. Scroll down to the **AWS Marketplace add-ons** section and type **"Trident"** in the search box.
   b. Select the check box at the top right corner of the Trident by NetApp box.
   c. Select **Next**.
6. On the **Configure selected add-ons** settings page, do the following:

   Skip these steps if you are using Pod Identity association.

   a. Select the **Version** you would like to use.
   b. If you're using IRSA authentication, make sure to set configuration values available in the Optional configuration settings:
      - Select the **Version** you would like to use.
      - Follow the **Add-on configuration schema** and set the **configurationValues** parameter on the **Configuration values** section to the role-arn you created on the previous step (Value should be in the following format):

```
{

  "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
  "cloudProvider": "AWS"

}
```

   If you select Override for the Conflict resolution method, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

7. Choose **Next**.
8. On the **Review and add** page, choose **Create**.

   After the add-on installation is complete, you see your installed add-on.

AWS CLI

1. Create the `add-on.json` file:

For Pod Identity, use the following format:

Use the

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
}
```

For IRSA authentication, use the following format:

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
  "serviceAccountRoleArn": "<role ARN>",
  "configurationValues": {
    "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
    "cloudProvider": "AWS"
  }
}
```

Replace `<role ARN>` with the ARN of the role that was created in the previous step.

2. Install the Trident EKS add-on.

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

eksctl

The following example command installs the Trident EKS add-on:

```
eksctl create addon --name netapp_trident-operator --cluster
<cluster_name> --force
```

Update the Trident EKS add-on

Management console

1. Open the Amazon EKS console https://console.aws.amazon.com/eks/home#/clusters.
2. On the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to update the NetApp Trident CSI add-on for.
4. Select the **Add-ons** tab.
5. Select **Trident by NetApp** and then select **Edit**.
6. On the **Configure Trident by NetApp** page, do the following:
   a. Select the **Version** you would like to use.
   b. Expand the **Optional configuration settings** and modify as needed.
   c. Select **Save changes**.

AWS CLI

The following example updates the EKS add-on:

```
aws eks update-addon --cluster-name <eks_cluster_name> --addon-name
netapp_trident-operator --addon-version v25.6.0-eksbuild.1 \
  --service-account-role-arn <role-ARN> --resolve-conflict preserve \
  --configuration-values "{\"cloudIdentity\":
\"'eks.amazonaws.com/role-arn: <role ARN>'\"}"
```

eksctl

- Check the current version of your FSxN Trident CSI add-on. Replace `my-cluster` with your cluster name.

```
eksctl get addon --name netapp_trident-operator --cluster my-cluster
```

Example output:

```
NAME                          VERSION              STATUS    ISSUES
IAMROLE    UPDATE AVAILABLE    CONFIGURATION VALUES
netapp_trident-operator    v25.6.0-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- Update the add-on to the version returned under UPDATE AVAILABLE in the output of the previous step.

```
eksctl update addon --name netapp_trident-operator --version
v25.6.0-eksbuild.1 --cluster my-cluster --force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the Amazon EKS add-on fails; you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on does not manage settings that you need to manage, because those settings are overwritten with this option.
For more information about other options for this setting, see Addons.
For more information about Amazon EKS Kubernetes field management, see Kubernetes field management.

Uninstall/remove the Trident EKS add-on

You have two options for removing an Amazon EKS add-on:

- **Preserve add-on software on your cluster** – This option removes Amazon EKS management of any settings. It also removes the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update. However, it preserves the add-on software on your cluster. This option makes the add-on a self-managed installation, rather than an Amazon EKS add-on. With this option, there's no downtime for the add-on. Retain the `--preserve` option in the command to preserve the add-on.
- **Remove add-on software entirely from your cluster** – NetApp recommends that you remove the Amazon EKS add-on from your cluster only if there are no resources on your cluster that are dependent on it. Remove the `--preserve` option from the `delete` command to remove the add-on.

If the add-on has an IAM account associated with it, the IAM account is not removed.

Management console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. In the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to remove the NetApp Trident CSI add-on for.
4. Select the **Add-ons** tab and then select **Trident by NetApp**.*
5. Select **Remove**.
6. In the **Remove netapp_trident-operator confirmation** dialog, do the following:
   a. If you want Amazon EKS to stop managing settings for the add-on, select **Preserve on cluster**. Do this if you want to retain the add-on software on your cluster so that you can manage all of the settings of the add-on on your own.
   b. Enter **netapp_trident-operator**.
   c. Select **Remove**.

AWS CLI

Replace `my-cluster` with the name of your cluster, and then run the following command.

```
aws eks delete-addon --cluster-name my-cluster --addon-name
netapp_trident-operator --preserve
```

eksctl

The following command uninstalls the Trident EKS add-on:

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

Configure a storage class

The Kubernetes StorageClass object identifies a provisioner and instructs the provisioner how to provision volumes. This section shows you how to configure a Kubernetes StorageClass object that specifies Trident as the provisioner.

Create a StorageClass Object

When you create a StorageClass for FSx for ONTAP, Trident will automatically create the backend configuration.

> If you'd like to manually configure the storage backend, please refer to the [create-a-kubernetes-storageclass-without-automatic-backend-configuration] section for how to create the Trident backend and storage class separately.

Specify required StorageClass parameters

The following three parameters need to be defined when creating a StorageClass:

Parameter	Required	Type	Description
`fsxFilesystemID`	Yes	string	FSx for NetApp ONTAP filesystem ID
`storageDriverName`	Yes	string	Trident storage driver (for example, `ontap-nas` or `ontap-san`)
`credentialsName`	Yes	string	Name of the Kubernetes Secret that contains FSx for ONTAP credentials

Specify optional parameters

You can pass optional backend parameters through the StorageClass. Define all optional values as strings in the StorageClass `parameters` section. For a complete list of backend parameters, see: FSx for NetApp ONTAP backend configuration.

Example StorageClass configuration files.

The following example shows a StorageClass that triggers automatic backend configuration.

YAML

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-fsx-demo
  annotations:
    description: "Demo StorageClass for FSx for NetApp ONTAP"
provisioner: csi.trident.netapp.io
parameters:
  fsxFilesystemID: "fs-0abc123"
  storageDriverName: "ontap-nas"
  credentialsName: trident-fsx-credentials
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

Create the StorageClass

Once you have created you configuration file, run the following command to create the storage class.

```
kubectl create -f storage-class-ontapnas.yaml
```

You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
```

```
NAME         PROVISIONER             AGE
basic-csi    csi.trident.netapp.io   15h
```

After you apply the StorageClass, Trident creates the backend automatically. You can then create PersistentVolumeClaims that reference this StorageClass.

Verify backend configuration status

Trident records the result of backend creation in StorageClass annotations.

Annotation	Description
`trident.netapp.io/configuratorStatus`	Configuration result (`Success` or `Failure`)
`trident.netapp.io/configuratorMessage`	Detailed status or error message
`trident.netapp.io/configuratorName`	Name of the internal configurator resource
`trident.netapp.io/managed`	Indicates the StorageClass is managed by Trident
`trident.netapp.io/additionalStoragePools`	Storage pools created for this backend

To verify status, run:

```
kubectl get storageclass ontap-fsx-demo -o yaml
```

Confirm that `trident.netapp.io/configuratorStatus` is set to `Success`. If the value is `Failure`, review `trident.netapp.io/configuratorMessage` for the error.

Add additional FSxN file systems

If you need additional storage capacity while continuing to use the same StorageClass, add additional FSxN file system IDs.

Edit the StorageClass and add the following annotation:

```
metadata:
  annotations:
    trident.netapp.io/additionalFsxNFileSystemID: '["fs-
xxxxxxxxxxxxxxxxxx"]'
```

After you apply the change, Trident updates the backend configuration and updates the StorageClass annotations.

Operational considerations and limitations

- Deleting a StorageClass that as the automatic backend configuration usually deletes the associated Trident backend. This can disrupt storage connectivity and break running workloads. Validate the impact before you delete a managed StorageClass.
- Automatic backend configuration is supported only for AWS FSx for NetApp ONTAP.

Create a Kubernetes StorageClass without automatic backend configuration

If you want to create the Trident backend and StorageClass separately then follow these steps.

Understand how automatic backend configuration works

Trident derives backend configuration from the StorageClass definition. When you apply the StorageClass, Trident validates the required parameters, creates the backend, and annotates the StorageClass with status.

Trident creates the VolumeSnapshotClass only once. Trident reuses the same VolumeSnapshotClass for subsequent StorageClasses.

Create the Trident backend

To create a Trident backend, you need to create a configuration file in either JSON or YAML format. The file needs to specify the type of storage you want (NAS or SAN), the file system, and SVM to get it from and how to authenticate with it. The following example shows how to define NAS-based storage and using an AWS secret to store the credentials to the SVM you want to use:

YAML

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

JSON

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

FSx for ONTAP driver details

You can integrate Trident with Amazon FSx for NetApp ONTAP using the following drivers:

Driver Name	Description
`ontap-san`	Each PV provisioned is a LUN within its own Amazon FSx for NetApp ONTAP volume. Recommended for block storage.
`ontap-nas`	Each PV provisioned is a full Amazon FSx for NetApp ONTAP volume. Recommended for NFS and SMB.
`ontap-san-economy`	Each PV provisioned is a LUN with a configurable number of LUNs per Amazon FSx for NetApp ONTAP volume.
`ontap-nas-economy`	Each PV provisioned is a qtree, with a configurable number of qtrees per Amazon FSx for NetApp ONTAP volume.
`ontap-nas-flexgroup`	Each PV provisioned is a full Amazon FSx for NetApp ONTAP FlexGroup volume.

For driver details, refer to NAS drivers and SAN drivers.

Create the backend

After creating the configuration file run the following commands to create and validate the Trident Backend Configuration (TBC):

- Create trident backend configuration (TBC) from yaml file and run the following command:

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- Validate the trident backend configuration (TBC) was created successfully:

```
Kubectl get tbc -n trident
```

```
NAME                          BACKEND NAME          BACKEND UUID
PHASE   STATUS

backend-tbc-ontap-nas         tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9   Bound   Success
```

For more information on other configuration options, see the [Backend-advanced-configuration-and-examples] section below.

Configure a Storage Class without automatic backend configuration

The following are examples of Storage Class configurations for use with Trident and FSx for ONTAP.

Storage Class for NFS

You can use this example to setup StorageClass for volumes using NFS (Refer to Trident Attribute section below for the full list of attributes):

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

Storage Calss for iSCSI

Use this example to setup StorageClass for volumes using iSCSI:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

Storage Class using NFSv3 and AWS Bottlerocket

To provision NFSv3 volumes on AWS Bottlerocket, add the required `mountOptions` to the storage class:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

Trident StorageClass attributes

These parameters determine which Trident-managed storage pools should be utilized to provision volumes of a given type.

Attribute	Type	Values	Offer	Request	Supported by
media[1]	string	hdd, hybrid, ssd	Pool contains media of this type; hybrid means both	Media type specified	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san
provisioningType	string	thin, thick	Pool supports this provisioning method	Provisioning method specified	thick: all ontap; thin: all ontap & solidfire-san
backendType	string	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san, azure-netapp-files, ontap-san-economy	Pool belongs to this type of backend	Backend specified	All drivers
snapshots	bool	true, false	Pool supports volumes with snapshots	Volume with snapshots enabled	ontap-nas, ontap-san, solidfire-san
clones	bool	true, false	Pool supports cloning volumes	Volume with clones enabled	ontap-nas, ontap-san, solidfire-san

Attribute	Type	Values	Offer	Request	Supported by
encryption	bool	true, false	Pool supports encrypted volumes	Volume with encryption enabled	ontap-nas, ontap-nas-economy, ontap-nas-flexgroups, ontap-san
IOPS	int	positive integer	Pool is capable of guaranteeing IOPS in this range	Volume guaranteed these IOPS	solidfire-san

[1]: Not supported by ONTAP Select or FSx for ONTAP systems

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Create the storage class

Once you have configured the StorageClass, you can create it in Kubernetes.

Steps

1. This is a Kubernetes object, so use `kubectl` to create it in Kubernetes.

```
kubectl create -f storage-class-ontapnas.yaml
```

2. You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

Provision SMB volumes

You can provision SMB volumes using the `ontap-nas` driver.
However, to do so you must complete these steps: Prepare to provision SMB volumes.

Backend advanced configuration and examples

See the following table for the backend configuration options:

Parameter	Description	Example
`version`		Always 1

Parameter	Description	Example
`storageDriverName`	Name of the storage driver	`ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]. If you provide the `fsxFilesystemID` under the `aws` field, you need not to provide the `managementLIF` because Trident retrieves the SVM `managementLIF` information from AWS. So, you must provide credentials for a user under the SVM (For example: vsadmin) and the user must have the `vsadmin` role.	"10.0.0.1", "[2001:1234:abcd::fefe]"

Parameter	Description	Example
`dataLIF`	IP address of protocol LIF. **ONTAP NAS drivers**: NetApp recommends specifying dataLIF. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. Can be changed after initial setting. **ONTAP SAN drivers**: Do not specify for iSCSI. Trident uses ONTAP Selective LUN Map to discover the iSCI LIFs needed to establish a multi path session. A warning is generated if dataLIF is explicitly defined. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555].	
`autoExportPolicy`	Enable automatic export policy creation and updating [Boolean]. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	`false`
`autoExportCIDRs`	List of CIDRs to filter Kubernetes' node IPs against when `autoExportPolicy` is enabled. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	"["0.0.0.0/0", "::/0"]"
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""

Parameter	Description	Example
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based authentication.	""
`username`	Username to connect to the cluster or SVM. Used for credential-based authentication. For example, vsadmin.	
`password`	Password to connect to the cluster or SVM. Used for credential-based authentication.	
`svm`	Storage virtual machine to use	Derived if an SVM managementLIF is specified.
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be modified after creation. To update this parameter, you will need to create a new backend.	`trident`
`limitAggregateUsage`	**Do not specify for Amazon FSx for NetApp ONTAP.** The provided `fsxadmin` and `vsadmin` do not contain the permissions required to retrieve aggregate usage and limit it using Trident.	Do not use.
`limitVolumeSize`	Fail provisioning if requested volume size is above this value. Also restricts the maximum size of the volumes it manages for qtrees and LUNs, and the `qtreesPerFlexvol` option allows customizing the maximum number of qtrees per FlexVol volume	"" (not enforced by default)
`lunsPerFlexvol`	Maximum LUNs per Flexvol volume, must be in range [50, 200]. SAN only.	“100”

Parameter	Description	Example
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.	null
`nfsMountOptions`	Comma-separated list of NFS mount options. The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in the storage class or the configuration file, Trident will not set any mount options on an associated persistent volume.	""
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb`, or null. **Must set to `smb` for SMB volumes.** Setting to null defaults to NFS volumes.	`nfs`
`qtreesPerFlexvol`	Maximum Qtrees per FlexVol volume, must be in range [50, 300]	`"200"`
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI or a name to allow Trident to create the SMB share. This parameter is required for Amazon FSx for ONTAP backends.	`smb-share`

Parameter	Description	Example
useREST	Boolean parameter to use ONTAP REST APIs. When set to true, Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the ontap application. This is satisfied by the pre-defined vsadmin and cluster-admin roles.	false
aws	You can specify the following in the configuration file for AWS FSx for ONTAP: - fsxFilesystemID: Specify the ID of the AWS FSx file system. - apiRegion: AWS API region name. - apikey: AWS API key. - secretKey: AWS secret key.	"" "" ""
credentials	Specify the FSx SVM credentials to store in AWS Secrets Manager. - name: Amazon Resource Name (ARN) of the secret, which contains the credentials of SVM. - type: Set to awsarn. Refer to Create an AWS Secrets Manager secret for more information.	

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the defaults section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
spaceAllocation	Space-allocation for LUNs	true
spaceReserve	Space reservation mode; "none" (thin) or "volume" (thick)	none
snapshotPolicy	Snapshot policy to use	none

Parameter	Description	Default
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend. Using QoS policy groups with Trident requires ONTAP 9.8 or later. You should use a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group enforces the ceiling for the total throughput of all workloads.	""
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend. Not supported by ontap-nas-economy.	""
`snapshotReserve`	Percentage of volume reserved for snapshots "0"	If `snapshotPolicy` is `none`, `else` ""
`splitOnClone`	Split a clone from its parent upon creation	`false`
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	`false`
`luksEncryption`	Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS). SAN only.	""
`tieringPolicy`	Tiering policy to use `none`	
`unixPermissions`	Mode for new volumes. **Leave empty for SMB volumes.**	""

Parameter	Description	Default
`securityStyle`	Security style for new volumes. NFS supports `mixed` and `unix` security styles. SMB supports `mixed` and `ntfs` security styles.	NFS default is `unix`. SMB default is `ntfs`.

Configure a PVC

This sections includes instructions on how to create a PersistentVolumeClaim (PVC) that uses the configured Kubernetes StorageClass to request a PV.
If successful, you can then mount the PV to a pod.

Create the PVC

A *PersistentVolumeClaim* (PVC) is a request for access to the PersistentVolume on the cluster. The PVC can be configured to request storage of a certain size or access mode. Using the associated StorageClass, the cluster administrator can control more than PersistentVolume size and access mode—such as performance or service level.

After you create the Trident backend and StorageClass you can create a PVC. After the PVC is created, you can mount the volume in a pod.

Sample manifests

The following examples show basic PVC configuration options.

PVC with RWX access

This example shows a basic PVC with RWX access that is associated with a StorageClass named `basic-csi`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-gold
```

PVC using iSCSI example

This example shows a basic PVC for iSCSI with RWO access that is associated with a StorageClass named `protection-gold`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: protection-gold
```

Create PVC

Steps

1. Create the PVC.

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

2. Verify the PVC status.

```
kubectl get pvc
```

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

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Deploy an application

When the storage class and PVC are created, you can mount the PV to a pod. This section lists the example command and configuration to attach the PV to a pod.

Deploy a sample application

Steps

1. Mount the volume in a pod.

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

These examples show basic configurations to attach the PVC to a pod:

Basic configuration:

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

You can monitor the progress using `kubectl get pod --watch`.

2. Verify that the volume is mounted on `/my/mount/path`.

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

```
Filesystem                                                      Size
Used  Avail Use%  Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1%   /my/mount/path
```

You can now delete the Pod. The Pod application will no longer exist, but the volume will remain.

```
kubectl delete pod pv-pod
```

Configure the Trident EKS add-on on an EKS cluster

NetApp Trident streamlines Amazon FSx for NetApp ONTAP storage management in Kubernetes to enable your developers and administrators focus on application deployment. The NetApp Trident EKS add-on includes the latest security patches, bug fixes, and is validated by AWS to work with Amazon EKS. The EKS add-on enables you

to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons.

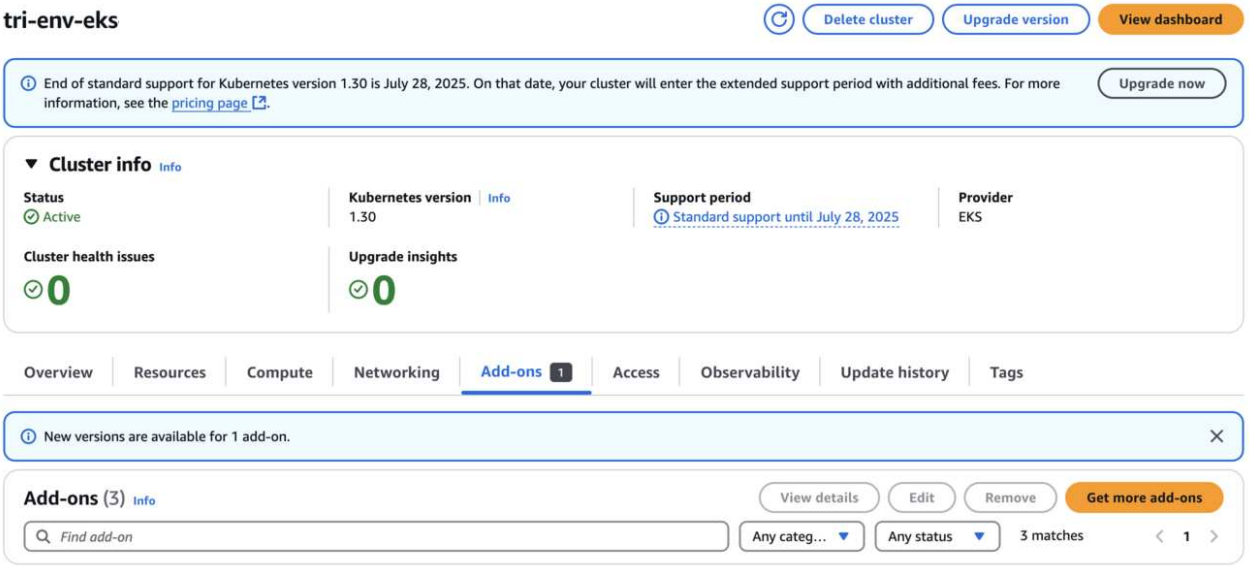
Prerequisites

Ensure that you have the following before configuring the Trident add-on for AWS EKS:

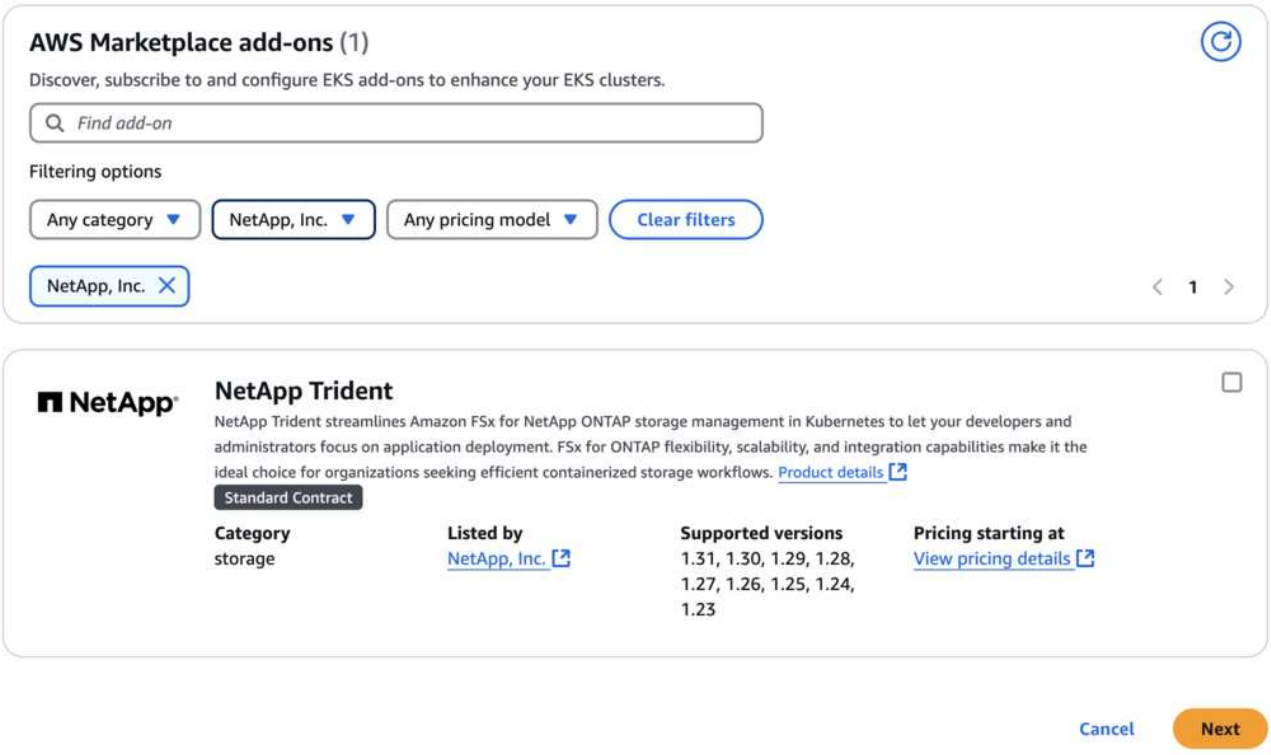
- An Amazon EKS cluster account with permissions to work with add-ons. Refer to Amazon EKS add-ons.
- AWS permissions to the AWS marketplace:
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI type: Amazon Linux 2 (AL2_x86_64) or Amazon Linux 2 Arm(AL2_ARM_64)
- Node type: AMD or ARM
- An existing Amazon FSx for NetApp ONTAP file system

Steps

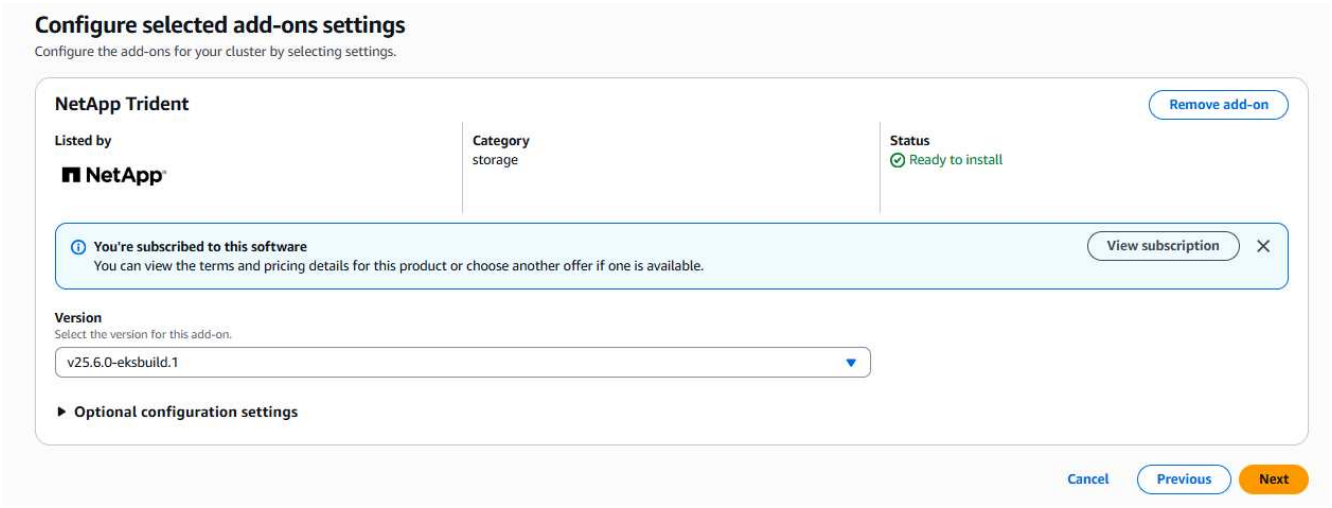
1. Make sure to create IAM role and AWS secret to enable EKS pods to access AWS resources. For instructions, see Create an IAM role and AWS Secret.
2. On your EKS Kubernetes cluster, navigate to the **Add-ons** tab.



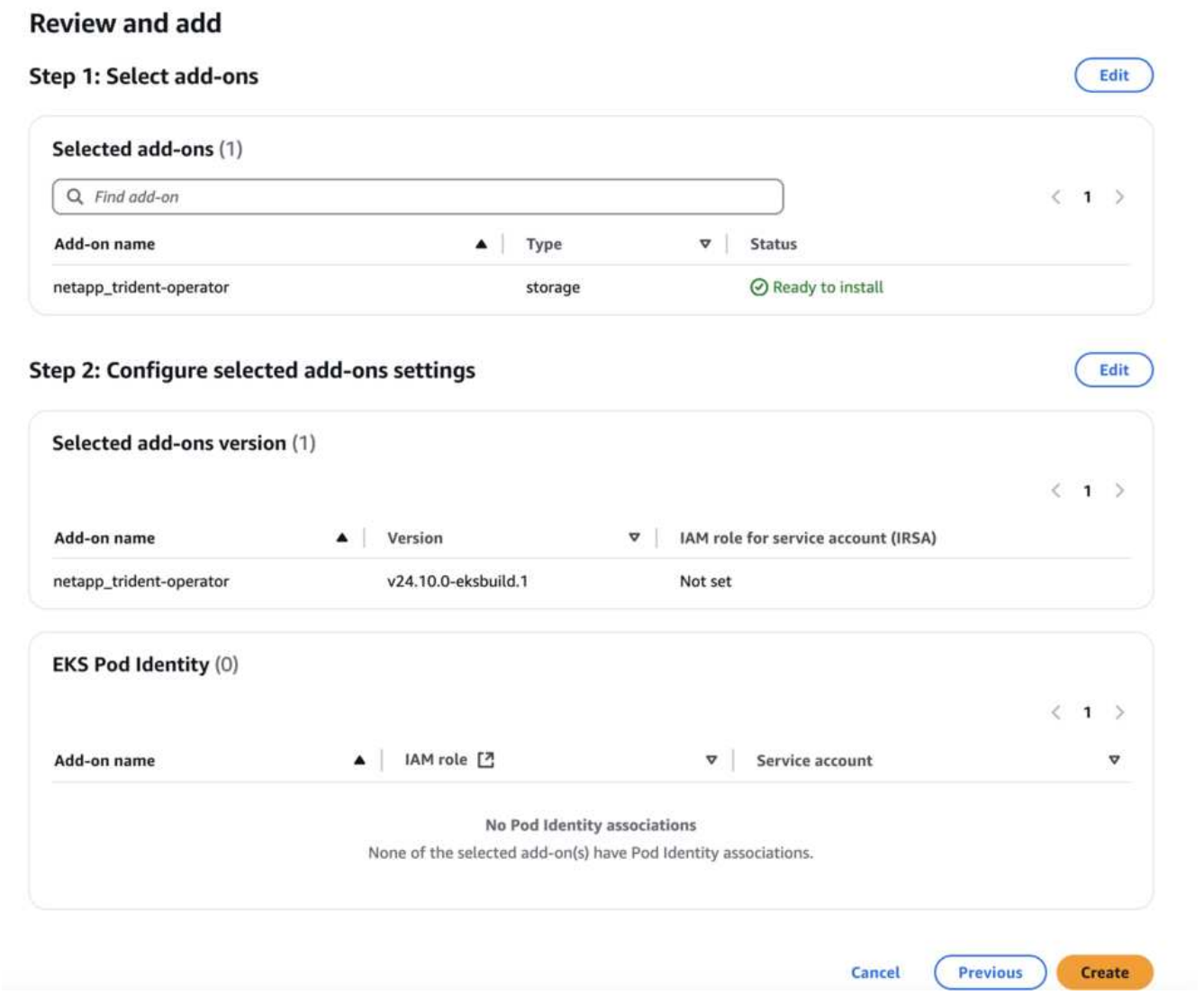
3. Go to **AWS Marketplace add-ons** and choose the *storage* category.



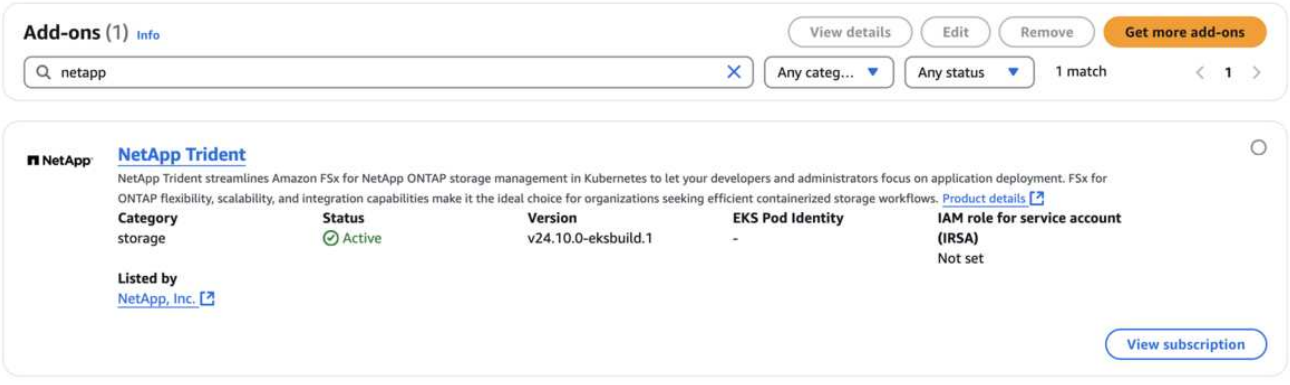
4. Locate **NetApp Trident** and select the checkbox for the Trident add-on, and click **Next**.
5. Choose the desired version of the add-on.



6. Configure the required add-on settings.



7. If you are using IRSA (IAM roles for service account), refer to the additional configuration steps here.
8. Select **Create**.
9. Verify that the status of the add-on is *Active*.



10. Run the following command to verify that Trident is properly installed on the cluster:

```
kubectl get pods -n trident
```

11. Continue the setup and configure the storage backend. For information, see Configure the Storage Backend.

Install/uninstall the Trident EKS add-on using CLI

Install the NetApp Trident EKS add-on using CLI:

The following example command installs the Trident EKS add-on:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.0-eksbuild.1` (with a dedicated version)

The following example command installs the Trident EKS add-on version 25.6.1:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.1-eksbuild.1` (with a dedicated version)

The following example command installs the Trident EKS add-on version 25.6.2:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.2-eksbuild.1` (with a dedicated version)

Uninstall the NetApp Trident EKS add-on using CLI:

The following command uninstalls the Trident EKS add-on:

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

Create backends with kubectl

A backend defines the relationship between Trident and a storage system. It tells Trident how to communicate with that storage system and how Trident should provision volumes from it. After Trident is installed, the next step is to create a backend. The `TridentBackendConfig` Custom Resource Definition (CRD) enables you to create and manage Trident backends directly through the Kubernetes interface. You can do this by using `kubectl` or the equivalent CLI tool for your Kubernetes distribution.

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`, `tbconfig`, `tbackendconfig`) is a frontend, namespaced CRD that enables you to manage Trident backends using `kubectl`. Kubernetes and storage admins can now create and manage backends directly through the Kubernetes CLI without requiring a dedicated command-line utility (`tridentctl`).

Upon the creation of a `TridentBackendConfig` object, the following happens:

- A backend is created automatically by Trident based on the configuration you provide. This is represented internally as a `TridentBackend` (`tbe`, `tridentbackend`) CR.
- The `TridentBackendConfig` is uniquely bound to a `TridentBackend` that was created by Trident.

Each `TridentBackendConfig` maintains a one-to-one mapping with a `TridentBackend`. The former is the interface provided to the user to design and configure backends; the latter is how Trident represents the actual backend object.

> `TridentBackend` CRs are created automatically by Trident. You **should not** modify them. If you want to make updates to backends, do this by modifying the `TridentBackendConfig` object.

See the following example for the format of the `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
```

You can also take a look at the examples in the trident-installer directory for sample configurations for the desired storage platform/service.

The `spec` takes backend-specific configuration parameters. In this example, the backend uses the `ontap-san` storage driver and uses the configuration parameters that are tabulated here. For the list of configuration options for your desired storage driver, refer to the backend configuration information for your storage driver.

The `spec` section also includes `credentials` and `deletionPolicy` fields, which are newly introduced in the `TridentBackendConfig` CR:

- `credentials`: This parameter is a required field and contains the credentials used to authenticate with the storage system/service. This is set to a user-created Kubernetes Secret. The credentials cannot be passed in plain text and will result in an error.
- `deletionPolicy`: This field defines what should happen when the `TridentBackendConfig` is deleted. It can take one of two possible values:
  - `delete`: This results in the deletion of both `TridentBackendConfig` CR and the associated backend. This is the default value.
  - `retain`: When a `TridentBackendConfig` CR is deleted, the backend definition will still be present and can be managed with `tridentctl`. Setting the deletion policy to `retain` lets users downgrade to an earlier release (pre-21.04) and retain the created backends. The value for this field can be updated after a `TridentBackendConfig` is created.

The name of a backend is set using `spec.backendName`. If unspecified, the name of the backend is set to the name of the `TridentBackendConfig` object (metadata.name). It is recommended to explicitly set backend names using `spec.backendName`.

Backends that were created with `tridentctl` do not have an associated `TridentBackendConfig` object. You can choose to manage such backends with `kubectl` by creating a `TridentBackendConfig` CR. Care must be taken to specify identical config parameters (such as `spec.backendName`, `spec.storagePrefix`, `spec.storageDriverName`, and so on). Trident will automatically bind the newly-created `TridentBackendConfig` with the pre-existing backend.

Steps overview

To create a new backend by using `kubectl`, you should do the following:

1. Create a Kubernetes Secret. The secret contains the credentials Trident needs to communicate with the storage cluster/service.
2. Create a `TridentBackendConfig` object. This contains specifics about the storage cluster/service and references the secret created in the previous step.

After you create a backend, you can observe its status by using `kubectl get tbc <tbc-name> -n <trident-namespace>` and gather additional details.

Step 1: Create a Kubernetes Secret

Create a Secret that contains the access credentials for the backend. This is unique to each storage service/platform. Here's an example:

```
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
```

This table summarizes the fields that must be included in the Secret for each storage platform:

Storage platform Secret Fields description	Secret	Fields description
Azure NetApp Files	clientID	The client ID from an app registration

Storage platform Secret Fields description	Secret	Fields description
Element (NetApp HCI/SolidFire)	Endpoint	MVIP for the SolidFire cluster with tenant credentials
ONTAP	username	Username to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	password	Password to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	clientPrivateKey	Base64-encoded value of client private key. Used for certificate-based authentication
ONTAP	chapUsername	Inbound username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetUsername	Target username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP target initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`

The Secret created in this step will be referenced in the `spec.credentials` field of the `TridentBackendConfig` object that is created in the next step.

Step 2: Create the `TridentBackendConfig` CR

You are now ready to create your `TridentBackendConfig` CR. In this example, a backend that uses the `ontap-san` driver is created by using the `TridentBackendConfig` object shown below:

```
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
```

Step 3: Verify the status of the `TridentBackendConfig` CR

Now that you created the `TridentBackendConfig` CR, you can verify the status. See the following example:

```
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
```

A backend was successfully created and bound to the `TridentBackendConfig` CR.

Phase can take one of the following values:

- `Bound`: The `TridentBackendConfig` CR is associated with a backend, and that backend contains `configRef` set to the `TridentBackendConfig` CR's uid.
- `Unbound`: Represented using `""`. The `TridentBackendConfig` object is not bound to a backend. All newly created `TridentBackendConfig` CRs are in this phase by default. After the phase changes, it cannot revert to Unbound again.
- `Deleting`: The `TridentBackendConfig` CR's `deletionPolicy` was set to delete. When the `TridentBackendConfig` CR is deleted, it transitions to the Deleting state.
  - If no persistent volume claims (PVCs) exist on the backend, deleting the `TridentBackendConfig` will result in Trident deleting the backend as well as the `TridentBackendConfig` CR.
  - If one or more PVCs are present on the backend, it goes to a deleting state. The `TridentBackendConfig` CR subsequently also enters deleting phase. The backend and `TridentBackendConfig` are deleted only after all PVCs are deleted.
- `Lost`: The backend associated with the `TridentBackendConfig` CR was accidentally or deliberately deleted and the `TridentBackendConfig` CR still has a reference to the deleted backend. The `TridentBackendConfig` CR can still be deleted irrespective of the `deletionPolicy` value.

- `Unknown`: Trident is unable to determine the state or existence of the backend associated with the `TridentBackendConfig` CR. For example, if the API server is not responding or if the `tridentbackends.trident.netapp.io` CRD is missing. This might require intervention.

At this stage, a backend is successfully created! There are several operations that can additionally be handled, such as backend updates and backend deletions.

(Optional) Step 4: Get more details

You can run the following command to get more information about your backend:

```
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
```

In addition, you can also obtain a YAML/JSON dump of `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` contains the `backendName` and the `backendUUID` of the backend that got created in response to the `TridentBackendConfig` CR. The `lastOperationStatus` field represents the status of the last operation of the `TridentBackendConfig` CR, which can be user-triggered (for example, user changed something in `spec`) or triggered by Trident (for example, during Trident restarts). It can either be Success or Failed. `phase` represents the status of the relation between the `TridentBackendConfig` CR and the backend. In the example above, `phase` has the value Bound, which means that the `TridentBackendConfig` CR is associated with the backend.

You can run the `kubectl -n trident describe tbc <tbc-cr-name>` command to get details of the event logs.

You cannot update or delete a backend which contains an associated `TridentBackendConfig` object using `tridentctl`. To understand the steps involved in switching between `tridentctl` and `TridentBackendConfig`, see here.

Manage backends

Perform backend management with kubectl

Learn about how to perform backend management operations by using `kubectl`.

Delete a backend

By deleting a `TridentBackendConfig`, you instruct Trident to delete/retain backends (based on `deletionPolicy`). To delete a backend, ensure that `deletionPolicy` is set to delete. To delete just the `TridentBackendConfig`, ensure that `deletionPolicy` is set to retain. This ensures the backend is still present and can be managed by using `tridentctl`.

Run the following command:

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

Trident does not delete the Kubernetes Secrets that were in use by `TridentBackendConfig`. The Kubernetes user is responsible for cleaning up secrets. Care must be taken when deleting secrets. You should delete secrets only if they are not in use by the backends.

View the existing backends

Run the following command:

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

You can also run `tridentctl get backend -n trident` or `tridentctl get backend -o yaml -n trident` to obtain a list of all backends that exist. This list will also include backends that were created with `tridentctl`.

Update a backend

There can be multiple reasons to update a backend:

- Credentials to the storage system have changed. To update credentials, the Kubernetes Secret that is used in the `TridentBackendConfig` object must be updated. Trident will automatically update the backend with the latest credentials provided. Run the following command to update the Kubernetes Secret:

  ```
  kubectl apply -f <updated-secret-file.yaml> -n trident
  ```

- Parameters (such as the name of the ONTAP SVM being used) need to be updated.
  - You can update `TridentBackendConfig` objects directly through Kubernetes using the following command:

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

- Alternatively, you can make changes to the existing `TridentBackendConfig` CR using the following command:

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

- If a backend update fails, the backend continues to remain in its last known configuration. You can view the logs to determine the cause by running `kubectl get tbc <tbc-name> -o yaml -n trident` or `kubectl describe tbc <tbc-name> -n trident`.
- After you identify and correct the problem with the configuration file, you can re-run the update command.

Perform backend management with tridentctl

Learn about how to perform backend management operations by using `tridentctl`.

Create a backend

After you create a backend configuration file, run the following command:

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

If backend creation fails, something was wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs -n trident
```

After you identify and correct the problem with the configuration file, you can simply run the `create` command again.

Delete a backend

To delete a backend from Trident, do the following:

1. Retrieve the backend name:

```
tridentctl get backend -n trident
```

2. Delete the backend:

```
tridentctl delete backend <backend-name> -n trident
```

If Trident has provisioned volumes and snapshots from this backend that still exist, deleting the backend prevents new volumes from being provisioned by it. The backend will continue to exist in a "Deleting" state.

View the existing backends

To view the backends that Trident knows about, do the following:

- To get a summary, run the following command:

```
tridentctl get backend -n trident
```

- To get all the details, run the following command:

```
tridentctl get backend -o json -n trident
```

Update a backend

After you create a new backend configuration file, run the following command:

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

If backend update fails, something was wrong with the backend configuration or you attempted an invalid update. You can view the logs to determine the cause by running the following command:

```
tridentctl logs -n trident
```

After you identify and correct the problem with the configuration file, you can simply run the `update` command again.

Identify the storage classes that use a backend

This is an example of the kind of questions you can answer with the JSON that `tridentctl` outputs for backend objects. This uses the `jq` utility, which you need to install.

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

This also applies for backends that were created by using `TridentBackendConfig`.

Move between backend management options

Learn about the different ways of managing backends in Trident.

Options for managing backends

With the introduction of `TridentBackendConfig`, administrators now have two unique ways of managing backends. This poses the following questions:

- Can backends created using `tridentctl` be managed with `TridentBackendConfig`?
- Can backends created using `TridentBackendConfig` be managed using `tridentctl`?

Manage `tridentctl` backends using `TridentBackendConfig`

This section covers the steps required to manage backends that were created using `tridentctl` directly through the Kubernetes interface by creating `TridentBackendConfig` objects.

This will apply to the following scenarios:

- Pre-existing backends, that don't have a `TridentBackendConfig` because they were created with `tridentctl`.
- New backends that were created with `tridentctl`, while other `TridentBackendConfig` objects exist.

In both scenarios, backends will continue to be present, with Trident scheduling volumes and operating on them. Administrators have one of two choices here:

- Continue using `tridentctl` to manage backends that were created using it.
- Bind backends created using `tridentctl` to a new `TridentBackendConfig` object. Doing so would mean the backends will be managed using `kubectl` and not `tridentctl`.

To manage a pre-existing backend using `kubectl`, you will need to create a `TridentBackendConfig` that binds to the existing backend. Here is an overview of how that works:

1. Create a Kubernetes Secret. The secret contains the credentials Trident needs to communicate with the storage cluster/service.
2. Create a `TridentBackendConfig` object. This contains specifics about the storage cluster/service and references the secret created in the previous step. Care must be taken to specify identical config parameters (such as `spec.backendName`, `spec.storagePrefix`, `spec.storageDriverName`, and so on). `spec.backendName` must be set to the name of the existing backend.

Step 0: Identify the backend

To create a `TridentBackendConfig` that binds to an existing backend, you will need to obtain the backend configuration. In this example, let us assume a backend was created using the following JSON definition:

```
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"
      }
    }
  ]
}
```

Step 1: Create a Kubernetes Secret

Create a Secret that contains the credentials for the backend, as shown in this example:

```
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
```

Step 2: Create a `TridentBackendConfig` CR

The next step is to create a `TridentBackendConfig` CR that will automatically bind to the pre-existing `ontap-nas-backend` (as in this example). Ensure the following requirements are met:

- The same backend name is defined in `spec.backendName`.
- Configuration parameters are identical to the original backend.
- Virtual pools (if present) must retain the same order as in the original backend.
- Credentials are provided through a Kubernetes Secret and not in plain text.

In this case, the `TridentBackendConfig` will look like this:

```
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
```

Step 3: Verify the status of the `TridentBackendConfig` CR

After the `TridentBackendConfig` has been created, its phase must be `Bound`. It should also reflect the same backend name and UUID as that of the existing backend.

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                   BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

The backend will now be completely managed using the `tbc-ontap-nas-backend TridentBackendConfig` object.

Manage `TridentBackendConfig` backends using `tridentctl`

`tridentctl` can be used to list backends that were created using `TridentBackendConfig`. In addition, administrators can also choose to completely manage such backends through `tridentctl` by deleting `TridentBackendConfig` and making sure `spec.deletionPolicy` is set to `retain`.

Step 0: Identify the backend

For example, let us assume the following backend was created using `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 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

From the output, it is seen that `TridentBackendConfig` was created successfully and is bound to a backend [observe the backend's UUID].

Step 1: Confirm `deletionPolicy` is set to `retain`

Let us take a look at the value of `deletionPolicy`. This needs to be set to `retain`. This ensures that when a `TridentBackendConfig` CR is deleted, the backend definition will still be present and can be managed with `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
```

Do not proceed to the next step unless `deletionPolicy` is set to `retain`.

Step 2: Delete the `TridentBackendConfig` CR

The final step is to delete the `TridentBackendConfig` CR. After confirming the `deletionPolicy` is set to `retain`, you can go ahead with the deletion:

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

Upon the deletion of the `TridentBackendConfig` object, Trident simply removes it without actually deleting the backend itself.

Create and manage storage classes

Create a storage class

Configure a Kubernetes StorageClass object and create the storage class to instruct Trident how to provision volumes.

Configure a Kubernetes StorageClass object

The Kubernetes StorageClass object identifies Trident as the provisioner that is used for that class and instructs Trident how to provision a volume. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Create a storage class

After you create the StorageClass object, you can create the storage class. Storage class samples provides some basic samples you can use or modify.

Steps

1. This is a Kubernetes object, so use `kubectl` to create it in Kubernetes.

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

2. You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
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"
        ]
      }
    }
  ]
}
```


Storage class samples

Trident provides simple storage class definitions for specific backends.

Alternatively, you can edit `sample-input/storage-class-csi.yaml.templ` file that comes with the installer and replace *`BACKEND_TYPE`* with the storage driver name.

```
./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
```

Manage storage classes

You can view existing storage classes, set a default storage class, identify the storage class backend, and delete storage classes.

View the existing storage classes

- To view existing Kubernetes storage classes, run the following command:

```
kubectl get storageclass
```

- To view Kubernetes storage class detail, run the following command:

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

- To view Trident's synchronized storage classes, run the following command:

```
tridentctl get storageclass
```

- To view Trident's synchronized storage class detail, run the following command:

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

Set a default storage class

Kubernetes 1.6 added the ability to set a default storage class. This is the storage class that will be used to provision a Persistent Volume if a user does not specify one in a Persistent Volume Claim (PVC).

- Define a default storage class by setting the annotation `storageclass.kubernetes.io/is-default-class` to true in the storage class definition. According to the specification, any other value or absence of the annotation is interpreted as false.
- You can configure an existing storage class to be the default storage class by using the following command:

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- Similarly, you can remove the default storage class annotation by using the following command:

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

There are also examples in the Trident installer bundle that include this annotation.

There should be only one default storage class in your cluster at a time. Kubernetes does not technically prevent you from having more than one, but it will behave as if there is no default storage class at all.

Identify the backend for a storage class

This is an example of the kind of questions you can answer with the JSON that `tridentctl` outputs for Trident backend objects. This uses the `jq` utility, which you may need to install first.

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

Delete a storage class

To delete a storage class from Kubernetes, run the following command:

```
kubectl delete storageclass <storage-class>
```

`<storage-class>` should be replaced with your storage class.

Any persistent volumes that were created through this storage class will remain untouched, and Trident will continue to manage them.

Trident enforces a blank `fsType` for the volumes it creates. For iSCSI backends, it is recommended to enforce `parameters.fsType` in the StorageClass. You should delete existing StorageClasses and re-create them with `parameters.fsType` specified.

Provision and manage volumes

Provision a volume

Create a PersistentVolumeClaim (PVC) that uses the configured Kubernetes StorageClass to request access to the PV. You can then mount the PV to a pod.

Overview

A *PersistentVolumeClaim* (PVC) is a request for access to the PersistentVolume on the cluster.

The PVC can be configured to request storage of a certain size or access mode. Using the associated StorageClass, the cluster administrator can control more than PersistentVolume size and access mode—such as performance or service level.

After you create the PVC you can mount the volume in a pod.

Create the PVC

Steps

1. Create the PVC.

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

2. Verify the PVC status.

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS AGE
pvc-storage   Bound    pv-name    1Gi        RWO                    5m
```

1. Mount the volume in a pod.

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

You can monitor the progress using `kubectl get pod --watch`.

2. Verify that the volume is mounted on `/my/mount/path`.

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

3. You can now delete the Pod. The Pod application will no longer exist, but the volume will remain.

```
kubectl delete pod pv-pod
```

Sample manifests

PersistentVolumeClaim sample manifests

These examples show basic PVC configuration options.

PVC with RWO access

This example shows a basic PVC with RWO access that is associated with a StorageClass named `basic-csi`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

PVC with NVMe/TCP

This example shows a basic PVC for NVMe/TCP with RWO access that is associated with a StorageClass named `protection-gold`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Pod manifest samples

These examples show basic configurations to attach the PVC to a pod.

Basic configuration

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

Basic NVMe/TCP configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Expand volumes

Trident provides Kubernetes users the ability to expand their volumes after they are created. Find information about the configurations required to expand iSCSI, NFS, SMB, NVMe/TCP, and FC volumes.

Expand an iSCSI volume

You can expand an iSCSI Persistent Volume (PV) by using the CSI provisioner.

> iSCSI volume expansion is supported by the `ontap-san`, `ontap-san-economy`, `solidfire-san` drivers and requires Kubernetes 1.16 and later.

Step 1: Configure the StorageClass to support volume expansion

Edit the StorageClass definition to set the `allowVolumeExpansion` field to `true`.

```
cat storageclass-ontapsan.yaml
```

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

For an already existing StorageClass, edit it to include the `allowVolumeExpansion` parameter.

Step 2: Create a PVC with the StorageClass you created

Edit the PVC definition and update the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.

```
cat pvc-ontapsan.yaml
```

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

Trident creates a Persistent Volume (PV) and associates it with this Persistent Volume Claim (PVC).

```
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
```

Step 3: Define a pod that attaches the PVC

Attach the PV to a pod for it to be resized. There are two scenarios when resizing an iSCSI PV:

- If the PV is attached to a pod, Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Trident expands the volume on the storage backend. After the PVC is bound to a pod, Trident rescans the device and resizes the filesystem. Kubernetes then updates the PVC size after the expand operation has successfully completed.

In this example, a pod is created that uses the `san-pvc`.

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

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

Step 4: Expand the PV

To resize the PV that has been created from 1Gi to 2Gi, edit the PVC definition and update the `spec.resources.requests.storage` to 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
 # ...
```

Step 5: Validate the expansion

You can validate the expansion worked correctly by checking the size of the PVC, PV, and the Trident volume:

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

Expand an FC volume

You can expand an FC Persistent Volume (PV) by using the CSI provisioner.

> FC volume expansion is supported by the `ontap-san` driver and requires Kubernetes 1.16 and later.

Step 1: Configure the StorageClass to support volume expansion

Edit the StorageClass definition to set the `allowVolumeExpansion` field to `true`.

```
cat storageclass-ontapsan.yaml
```

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

For an already existing StorageClass, edit it to include the `allowVolumeExpansion` parameter.

Step 2: Create a PVC with the StorageClass you created

Edit the PVC definition and update the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.

```
cat pvc-ontapsan.yaml
```

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

Trident creates a Persistent Volume (PV) and associates it with this Persistent Volume Claim (PVC).

```
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
```

Step 3: Define a pod that attaches the PVC

Attach the PV to a pod for it to be resized. There are two scenarios when resizing an FC PV:

- If the PV is attached to a pod, Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Trident expands the volume on the storage backend. After the PVC is bound to a pod, Trident rescans the device and resizes the filesystem. Kubernetes then updates the PVC size after the expand operation has successfully completed.

In this example, a pod is created that uses the `san-pvc`.

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

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

Step 4: Expand the PV

To resize the PV that has been created from 1Gi to 2Gi, edit the PVC definition and update the `spec.resources.requests.storage` to 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
 # ...
```

Step 5: Validate the expansion

You can validate the expansion worked correctly by checking the size of the PVC, PV, and the Trident volume:

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

Expand an NFS volume

Trident supports volume expansion for NFS PVs provisioned on `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, and `azure-netapp-files` backends.

Step 1: Configure the StorageClass to support volume expansion

To resize an NFS PV, the admin first needs to configure the storage class to allow volume expansion by setting the `allowVolumeExpansion` field to `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
```

If you have already created a storage class without this option, you can simply edit the existing storage class

by using `kubectl edit storageclass` to allow volume expansion.

Step 2: Create a PVC with the StorageClass you created

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident should create a 20 MiB NFS PV for this PVC:

```
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
```

Step 3: Expand the PV

To resize the newly created 20 MiB PV to 1 GiB, edit the PVC and set `spec.resources.requests.storage` to 1 GiB:

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

Step 4: Validate the expansion

You can validate the resize worked correctly by checking the size of the PVC, PV, and the Trident volume:

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

Understand RWX NVMe subsystem limits

ReadWriteMany (RWX) volumes that use the NVMe protocol have a scalability limit of 64 nodes per volume. The following includes the limitations, explains the NVMe subsystem architecture involved, and outlines the required resolution steps.

Understand the 64-node limit

If you plan to use ReadWriteMany (RWX) volumes with the NVMe protocol, a single RWX NVMe volume cannot be mounted by more than 64 nodes in a Kubernetes cluster.

Do not schedule workloads that mount the same RWX NVMe PersistentVolumeClaim across more than 64 nodes.

This limitation applies only to RWX volumes that use the NVMe protocol.

Understand NVMe subsystem models

Per-volume subsystem model (Trident releases earlier than 26.02)

In Trident releases earlier than 26.02, RWX NVMe volumes are provisioned using a per-volume subsystem

model.
Each RWX NVMe volume is mapped to its own dedicated NVMe subsystem on ONTAP.

This model is simple, but it has a lower scalability limit.
In large Kubernetes clusters, subsystem controller limits are reached quickly because each RWX volume consumes a dedicated subsystem.

Super-subsystem model (introduced in Trident 26.02)

Starting with Trident 26.02, RWX NVMe volumes use a shared super-subsystem model.
Multiple RWX NVMe volumes share the same NVMe subsystem.

Each super-subsystem supports up to 1024 namespaces (volumes).
This model significantly improves scalability for RWX workloads and reduces the likelihood of reaching ONTAP subsystem limits.

Each RWX NVMe volume supports up to 64 nodes.

Identify error symptoms

If you create or attach RWX NVMe volumes at scale, you might observe errors similar to the following:

```
Maximum number of controllers reached. No more controllers can be created.
```

This error indicates that the ONTAP NVMe subsystem controller limit has been reached.

Resolve subsystem limit errors

To move beyond per-volume subsystem limitations and take advantage of the super-subsystem model, upgrade to Trident 26.02 or later.

Upgrade Trident to apply the super-subsystem model

To apply the super-subsystem model for RWX NVMe volumes:

1. Upgrade Trident to version 26.02 or later.
2. Scale down all pods that use RWX NVMe volumes to zero replicas.
3. Verify that no workloads are actively using RWX NVMe volumes.
4. Scale the pods back up.

This restart sequence ensures that RWX NVMe volumes are attached using the super-subsystem model.

- This limitation applies only to RWX volumes that use the NVMe protocol.
- The 64-node limit applies per RWX NVMe volume.
- Other access modes and other protocols are not affected.

Controller scalability

Trident introduces controller scalability through improved concurrency across multiple storage drivers. Customers can identify which Trident drivers support controller scalability

at general availability and which drivers are available as a technical preview in Trident 26.02. This enables informed deployment decisions and appropriate risk management for scalable Kubernetes environments.

Key concepts and definitions

Controller scalability

Controller scalability refers to the Trident controller's ability to process multiple storage operations in parallel rather than serializing them behind a single lock.
These operations include volume creation, deletion, resizing, snapshot creation and deletion, volume publish and unpublish, and backend management.

When controller scalability is enabled, operations on different volumes and backends proceed concurrently. This increases throughput and reduces end-to-end operation time in environments with high numbers of concurrent PersistentVolumeClaim and VolumeSnapshot operations.

Controller scalability support

Trident supports controller scalability with different maturity levels depending on the storage driver.

General availability

The following drivers support controller scalability at general availability in Trident 26.02:

- `ontap-san`
- `ontap-nas`
- `google-cloud-netapp-volumes`

> The `google-cloud-netapp-volumes` and `google-cloud-netapp-volumes-san` drivers are different.
> Only `google-cloud-netapp-volumes` is supported. Do not use `google-cloud-netapp-volumes-san` in backend configurations or examples.

Enable controller scalability

Controller scalability is controlled by the `enableConcurrency` configuration option. This option must be explicitly enabled during Trident installation or by updating an existing deployment.

Trident operator deployment

To enable controller scalability with the Trident operator, set `enableConcurrency` to `true` in the `TridentOrchestrator` custom resource.

New installation

Create or edit the `TridentOrchestrator` CR with `enableConcurrency` set to `true`:

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

Apply the CR:

```
kubectl apply -f tridentorchestrator_cr.yaml
```

Existing installation

Patch the existing `TridentOrchestrator` CR to enable controller scalability:

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

Verify the setting was applied:

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm deployment

To enable controller scalability with Helm, set the `enableConcurrency` value to `true`.

New installation

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

Existing installation

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

Alternatively, set `enableConcurrency` to `true` in a custom `values.yaml` file:

```
# values.yaml
enableConcurrency: true
```

Then install or upgrade using the values file:

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl deployment

To enable controller scalability with `tridentctl`, pass the `--enable-concurrency` flag during installation.

New installation

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

Existing installation

To enable controller scalability on an existing `tridentctl`-based deployment, uninstall and reinstall with the flag:

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

Verify controller scalability is enabled

After enabling controller scalability, verify that the Trident controller is running with concurrency enabled by checking the controller pod logs:

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

You should see a log entry indicating that concurrency is enabled.

Technical preview

The following drivers support controller scalability as a technical preview in Trident 26.02:

- `nas-eco`
- `san-eco`

For these drivers:

- Controller concurrency is available for evaluation and testing

- Behavior may change in future releases
- Use in production environments is not recommended

Concurrency behavior

When controller scalability is enabled:

- Trident replaces the single global lock with fine-grained, per-resource locking
- Operations that modify the same resource are serialized to maintain data consistency
- Operations that only read from a resource can proceed concurrently with other read operations on that resource
- Trident limits concurrent ONTAP API requests to 20 per management LIF to prevent overloading backend storage systems
- If multiple backends share the same management LIF, they share this 20-request limit

Known limitations and considerations

The following considerations apply to controller scalability:

- Concurrency is managed internally by the Trident controller
- There are no user-configurable concurrency limits in this release
- Overall throughput depends on:
  - The storage driver in use
  - Backend responsiveness
  - Kubernetes API server performance
- High concurrency can increase load on backend storage systems

Caveats and limitations

The following limitations apply in Trident 26.02:

- Controller scalability behavior is not identical across all drivers
- Technical preview drivers may exhibit:
  - Inconsistent performance under high load
  - Changes in behavior between releases
- Debugging concurrent operations may be more complex due to parallel execution
- Metrics and logs may show interleaved operation output

Recommendations

- Use general availability (GA) drivers for production environments that require high scalability
- Evaluate technical preview drivers in non-production environments
- Monitor backend and controller performance when operating at scale
- Avoid assuming operation ordering in automation scripts

Automatic volume expansion

Automatic volume expansion enables Persistent Volumes provisioned by Trident to grow automatically when used capacity reaches a defined threshold. This capability reduces operational overhead and helps prevent application disruption caused by capacity exhaustion. Automatic volume expansion is implemented using Autogrow Policies. An Autogrow Policy defines:

- The utilization threshold that triggers expansion
- The amount by which the volume grows
- The maximum size the volume can reach

> Volumes increase in size automatically when the defined utilization threshold is exceeded. Volumes are never automatically reduced.

Requirements

Before configuring automatic volume expansion, ensure that the following requirements are met:

- Trident 26.02 or later
- Role-based access control permissions to create `TridentAutogrowPolicy` custom resources
- StorageClasses configured with `allowVolumeExpansion: true`
- Supported ONTAP protocols:
  - Network File System (NFS)
  - Internet Small Computer Systems Interface (iSCSI)
  - Non-Volatile Memory Express (NVMe)
  - Fibre Channel Protocol (FCP)

Limitations

- ONTAP Non-Volatile Memory Express raw block volumes earlier than ONTAP 9.16.1 do not support automatic expansion.
- For storage area network volumes, if `growthAmount` is less than or equal to 50 mebibytes, Trident automatically increases the value to 51 mebibytes before resizing, provided the resulting size does not exceed `maxSize`.
- In brownfield environments, automatic expansion might not function for certain existing volumes due to volume publication migration behavior.
- When a volume reaches `maxSize`, no further expansion occurs.
- Supported protocols for automatic volume expansion:
  - Network File System (NFS)
  - Internet Small Computer Systems Interface (iSCSI)
  - Non-Volatile Memory Express (NVMe)
  - Fibre Channel Protocol (FCP)

Provision Volumes with Autogrow Policy

Autogrow Policy can be configured at two levels:

- Storage class level: Sets default for all volumes (using annotation)
- PVC level: Overrides storage class default (using annotation)

Create an Autogrow Policy

Autogrow Policies enable automatic volume expansion when volumes reach a defined capacity threshold.

Ensure you have:

- Trident 26.02 or later installed
- Role-based access control permissions to create `TridentAutogrowPolicy` resources
- Understanding of workload growth requirements

An Autogrow Policy defines how volumes expand automatically when they reach a defined capacity threshold.

You can create Autogrow Policies at any point in your workflow:

- Before StorageClasses and volumes are created
- After StorageClasses exist
- After volumes are provisioned

This flexibility allows you to introduce automatic expansion without recreating existing resources.

Autogrow Policy specifications

Autogrow Policies are Kubernetes custom resources defined as follows:

Field	Description	Format	Required	Example	Default
name	Unique policy identifier	String	Yes	production-db-policy	None
usedThreshold	Capacity percentage that triggers expansion	Percentage string	Yes	"80%"	None
growthAmount	Amount to grow when threshold is reached	Percentage or size	No	"10%" or "5Gi"	"10%"
maxSize	Maximum volume size limit	Kubernetes quantity	No	"500Gi"	Unlimited

Create an Autogrow Policy

Steps

1. Create a YAML file that defines your Autogrow Policy:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. Apply the policy to your cluster:

```
kubectl apply -f autogrow-policy.yaml
```

3. Verify that the policy was created:

```
kubectl get tridentautogrowpolicy standard-autogrow
```

Expected output

```
NAME                USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow   80%              10%             Success
```

Policy states

After you create a policy, Trident validates the specification and assigns one of the following states:

State	Description	Action required
Success	Policy is validated and ready to use.	None.
Failed	Validation errors detected.	Review and fix the specification.
Deleting	Deletion is in progress.	Wait for completion.

Associate a policy with a StorageClass

You can associate an Autogrow Policy with a StorageClass by using the `trident.netapp.io/autogrowPolicy` annotation. All volumes provisioned from that StorageClass inherit the policy.

The StorageClass must have `allowVolumeExpansion: true`.

Steps

1. Create or modify a StorageClass with the Autogrow Policy annotation:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. Apply the StorageClass:

```
kubectl apply -f storageclass.yaml
```

3. Verify the annotation:

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Expected output

```
production-db-policy
```

Policy precedence

When Autogrow Policy annotations are set on both a StorageClass and a PVC, Trident applies the following precedence rules:

1. **PVC annotation takes priority.** If a PVC sets `trident.netapp.io/autogrowPolicy`, that value is always used.
2. **StorageClass annotation applies only when the PVC has no annotation.**
3. **If neither has the annotation,** no Autogrow Policy is applied.

Table 1. Precedence examples

StorageClass annotation	PVC annotation	Effective behavior
`trident.netapp.io/autogrowPolicy: standard-agp`	Not set	Uses `standard-agp`.

StorageClass annotation	PVC annotation	Effective behavior
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	Uses `logs-policy` (PVC overrides StorageClass).
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	No Autogrow Policy (PVC disables autogrow).
Not set	`trident.netapp.io/autogrowPolicy: dev-policy`	Uses `dev-policy`.
Not set	Not set	No Autogrow Policy.

Configuration examples

The following examples show common Autogrow Policy configurations for different use cases.

Conservative policy for production databases

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

Log storage with fixed growth increments

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

Minimal policy with defaults

When you omit `growthAmount` and `maxSize`, Trident uses the defaults (`10%` growth, unlimited size):

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

Policy with a custom maxSize and default growthAmount

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

Aggressive growth with unlimited maxSize

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

Policy with fractional percentages

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

Fractional percentages are supported. If you specify more than three decimal places, Trident rounds the value to three decimal places.

NAS StorageClass with Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass with Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

Manage Autogrow Policies

After you create Autogrow Policies, you can view, update, and delete them as needed. You can also monitor which volumes are using a given policy.

View Autogrow Policies

List all policies

Use `kubectl` to list all Autogrow Policies in your cluster:

```
kubectl get tridentautogrowpolicy
```

Alternatively, use `tridentctl`:

```
tridentctl get autogrowpolicy
```

View policy details

To view the full specification and status of a policy:

```
kubectl describe tridentautogrowpolicy production-db-policy
```

To view a policy with its associated volumes in YAML format:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

Update an Autogrow Policy

You can modify an existing policy to change its threshold, growth amount, or maximum size. Changes take effect immediately for all volumes that use the policy.

> Changes affect all volumes currently using the policy. Test changes in a non-production environment first when possible.

Steps

1. Edit the policy:

   ```
   kubectl edit tridentautogrowpolicy production-db-policy
   ```

2. Modify the `spec` fields as needed:

   ```
   spec:
     usedThreshold: "75%"     # Changed from 80%
     growthAmount: "20%"      # Changed from 10%
     maxSize: "1Ti"           # Changed from 500Gi
   ```

3. Save and exit. The changes take effect immediately.

Update considerations

- **Immediate effect:** All volumes using the policy adopt new parameters at the next growth evaluation.
- **No volume restart needed:** Changes apply to the next growth operation.
- **Test first:** Validate changes in a non-production environment when possible.
- **Communicate changes:** Notify teams when you modify shared policies.

Delete an Autogrow Policy

Autogrow Policies use finalizer protection to prevent accidental deletion while volumes are actively using them.

Steps

1. Delete the policy:

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. If volumes are still using the policy, the deletion enters a `Deleting` state. Check which volumes are affected:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. Remove the policy from each affected volume. Choose one of the following options:

   - **Option A: Explicitly disable autogrow** by setting the annotation to `"none"`:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

   - **Option B: Remove the annotation entirely:**

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

Deletion behavior

Scenario	Behavior
No volumes use the policy	Policy is deleted immediately.
Volumes are using the policy	Policy enters `Deleting` state. A finalizer blocks completion until all volumes are removed.
All volumes are removed from the policy	Finalizers are removed and the policy is deleted.

Monitor Autogrow Policy usage

Check volumes using a policy

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

Find which policy a volume uses

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Monitor policy events

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

Supported protocols

Autogrow supports the following storage protocols:

- NFS
- iSCSI
- FCP
- NVMe

> For SAN volumes, if the configured `growthAmount` is 50 MiB or less, Trident automatically increases the growth amount to 51 MB for the resize operation, as long as the resulting size does not exceed `maxSize`.

Known limitations

- **ONTAP NVMe raw block volumes:** Volumes created with ONTAP versions earlier than 9.16.1 do not support autogrow.
- **Existing volumes (brownfield deployments):** Autogrow might not work for existing volumes even if a valid Autogrow Policy is applied. This is due to an ongoing migration of volume publications. To confirm migration has completed, check the Trident controller logs for "`Migration completed`" messages.

Frequently asked questions

When does Trident evaluate the threshold?

Trident continuously monitors volume usage. When the used capacity crosses the `usedThreshold`, Trident creates an internal resize request and expands the volume by the configured `growthAmount`.

For example, this policy triggers expansion at 80% capacity and grows the volume by 10% each time, up to a maximum of 500 GiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

Can I apply a policy after volumes are already provisioned?

Yes. You can create an Autogrow Policy at any time and apply it to existing PVCs by adding or updating the `trident.netapp.io/autogrowPolicy` annotation. You do not need to recreate the PVC or the StorageClass.

To apply a policy to an existing PVC:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

To apply a policy to an existing StorageClass:

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

What happens if I set an Autogrow Policy on both the StorageClass and the PVC?

The PVC annotation always takes precedence. If a PVC has the `trident.netapp.io/autogrowPolicy` annotation, Trident uses that value regardless of what the StorageClass specifies. Refer to Policy precedence for details.

For example, given this StorageClass:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

And this PVC that overrides the StorageClass policy:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Trident uses `logs-policy` for `database-pvc`, not `standard-agp`.

How do I disable autogrow for a specific volume?

Set the PVC annotation to `"none"`. This overrides any StorageClass-level policy for that volume:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

You can verify that autogrow is disabled:

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Expected output

```
none
```

What happens when a volume reaches maxSize?

Trident stops expanding the volume. No further resize requests are created for that volume, even if usage continues to increase beyond the `usedThreshold`.

For example, with this policy, Trident stops growing the volume once it reaches 100 GiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

To allow unlimited growth, omit `maxSize` or set it to `0`:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

Can I change a policy without restarting volumes?

Yes. When you update a policy, all volumes using that policy adopt the new parameters at the next growth evaluation. No volume restarts are required.

To update a policy in place:

```
kubectl edit tridentautogrowpolicy production-db-policy
```

Modify the fields as needed:

```
spec:
  usedThreshold: "75%"    # Changed from 80%
  growthAmount: "20%"     # Changed from 10%
  maxSize: "1Ti"          # Changed from 500Gi
```

Save and exit. Verify the updated policy:

```
kubectl get tridentautogrowpolicy production-db-policy
```

Expected output

```
NAME                    USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy    75%              20%             Success
```

Why is my policy in a Failed state?

A `Failed` state indicates that the policy specification contains validation errors. Run the following command to view the error details:

```
kubectl describe tridentautogrowpolicy <policy-name>
```

Common causes include an invalid `usedThreshold` (must be 1–99%), a `growthAmount` that exceeds `maxSize`, or an invalid Kubernetes quantity format. Correct the specification and reapply:

```
kubectl apply -f autogrow-policy.yaml
```

Why can't I delete a policy?

Policies use finalizer protection. If volumes are still using the policy, deletion enters a `Deleting` state and waits until all volumes are removed from the policy.

Identify the affected volumes:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

Then remove the annotation from each PVC:

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

After all volumes are removed, the finalizer is released and the policy is deleted.

Does autogrow work with all ONTAP backends?

Autogrow supports NFS, iSCSI, FCP, and NVMe protocols. However, NVMe raw block volumes require ONTAP 9.16.1 or later.

Existing volumes in brownfield deployments might require volume publication migration to complete before

autogrow takes effect. Verify migration status by checking the Trident controller logs:

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
completed"
```

The following StorageClass examples show autogrow configured for NAS and SAN backends:

NAS backend

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN backend

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

What is the minimum growth amount for SAN volumes?

For SAN volumes, the effective minimum growth amount is 51 MB. If you configure a `growthAmount` of 50 MiB or less, Trident automatically increases the growth to 51 MB for the resize operation.

For example, this policy sets a `growthAmount` of `"40Mi"`, but Trident applies a 51 MB growth for any SAN volume that uses it:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

To avoid this automatic adjustment, set `growthAmount` to a value greater than 50 MiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

Import volumes

You can import existing storage volumes as a Kubernetes PV using `tridentctl import` or by creating a Persistent Volume Claim (PVC) with Trident import annotations.

Overview and considerations

You might import a volume into Trident to:

- Containerize an application and reuse its existing data set
- Use a clone of a data set for an ephemeral application
- Rebuild a failed Kubernetes cluster
- Migrate application data during disaster recovery

Considerations

Before importing a volume, review the following considerations.

- Trident can import RW (read-write) type ONTAP volumes only. DP (data protection) type volumes are SnapMirror destination volumes. You should break the mirror relationship before importing the volume into Trident.
- We suggest importing volumes without active connections. To import an actively-used volume, clone the volume and then perform the import.

This is especially important for block volumes as Kubernetes would be unaware of the previous connection and could easily attach an active volume to a pod. This can result in data corruption.

- Though `StorageClass` must be specified on a PVC, Trident does not use this parameter during import. Storage classes are used during volume creation to select from available pools based on storage characteristics. Because the volume already exists, no pool selection is required during import. Therefore, the import will not fail even if the volume exists on a backend or pool that does not match the storage class specified in the PVC.
- The existing volume size is determined and set in the PVC. After the volume is imported by the storage driver, the PV is created with a ClaimRef to the PVC.
  - The reclaim policy is initially set to `retain` in the PV. After Kubernetes successfully binds the PVC and PV, the reclaim policy is updated to match the reclaim policy of the Storage Class.
  - If the reclaim policy of the Storage Class is `delete`, the storage volume will be deleted when the PV is deleted.
- By default, Trident manages the PVC and renames the FlexVol volume and LUN on the backend. You can pass the `--no-manage` flag to import an unmanaged volume and the `--no-rename` flag to retain the volume name.
  - **`--no-manage`** - If you use the `--no-manage` flag, Trident does not perform any additional operations on the PVC or PV for the lifecycle of the objects. The storage volume is not deleted when the PV is deleted and other operations such as volume clone and volume resize are also ignored.
  - **`--no-rename`** - If you use the `--no-rename` flag, Trident retains the existing volume name while importing volumes, and manages the lifecycle of the volumes. This option is supported only for the `ontap-nas`, `ontap-san` (including ASA r2 systems), and `ontap-san-economy` drivers.

    These options are useful if you want to use Kubernetes for containerized workloads but otherwise want to manage the lifecycle of the storage volume outside of Kubernetes.
- An annotation is added to the PVC and PV that serves a dual purpose of indicating that the volume was imported and if the PVC and PV are managed. This annotation should not be modified or removed.

Import a volume

You can import a volume using either `tridentctl import` or by creating a PVC with Trident import annotations.

If you use PVC annotations, you don't need to download or use `tridentctl` to import the volume.

Using tridentctl

Steps

1. Create a PVC file (for example, `pvc.yaml`) that will be used to create the PVC. The PVC file should include `name`, `namespace`, `accessModes`, and `storageClassName`. Optionally, you can specify `unixPermissions` in your PVC definition.

   The following is an example of a minimum specification:

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

   > Include only the required parameters. Additional parameters such as PV name or volume size can cause the import command to fail.

2. Use the `tridentctl import` command to specify the name of the Trident backend containing the volume and the name that uniquely identifies the volume on the storage (for example: ONTAP FlexVol, Element Volume). The `-f` argument is required to specify the path to the PVC file.

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

Using PVC annotations

Steps

1. Create a PVC YAML file (for example, `pvc.yaml`) with the required Trident import annotations. The PVC file should include:

   - `name` and `namespace` in metadata
   - `accessModes`, `resources.requests.storage`, and `storageClassName` in spec
   - Annotations:
     - `trident.netapp.io/importOriginalName`: Volume name on the backend
     - `trident.netapp.io/importBackendUUID`: Backend UUID where volume exists
     - `trident.netapp.io/notManaged` (*Optional*): Set to `"true"` for unmanaged volumes. Default is `"false"`.

       The following is an example specification for importing a managed volume:

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

2. Apply the PVC YAML file to your Kubernetes cluster:

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

Trident will automatically import the volume and bind it to the PVC.

Examples

Review the following volume import examples for supported drivers.

ONTAP NAS and ONTAP NAS FlexGroup

Trident supports volume import using the `ontap-nas` and `ontap-nas-flexgroup` drivers.

- Trident does not support volume import using the `ontap-nas-economy` driver.
- The `ontap-nas` and `ontap-nas-flexgroup` drivers do not allow duplicate volume names.

Each volume created with the `ontap-nas` driver is a FlexVol volume on the ONTAP cluster. Importing FlexVol volumes with the `ontap-nas` driver works the same. A FlexVol volumes that already exists on an ONTAP cluster can be imported as a `ontap-nas` PVC. Similarly, FlexGroup vols can be imported as `ontap-nas-flexgroup` PVCs.

ONTAP NAS examples using tridentctl

The following examples show how to import managed and unmanaged volumes using `tridentctl`.

Managed volume

The following example imports a volume named `managed_volume` on a backend named `ontap_nas`:

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

Unmanaged volume

When using the `--no-manage` argument, Trident does not rename the volume.

The following example imports `unmanaged_volume` on the `ontap_nas` backend:

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

ONTAP NAS examples using PVC annotations

The following examples show how to import managed and unmanaged volumes using PVC annotations.

Managed volume

The following example imports a 1GiB `ontap-nas` volume named `ontap_volume1` from backend `81abcb27-ea63-49bb-b606-0a5315ac5f21` with RWO access mode set using PVC annotations:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

Unmanaged volume

The following example imports 1Gi `ontap-nas` volume named `ontap-volume2` from backend `34abcb27-ea63-49bb-b606-0a5315ac5f34` with RWO access mode set using PVC annotations:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Trident supports volume import using the `ontap-san` (iSCSI, NVMe/TCP, and FC) and `ontap-san-economy` drivers.

Trident can import ONTAP SAN FlexVol volumes that contain a single LUN. This is consistent with the `ontap-san` driver, which creates a FlexVol volume for each PVC and a LUN within the FlexVol volume. Trident imports the FlexVol volume and associates it with the PVC definition. Trident can import `ontap-san-economy` volumes that contain multiple LUNs.

The following examples show how to import managed and unmanaged volumes:

Managed volume

For managed volumes, Trident renames the FlexVol volume to the `pvc-<uuid>` format and the LUN within the FlexVol volume to `lun0`.

The following example imports the `ontap-san-managed` FlexVol volume that is present on the `ontap_san_default` backend:

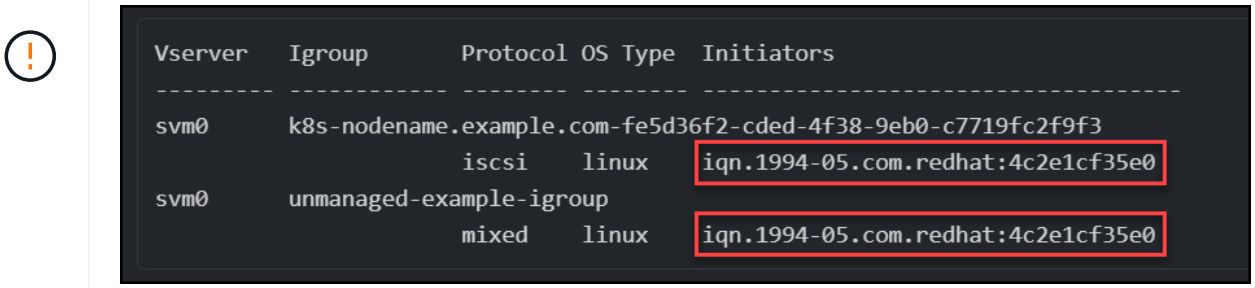
```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

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

Unmanaged volume

The following example imports `unmanaged_example_volume` on the `ontap_san` backend:

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

If you have LUNS mapped to igroups that share an IQN with a Kubernetes node IQN, as shown in the following example, you will receive the error: `LUN already mapped to initiator(s) in this group`. You will need to remove the initiator or unmap the LUN to import the volume.



```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economy examples

The following examples show how to import managed and unmanaged volumes for the `ontap-san-economy` backend.

Managed volume

When you import a managed volume, Trident takes ownership of the FlexVol and renames it. You must account for this renaming when you import multiple LUNs from the same FlexVol.

The following example imports `lun1` from the FlexVol `toimport` as a managed volume named `vol-managed-saneco`:

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

After importing `lun1`, Trident renames the FlexVol (for example, to `trident_lun_pool_xyz`). To import additional LUNs from the same FlexVol, use the new FlexVol name:

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

The `ontap-san-economy` backend imports one LUN at a time. You can automate multiple imports using a script.

Unmanaged volume

When you import an unmanaged volume, Trident does not take ownership of the FlexVol. However, the FlexVol and LUN must follow Trident naming conventions.

FlexVol naming format

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX` is the value of `storagePrefix` in your backend configuration. The default is `trident`.
- `RANDOMSTRING` is any string you choose.

LUN naming requirement

The LUN must be named `lun0`.

Example

If your `storagePrefix` is `xyz`, the full path to the LUN is:

```
trident_lun_pool_xyz_randomstring/lun0
```

Element

Trident supports NetApp Element software and NetApp HCI volume import using the `solidfire-san` driver.

The Element driver supports duplicate volume names. However, Trident returns an error if there are duplicate volume names. As a workaround, clone the volume, provide a unique volume name, and import the cloned volume.

The following example imports an `element-managed` volume on backend `element_default`.

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident supports volume import using the `azure-netapp-files` driver.

To import an Azure NetApp Files volume, identify the volume by its volume path. The volume path is the portion of the volume's export path after the `:/`. For example, if the mount path is `10.0.0.2:/importvol1`, the volume path is `importvol1`.

The following example imports an `azure-netapp-files` volume on backend `azurenetappfiles_40517` with the volume path `importvol1`.

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetApp Volumes

Trident supports volume import using the `google-cloud-netapp-volumes` driver.

The following example imports a volume on backend `backend-tbc-gcnv1` with the volume `testvoleasiaeast1`.

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

The following example imports a `google-cloud-netapp-volumes` volume when two volumes are present in the same region:

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

Customize volume names and labels

With Trident, you can assign meaningful names and labels to volumes you create. This helps you identify and easily map volumes to their respective Kubernetes resources (PVCs). You can also define templates at the backend level for creating custom volume names and custom labels; any volumes that you create, import, or clone will adhere to the templates.

Before you begin

Customizable volume names and labels support:

- Volume create, import, and clone operations.
- In the case of the `ontap-nas-economy` driver, only the name of the Qtree volume complies with the name template.
- In the case of the `ontap-san-economy` driver, only the LUN name complies with the name template.

Limitations

- Custom volume names are compatible with ONTAP on-premises drivers only.
- Custom labels are supported only for the `ontap-san`, `ontap-nas`, and `ontap-nas-flexgroup` drivers.
- Custom volume names do not apply to existing volumes.

Key behaviors of customizable volume names

- If a failure occurs due to invalid syntax in a name template, the backend creation fails. However, if the template application fails, the volume will be named according to existing naming convention.

- Storage prefix is not applicable when a volume is named using a name template from the backend configuration. Any desired prefix value may be directly added to the template.

Backend configuration examples with name template and labels

Custom name templates can be defined at the root and/or pool level.

Root level example

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

Pool level example

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

Name template examples

Example 1:

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

Example 2:

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

Points to consider

1. In the case of volume imports, the labels are updated only if the existing volume has labels in a specific format. For example: `{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`.
2. In the case of managed volume imports, the volume name follows the name template defined at the root level in the backend definition.
3. Trident does not support the use of a slice operator with the storage prefix.
4. If the templates do not result in unique volume names, Trident will append a few random characters to create unique volume names.
5. If the custom name for a NAS economy volume exceeds 64 characters in length, Trident will name the volumes according to the existing naming convention. For all other ONTAP drivers, if the volume name exceeds the name limit, the volume creation process fails.

Share an NFS volume across namespaces

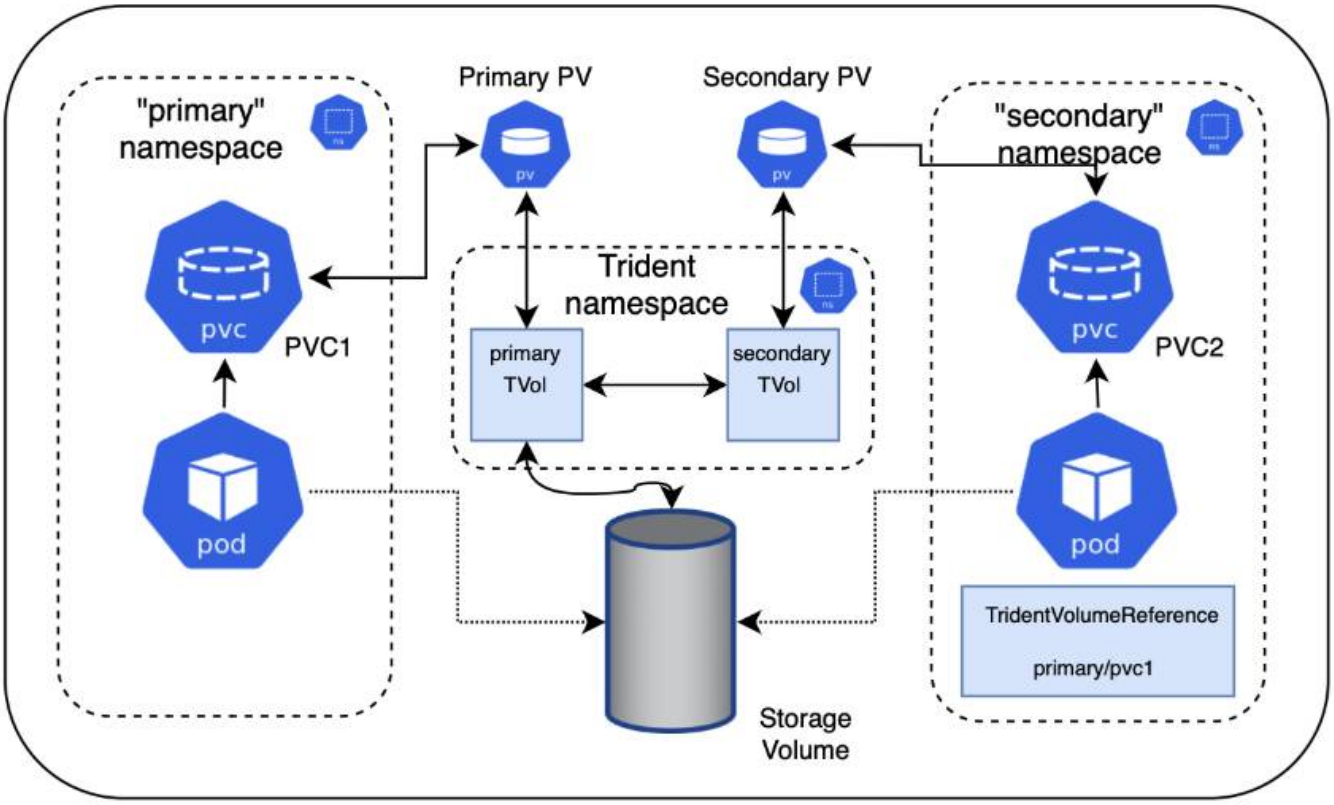
Using Trident, you can create a volume in a primary namespace and share it in one or more secondary namespaces.

Features

The TridentVolumeReference CR allows you to securely share ReadWriteMany (RWX) NFS volumes across one or more Kubernetes namespaces. This Kubernetes-native solution has the following benefits:

- Multiple levels of access control to ensure security
- Works with all Trident NFS volume drivers
- No reliance on tridentctl or any other non-native Kubernetes feature

This diagram illustrates NFS volume sharing across two Kubernetes namespaces.



Quick start

You can set up NFS volume sharing in just a few steps.

1 **Configure source PVC to share the volume**

The source namespace owner grants permission to access the data in the source PVC.

2 **Grant permission to create a CR in the destination namespace**

The cluster administrator grants permission to the owner of the destination namespace to create the TridentVolumeReference CR.

3 **Create TridentVolumeReference in the destination namespace**

The owner of the destination namespace creates the TridentVolumeReference CR to refer to the source PVC.

4 **Create the subordinate PVC in the destination namespace**

The owner of the destination namespace creates the subordinate PVC to use the data source from the source PVC.

Configure the source and destination namespaces

To ensure security, cross namespace sharing requires collaboration and action by the source namespace

owner, cluster administrator, and destination namespace owner. The user role is designated in each step.

Steps

1. **Source namespace owner:** Create the PVC (`pvc1`) in the source namespace that grants permission to share with the destination namespace (`namespace2`) using the `shareToNamespace` annotation.

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

Trident creates the PV and its backend NFS storage volume.

- You can share the PVC to multiple namespaces using a comma-delimited list. For example, `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`.
- You can share to all namespaces using `*`. For example, `trident.netapp.io/shareToNamespace: *`
- You can update the PVC to include the `shareToNamespace` annotation at any time.

2. **Cluster admin:** Ensure that proper RBAC is in place to grant permission to the destination namespace owner to create the TridentVolumeReference CR in the destination namespace.
3. **Destination namespace owner:** Create a TridentVolumeReference CR in the destination namespace that refers to the source namespace `pvc1`.

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

4. **Destination namespace owner:** Create a PVC (`pvc2`) in destination namespace (`namespace2`) using the `shareFromPVC` annotation to designate the source PVC.

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

The size of the destination PVC must be less than or equal than the source PVC.

Results

Trident reads the `shareFromPVC` annotation on the destination PVC and creates the destination PV as a subordinate volume with no storage resource of its own that points to the source PV and shares the source PV storage resource. The destination PVC and PV appear bound as normal.

Delete a shared volume

You can delete a volume that is shared across multiple namespaces. Trident will remove access to the volume on the source namespace and maintain access for other namespaces that share the volume. When all namespaces that reference the volume are removed, Trident deletes the volume.

Use `tridentctl get` to query subordinate volumes

Using the `tridentctl` utility, you can run the `get` command to get subordinate volumes. For more information, refer to `tridentctl` commands and options.

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

Flags:

- `-h, --help`: Help for volumes.
- `--parentOfSubordinate string`: Limit query to subordinate source volume.
- `--subordinateOf string`: Limit query to subordinates of volume.

Limitations

- Trident cannot prevent destination namespaces from writing to the shared volume. You should use file locking or other processes to prevent overwriting shared volume data.
- You cannot revoke access to the source PVC by removing the `shareToNamespace` or `shareFromNamespace` annotations or deleting the `TridentVolumeReference` CR. To revoke access, you must delete the subordinate PVC.
- Snapshots, clones, and mirroring are not possible on subordinate volumes.

For more information

To learn more about cross-namespace volume access:

- Watch the demo on NetAppTV.

Clone volumes across namespaces

Using Trident, you can create new volumes using existing volumes or volumesnapshots from a different namespace inside the same Kubernetes cluster.

Prerequisites

Before cloning volumes, ensure that the source and destination backends are of the same type and have the same storage class.

> Cloning across namespaces is supported only for the `ontap-san` and `ontap-nas` storage drivers. Read-only clones are not supported.

Quick start

You can set up volume cloning in just a few steps.

Configure source PVC to clone the volume

The source namespace owner grants permission to access the data in the source PVC.

Grant permission to create a CR in the destination namespace

The cluster administrator grants permission to the owner of the destination namespace to create the TridentVolumeReference CR.

Create TridentVolumeReference in the destination namespace

The owner of the destination namespace creates the TridentVolumeReference CR to refer to the source PVC.

Create the clone PVC in the destination namespace

The owner of the destination namespace creates PVC to clone the PVC from the source namespace.

Configure the source and destination namespaces

To ensure security, cloning volumes across namespaces requires collaboration and action by the source namespace owner, cluster administrator, and destination namespace owner. The user role is designated in each step.

Steps

1. **Source namespace owner:** Create the PVC (`pvc1`) in the source namespace (`namespace1`) that grants permission to share with the destination namespace (`namespace2`) using the `cloneToNamespace` annotation.

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

   Trident creates the PV and its backend storage volume.

   - You can share the PVC to multiple namespaces using a comma-delimited list. For example, `trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`.
   - You can share to all namespaces using `*`. For example, `trident.netapp.io/cloneToNamespace: *`
   - You can update the PVC to include the `cloneToNamespace` annotation at any time.

2. **Cluster admin:** Ensure that proper RBAC is in place to grant permission to the destination namespace owner to create the TridentVolumeReference CR in the destination namespace (`namespace2`).
3. **Destination namespace owner:** Create a TridentVolumeReference CR in the destination namespace that refers to the source namespace `pvc1`.

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

4. **Destination namespace owner:** Create a PVC (`pvc2`) in destination namespace (`namespace2`) using the `cloneFromPVC` or `cloneFromSnapshot`, and `cloneFromNamespace` annotations to designate the source PVC.

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

Limitations

- For PVCs provisioned using ontap-nas-economy drivers, read-only clones are not supported.

Replicate volumes using SnapMirror

Trident supports mirror relationships between a source volume on one cluster and the destination volume on the peered cluster for replicating data for disaster recovery. You can use a namespaced Custom Resource Definition (CRD), called Trident Mirror Relationship (TMR) to perform the following operations:

- Create mirror relationships between volumes (PVCs)
- Remove mirror relationships between volumes
- Break the mirror relationships
- Promote the secondary volume during disaster conditions (failovers)

- Perform lossless transition of applications from cluster to cluster (during planned failovers or migrations)

Replication prerequisites

Ensure that the following prerequisites are met before you begin:

ONTAP clusters

- **Trident**: Trident version 22.10 or later must exist on both the source and destination Kubernetes clusters that utilize ONTAP as a backend.
- **Licenses**: ONTAP SnapMirror asynchronous licenses using the Data Protection bundle must be enabled on both the source and destination ONTAP clusters. Refer to SnapMirror licensing overview in ONTAP for more information.

  Beginning with ONTAP 9.10.1, all licenses are delivered as a NetApp license file (NLF), which is a single file that enables multiple features. Refer to Licenses included with ONTAP One for more information.

  > Only SnapMirror asynchronous protection is supported.

Peering

- **Cluster and SVM**: The ONTAP storage backends must be peered. Refer to Cluster and SVM peering overview for more information.

  > Ensure that the SVM names used in the replication relationship between two ONTAP clusters are unique.

- **Trident and SVM**: The peered remote SVMs must be available to Trident on the destination cluster.

Supported drivers

NetApp Trident supports volume replication with NetApp SnapMirror technology using storage classes backed by the following drivers:
`ontap-nas`: NFS
`ontap-san`: iSCSI
`ontap-san`: FC
`ontap-san`: NVMe/TCP (requires minimum ONTAP version 9.15.1)

> Volume replication using SnapMirror is not supported for ASA r2 systems. For information about ASA r2 systems, see Learn about ASA r2 storage systems.

Create a mirrored PVC

Follow these steps and use the CRD examples to create mirror relationship between primary and secondary volumes.

Steps

1. Perform the following steps on the primary Kubernetes cluster:
   a. Create a StorageClass object with the `trident.netapp.io/replication: true` parameter.

Example

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "nfs"
  trident.netapp.io/replication: "true"
```

b. Create a PVC with previously created StorageClass.

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. Create a MirrorRelationship CR with local information.

Example

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident fetches the internal information for the volume and the volume's current data protection (DP) state, then populates the status field of the MirrorRelationship.

d. Get the TridentMirrorRelationship CR to obtain the internal name and SVM of the PVC.

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. Perform the following steps on the secondary Kubernetes cluster:
   a. Create a StorageClass with the trident.netapp.io/replication: true parameter.

      Example

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        trident.netapp.io/replication: true
      ```

   b. Create a MirrorRelationship CR with destination and source information.

Example

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Trident will create a SnapMirror relationship with the configured relationship policy name (or default for ONTAP) and initialize it.

c. Create a PVC with previously created StorageClass to act as the secondary (SnapMirror destination).

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Trident will check for the TridentMirrorRelationship CRD and fail to create the volume if the relationship does not exist. If the relationship exists, Trident will ensure the new FlexVol volume is placed onto an SVM that is peered with the remote SVM defined in the MirrorRelationship.

Volume Replication States

A Trident Mirror Relationship (TMR) is a CRD that represents one end of a replication relationship between PVCs. The destination TMR has a state, which tells Trident what the desired state is. The destination TMR has the following states:

- **Established**: the local PVC is the destination volume of a mirror relationship, and this is a new relationship.
- **Promoted**: the local PVC is ReadWrite and mountable, with no mirror relationship currently in effect.
- **Reestablished**: the local PVC is the destination volume of a mirror relationship and was also previously in

that mirror relationship.

- The reestablished state must be used if the destination volume was ever in a relationship with the source volume because it overwrites the destination volume contents.
- The reestablished state will fail if the volume was not previously in a relationship with the source.

Promote secondary PVC during an unplanned failover

Perform the following step on the secondary Kubernetes cluster:

- Update the *spec.state* field of TridentMirrorRelationship to `promoted`.

Promote secondary PVC during a planned failover

During a planned failover (migration), perform the following steps to promote the secondary PVC:

Steps

1. On the primary Kubernetes cluster, create a snapshot of the PVC and wait until the snapshot is created.
2. On the primary Kubernetes cluster, create the SnapshotInfo CR to obtain internal details.

   Example

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

3. On secondary Kubernetes cluster, update the *spec.state* field of the *TridentMirrorRelationship* CR to *promoted* and *spec.promotedSnapshotHandle* to be the internalName of the snapshot.
4. On secondary Kubernetes cluster, confirm the status (status.state field) of TridentMirrorRelationship to promoted.

Restore a mirror relationship after a failover

Before restoring a mirror relationship, choose the side that you want to make as the new primary.

Steps

1. On the secondary Kubernetes cluster, ensure that the values for the *spec.remoteVolumeHandle* field on the TridentMirrorRelationship is updated.
2. On secondary Kubernetes cluster, update the *spec.mirror* field of TridentMirrorRelationship to `reestablished`.

Additional operations

Trident supports the following operations on the primary and secondary volumes:

Replicate primary PVC to a new secondary PVC

Ensure that you already have a primary PVC and a secondary PVC.

Steps

1. Delete the PersistentVolumeClaim and TridentMirrorRelationship CRDs from the established secondary (destination) cluster.
2. Delete the TridentMirrorRelationship CRD from the primary (source) cluster.
3. Create a new TridentMirrorRelationship CRD on the primary (source) cluster for the new secondary (destination) PVC you want to establish.

Resize a mirrored, primary or secondary PVC

The PVC can be resized as normal, ONTAP will automatically expand any destination flevxols if the amount of data exceeds the current size.

Remove replication from a PVC

To remove replication, perform one of the following operations on the current secondary volume:

- Delete the MirrorRelationship on the secondary PVC. This breaks the replication relationship.
- Or, update the spec.state field to *promoted*.

Delete a PVC (that was previously mirrored)

Trident checks for replicated PVCs, and releases the replication relationship before attempting to delete the volume.

Delete a TMR

Deleting a TMR on one side of a mirrored relationship causes the remaining TMR to transition to *promoted* state before Trident completes the deletion. If the TMR selected for deletion is already in *promoted* state, there is no existing mirror relationship and the TMR will be removed and Trident will promote the local PVC to *ReadWrite*. This deletion releases SnapMirror metadata for the local volume in ONTAP. If this volume is used in a mirror relationship in the future, it must use a new TMR with an *established* volume replication state when creating the new mirror relationship.

Update mirror relationships when ONTAP is online

Mirror relationships can be updated any time after they are established. You can use the `state: promoted` or `state: reestablished` fields to update the relationships.
When promoting a destination volume to a regular ReadWrite volume, you can use *promotedSnapshotHandle* to specify a specific snapshot to restore the current volume to.

Update mirror relationships when ONTAP is offline

You can use a CRD to perform a SnapMirror update without Trident having direct connectivity to the ONTAP cluster. Refer to the following example format of the TridentActionMirrorUpdate:

Example

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` reflects the state of the TridentActionMirrorUpdate CRD. It can take a value from *Succeeded*, *In Progress*, or *Failed*.

Use CSI Topology

Trident can selectively create and attach volumes to nodes present in a Kubernetes cluster by making use of the CSI Topology feature.

Overview

Using the CSI Topology feature, access to volumes can be limited to a subset of nodes, based on regions and availability zones. Cloud providers today enable Kubernetes administrators to spawn nodes that are zone based. Nodes can be located in different availability zones within a region, or across various regions. To facilitate the provisioning of volumes for workloads in a multi-zone architecture, Trident uses CSI Topology.

Learn more about the CSI Topology feature here.

Kubernetes provides two unique volume binding modes:

- With `VolumeBindingMode` set to `Immediate`, Trident creates the volume without any topology awareness. Volume binding and dynamic provisioning are handled when the PVC is created. This is the default `VolumeBindingMode` and is suited for clusters that do not enforce topology constraints. Persistent Volumes are created without having any dependency on the requesting pod's scheduling requirements.
- With `VolumeBindingMode` set to `WaitForFirstConsumer`, the creation and binding of a Persistent Volume for a PVC is delayed until a pod that uses the PVC is scheduled and created. This way, volumes are created to meet the scheduling constraints that are enforced by topology requirements.

The `WaitForFirstConsumer` binding mode does not require topology labels. This can be used independent of the CSI Topology feature.

What you'll need

To make use of CSI Topology, you need the following:

- A Kubernetes cluster running a supported Kubernetes version

```
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"}
```

- Nodes in the cluster should have labels that introduce topology awareness (`topology.kubernetes.io/region` and `topology.kubernetes.io/zone`). These labels **should be present on nodes in the cluster** before Trident is installed for Trident to be topology aware.

```
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"}]
```

Step 1: Create a topology-aware backend

Trident storage backends can be designed to selectively provision volumes based on availability zones. Each backend can carry an optional `supportedTopologies` block that represents a list of zones and regions that are supported. For StorageClasses that make use of such a backend, a volume would only be created if requested by an application that is scheduled in a supported region/zone.

Here is an example backend definition:

YAML

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

JSON

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

`supportedTopologies` is used to provide a list of regions and zones per backend. These regions and zones represent the list of permissible values that can be provided in a StorageClass. For StorageClasses that contain a subset of the regions and zones provided in a backend, Trident creates a volume on the backend.

You can define `supportedTopologies` per storage pool as well. See the following example:

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

In this example, the `region` and `zone` labels stand for the location of the storage pool. `topology.kubernetes.io/region` and `topology.kubernetes.io/zone` dictate where the storage pools can be consumed from.

Step 2: Define StorageClasses that are topology aware

Based on the topology labels that are provided to the nodes in the cluster, StorageClasses can be defined to contain topology information. This will determine the storage pools that serve as candidates for PVC requests made, and the subset of nodes that can make use of the volumes provisioned by Trident.

See the following example:

```
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
```

In the StorageClass definition provided above, `volumeBindingMode` is set to `WaitForFirstConsumer`. PVCs that are requested with this StorageClass will not be acted upon until they are referenced in a pod. And, `allowedTopologies` provides the zones and region to be used. The `netapp-san-us-east1` StorageClass creates PVCs on the `san-backend-us-east1` backend defined above.

Step 3: Create and use a PVC

With the StorageClass created and mapped to a backend, you can now create PVCs.

See the example `spec` below:

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

Creating a PVC using this manifest would result in the following:

```
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
```

For Trident to create a volume and bind it to the PVC, use the PVC in a pod. See the following example:

```
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
```

This podSpec instructs Kubernetes to schedule the pod on nodes that are present in the `us-east1` region, and choose from any node that is present in the `us-east1-a` or `us-east1-b` zones.

See the following output:

```
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
```

Update backends to include `supportedTopologies`

Pre-existing backends can be updated to include a list of `supportedTopologies` using `tridentctl backend update`. This will not affect volumes that have already been provisioned, and will only be used for subsequent PVCs.

Find more information

- Manage resources for containers
- nodeSelector
- Affinity and anti-affinity
- Taints and Tolerations

Work with snapshots

Kubernetes volume snapshots of Persistent Volumes (PVs) enable point-in-time copies of volumes. You can create a snapshot of a volume created using Trident, import a snapshot created outside of Trident, create a new volume from an existing snapshot, and recover volume data from snapshots.

Overview

Volume snapshot is supported by `ontap-nas`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `azure-netapp-files`, and `google-cloud-netapp-volumes` drivers.

Before you begin

You must have an external snapshot controller and Custom Resource Definitions (CRDs) to work with snapshots. This is the responsibility of the Kubernetes orchestrator (for example: Kubeadm, GKE, OpenShift).

If your Kubernetes distribution does not include the snapshot controller and CRDs, refer to Deploy a volume snapshot controller.

> Don't create a snapshot controller if creating on-demand volume snapshots in a GKE environment. GKE uses a built-in, hidden snapshot controller.

Create a volume snapshot

Steps

1. Create a `VolumeSnapshotClass`. For more information, refer to VolumeSnapshotClass.
   - The `driver` points to the Trident CSI driver.
   - `deletionPolicy` can be `Delete` or `Retain`. When set to `Retain`, the underlying physical snapshot on the storage cluster is retained even when the `VolumeSnapshot` object is deleted.

     Example

     ```
     cat snap-sc.yaml
     ```

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

2. Create a snapshot of an existing PVC.

   Examples

   - This example creates a snapshot of an existing PVC.

     ```
     cat snap.yaml
     ```

     ```
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshot
     metadata:
       name: pvc1-snap
     spec:
       volumeSnapshotClassName: csi-snapclass
       source:
         persistentVolumeClaimName: pvc1
     ```

   - This example creates a volume snapshot object for a PVC named `pvc1` and the name of the snapshot is set to `pvc1-snap`. A VolumeSnapshot is analogous to a PVC and is associated with a `VolumeSnapshotContent` object that represents the actual snapshot.

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

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

- You can identify the `VolumeSnapshotContent` object for the `pvc1-snap` VolumeSnapshot by describing it. The `Snapshot Content Name` identifies the VolumeSnapshotContent object which serves this snapshot. The `Ready To Use` parameter indicates that the snapshot can be used to create a new PVC.

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

Create a PVC from a volume snapshot

You can use `dataSource` to create a PVC using a VolumeSnapshot named `<pvc-name>` as the source of the data. After the PVC is created, it can be attached to a pod and used just like any other PVC.

> The PVC will be created in the same backend as the source volume. Refer to KB: Creating a PVC from a Trident PVC Snapshot cannot be created in an alternate backend.

The following example creates the PVC using `pvc1-snap` as the data source.

```
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
```

Import a volume snapshot

Trident supports the Kubernetes pre-provisioned snapshot process to enable the cluster administrator to create a `VolumeSnapshotContent` object and import snapshots created outside of Trident.

Before you begin

Trident must have created or imported the snapshot's parent volume.

Steps

1. **Cluster admin:** Create a `VolumeSnapshotContent` object that references the backend snapshot. This initiates the snapshot workflow in Trident.
   - Specify the name of the backend snapshot in `annotations` as `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`.
   - Specify `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` in `snapshotHandle`. This is the only information provided to Trident by the external snapshotter in the `ListSnapshots` call.

     > The `<volumeSnapshotContentName>` cannot always match the backend snapshot name due to CR naming constraints.

     Example

     The following example creates a `VolumeSnapshotContent` object that references backend snapshot `snap-01`.

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. **Cluster admin:** Create the `VolumeSnapshot` CR that references the `VolumeSnapshotContent` object. This requests access to use the `VolumeSnapshot` in a given namespace.

   Example

   The following example creates a `VolumeSnapshot` CR named `import-snap` that references the `VolumeSnapshotContent` named `import-snap-content`.

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: import-snap
   spec:
     # volumeSnapshotClassName: csi-snapclass (not required for pre-
   provisioned or imported snapshots)
     source:
       volumeSnapshotContentName: import-snap-content
   ```

3. **Internal processing (no action required):** The external snapshotter recognizes the newly created `VolumeSnapshotContent` and runs the `ListSnapshots` call. Trident creates the `TridentSnapshot`.
   - The external snapshotter sets the `VolumeSnapshotContent` to `readyToUse` and the `VolumeSnapshot` to `true`.
   - Trident returns `readyToUse=true`.
4. **Any user:** Create a `PersistentVolumeClaim` to reference the new `VolumeSnapshot`, where the `spec.dataSource` (or `spec.dataSourceRef`) name is the `VolumeSnapshot` name.

   Example

The following example creates a PVC referencing the `VolumeSnapshot` named `import-snap`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

Recover volume data using snapshots

The snapshot directory is hidden by default to facilitate maximum compatibility of volumes provisioned using the `ontap-nas` and `ontap-nas-economy` drivers. Enable the `.snapshot` directory to recover data from snapshots directly.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

In-place volume restoration from a snapshot

Trident provides rapid, in-place volume restoration from a snapshot using the `TridentActionSnapshotRestore` (TASR) CR. This CR functions as an imperative Kubernetes action and does not persist after the operation completes.

Trident supports snapshot restore on the `ontap-san`, `ontap-san-economy`, `ontap-nas`, `ontap-nas-flexgroup`, `azure-netapp-files`, `google-cloud-netapp-volumes`, and `solidfire-san` drivers.

Before you begin

You must have a bound PVC and available volume snapshot.

- Verify the PVC status is bound.

```
kubectl get pvc
```

- Verify the volume snapshot is ready to use.

```
kubectl get vs
```

Steps

1. Create the TASR CR. This example creates a CR for PVC `pvc1` and volume snapshot `pvc1-snapshot`.

   The TASR CR must be in a namespace where the PVC & VS exist.

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. Apply the CR to restore from the snapshot. This example restores from snapshot `pvc1`.

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

Results

Trident restores the data from the snapshot. You can verify the snapshot restore status:

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- In most cases, Trident will not automatically retry the operation in case of failure. You will need to perform the operation again.
- Kubernetes users without admin access might have to be granted permission by the admin to create a TASR CR in their application namespace.

Delete a PV with associated snapshots

When deleting a Persistent Volume with associated snapshots, the corresponding Trident volume is updated to a "Deleting state". Remove the volume snapshots to delete the Trident volume.

Deploy a volume snapshot controller

If your Kubernetes distribution does not include the snapshot controller and CRDs, you can deploy them as follows.

Steps

1. Create volume snapshot CRDs.

```
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. Create the snapshot controller.

```
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
```

If necessary, open `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` and update `namespace` to your namespace.

Related links

- Volume snapshots
- VolumeSnapshotClass

Work with volume group snapshots

Kubernetes volume group snapshots of Persistent Volumes (PVs) NetApp Trident provides the ability to create snapshots of multiple volumes (a group of volume snapshots). This volume group snapshot represents copies from multiple volumes that are taken at the same point-in-time.

VolumeGroupSnapshot is a beta feature in Kubernetes with beta APIs. Kubernetes 1.32 is the minimum version required for VolumeGroupSnapshot.

Create volume group snapshots

Volume group snapshot is supported with the following storage drivers:

- `ontap-san` driver - only for the iSCSI and FC protocols, not for the NVMe/TCP protocol.
- `ontap-san-economy` - only for the iSCSI protocol.
- `ontap-nas`

> Volume group snapshot is not supported for NetApp ASA r2 or AFX storage systems.

Before you begin

- Ensure that your Kubernetes version is K8s 1.32 or higher.
- You must have an external snapshot controller and Custom Resource Definitions (CRDs) to work with snapshots. This is the responsibility of the Kubernetes orchestrator (for example: Kubeadm, GKE, OpenShift).

  If your Kubernetes distribution does not include the external snapshot controller and CRDs, refer to Deploy a volume snapshot controller.

  > Don't create a snapshot controller if creating on-demand volume group snapshots in a GKE environment. GKE uses a built-in, hidden snapshot controller.

- In the snapshot controller YAML, set the `CSIVolumeGroupSnapshot` feature gate to 'true' to ensure that volume group snapshot is enabled.
- Create the required volume group snapshot classes before creating a volume group snapshot.
- Ensure that all PVCs/volumes are on the same SVM to be able to create VolumeGroupSnapshot.

Steps

- Create a VolumeGroupSnapshotClass prior to creating a VolumeGroupSnapshot. For more information, refer to VolumeGroupSnapshotClass.

```
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
```

- Create PVCs with required labels using existing storage classes, or add these labels to existing PVCs.

  The following example creates the PVC using `pvc1-group-snap` as the data source and label `consistentGroupSnapshot: groupA`. Define the label key and value based on your requirements.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- Create a VolumeGroupSnapshot with the same label (`consistentGroupSnapshot: groupA`) specified in the PVC.

  This example creates a volume group snapshot:

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

Recover volume data using a group snapshot

You can restore individual Persistent Volumes using the individual snapshots which have been created as part of the Volume Group Snapshot. You cannot recover the Volume Group Snapshot as a unit.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

In-place volume restoration from a snapshot

Trident provides rapid, in-place volume restoration from a snapshot using the `TridentActionSnapshotRestore` (TASR) CR. This CR functions as an imperative Kubernetes action and does not persist after the operation completes.

For more information, see In-place volume restoration from a snapshot.

Delete a PV with associated group snapshots

When deleting a group volume snapshot:

- You can delete VolumeGroupSnapshots as a whole, not individual snapshots in the group.
- If PersistentVolumes are deleted while a snapshot exists for that PersistentVolume, Trident will move that volume to a "deleting" state because the snapshot must be removed before the volume can be safely removed.
- If a clone has been created using a grouped snapshot and then the group is to be deleted, a split-on-clone operation will begin and the group cannot be deleted until the split is complete.

Deploy a volume snapshot controller

If your Kubernetes distribution does not include the snapshot controller and CRDs, you can deploy them as follows.

Steps

1. Create volume snapshot CRDs.

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.yaml
```

2. Create the snapshot controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

If necessary, open `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` and update `namespace` to your namespace.

Related links

- VolumeGroupSnapshotClass
- Volume snapshots

Manage and monitor Trident

Upgrade Trident

Upgrade Trident

Beginning with the 24.02 release, Trident follows a four-month release cadence, delivering three major releases every calendar year. Each new release builds on the previous releases and provides new features, performance enhancements, bug fixes, and improvements. We encourage you to upgrade at least once a year to take advantage of the new features in Trident.

Considerations before upgrading

When upgrading to the latest release of Trident, consider the following:

- There should be only one Trident instance installed across all the namespaces in a given Kubernetes cluster.
- Trident 23.07 and later requires v1 volume snapshots and no longer supports alpha or beta snapshots.
- When upgrading, it is important you provide `parameter.fsType` in `StorageClasses` used by Trident. You can delete and re-create `StorageClasses` without disrupting pre-existing volumes.
  - This is a **requirement** for enforcing security contexts for SAN volumes.
  - The sample input directory contains examples, such as `storage-class-basic.yaml.templ` and `storage-class-bronze-default.yaml`.
  - For more information, refer to Known Issues.

Step 1: Select a version

Trident versions follow a date-based `YY.MM` naming convention, where "YY" is the last two digits of the year and "MM" is the month. Dot releases follow a `YY.MM.X` convention, where "X" is the patch level. You will select the version to upgrade to based on the version you are upgrading from.

- You can perform a direct upgrade to any target release that is within a four-release window of your installed version. For example, you can directly upgrade from 24.06 (or any 24.06 dot release) to 25.06.
- If you are upgrading from a release outside of the four-release window, perform a multi-step upgrade. Use the upgrade instructions for the earlier version you are upgrading from to upgrade to the most recent release that fits the four-release window. For example, if you are running 23.07 and want to upgrade to 25.06:
  1. First upgrade from 23.07 to 24.06.
  2. Then upgrade from 24.06 to 25.06.

> When upgrading using the Trident operator on OpenShift Container Platform, you should upgrade to Trident 21.01.1 or later. The Trident operator released with 21.01.0 contains a known issue that has been fixed in 21.01.1. For more details, refer to the issue details on GitHub.

Step 2: Determine the original installation method

To determine which version you used to originally install Trident:

1. Use `kubectl get pods -n trident` to examine the pods.
   - If there is no operator pod, Trident was installed using `tridentctl`.
   - If there is an operator pod, Trident was installed using the Trident operator either manually or using Helm.
2. If there is an operator pod, use `kubectl describe torc` to determine if Trident was installed using Helm.
   - If there is a Helm label, Trident was installed using Helm.
   - If there is no Helm label, Trident was installed manually using the Trident operator.

Step 3: Select an upgrade method

Generally, you should upgrade using the same method you used for the initial installation, however you can move between installation methods. There are two options to upgrade Trident.

- Upgrade using the Trident operator

We suggest you review Understand the operator upgrade workflow before upgrading with the operator.

- Upgrade using `tridentctl`

Upgrade with the operator

Understand the operator upgrade workflow

Before using the Trident operator to upgrade Trident, you should understand the background processes that occur during upgrade. This includes changes to the Trident controller, controller Pod and node Pods, and node DaemonSet that enable rolling updates.

Trident operator upgrade handling

One of the many benefits of using the Trident operator to install and upgrade Trident is the automatic handling of Trident and Kubernetes objects without disrupting existing mounted volumes. In this way, Trident can support upgrades with zero downtime, or *rolling updates*. In particular, the Trident operator communicates with the Kubernetes cluster to:

- Delete and recreate the Trident Controller deployment and node DaemonSet.
- Replace the Trident Controller Pod and Trident Node Pods with new versions.
  - If a node is not updated, it does not prevent remaining nodes from being updated.
  - Only nodes with a running Trident Node Pod can mount volumes.

For more information about Trident architecture on the Kubernetes cluster, refer to Trident architecture.

Operator upgrade workflow

When you initiate an upgrade using the Trident operator:

1. The **Trident operator**:
   a. Detects the currently installed version of Trident (version *n*).
   b. Updates all Kubernetes objects including CRDs, RBAC, and Trident SVC.
   c. Deletes the Trident Controller deployment for version *n*.
   d. Creates the Trident Controller deployment for version *n+1*.
2. **Kubernetes** creates Trident Controller Pod for *n+1*.
3. The **Trident operator**:
   a. Deletes the Trident Node DaemonSet for *n*. The operator does not wait for Node Pod termination.
   b. Creates the Trident Node Daemonset for *n+1*.
4. **Kubernetes** creates Trident Node Pods on nodes not running Trident Node Pod *n*. This ensures there is never more than one Trident Node Pod, of any version, on a node.

Upgrade a Trident installation using Trident operator or Helm

You can upgrade Trident using the Trident operator either manually or using Helm. You can upgrade from a Trident operator installation to another Trident operator installation or upgrade from a `tridentctl` installation to a Trident operator version. Review Select an upgrade method before upgrading a Trident operator installation.

Upgrade a manual installation

You can upgrade from a cluster-scoped Trident operator installation to another cluster-scoped Trident operator installation. All Trident versions use a cluster-scoped operator.

> To upgrade from Trident that was installed using the namespace-scoped operator (versions 20.07 through 20.10), use the upgrade instructions for your installed version of Trident.

About this task

Trident provides a bundle file you can use to install the operator and create associated objects for your Kubernetes version.

- For clusters running Kubernetes 1.25 or later, use bundle_post_1_25.yaml.

Before you begin

Ensure you are using a Kubernetes cluster running a supported Kubernetes version.

Steps

1. Verify your Trident version:

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

2. Update the `operator.yaml`, `tridentorchestrator_cr.yaml`, and `post_1_25_bundle.yaml` with the registry and imagepaths for the version you are upgrading to (e.g.25.06), and the correct secret.

3. Delete the Trident operator that was used to install the current Trident instance. For example, if you are upgrading from 25.02, run the following command:

```
kubectl delete -f 25.02.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

4. If you customized your initial installation using `TridentOrchestrator` attributes, you can edit the `TridentOrchestrator` object to modify the installation parameters. This might include changes made to specify mirrored Trident and CSI image registries for offline mode, enable debug logs, or specify image pull secrets.
5. Install Trident using the correct bundle YAML file for your environment, where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version. For example, if you are installing Trident 25.06.0, run the following command:

```
kubectl create -f 25.06.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

6. Edit the trident torc to include the image 25.06.0.

Upgrade a Helm installation

You can upgrade a Trident Helm installation.

> When upgrading a Kubernetes cluster from 1.24 to 1.25 or later that has Trident installed, you must update values.yaml to set `excludePodSecurityPolicy` to `true` or add `--set excludePodSecurityPolicy=true` to the `helm upgrade` command before you can upgrade the cluster.

If you have already upgraded your Kubernetes cluster from 1.24 to 1.25 without upgrading the Trident helm, the helm upgrade fails. For the helm upgrade to go through, perform these steps as pre-requisites:

1. Install the helm-mapkubeapis plugin from https://github.com/helm/helm-mapkubeapis.
2. Perform a dry run for the Trident release in the namespace where Trident is installed. This lists out the resources, which will be cleaned up.

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. Perform a full run with helm to do the cleanup.

```
helm mapkubeapis trident  --namespace trident
```

Steps

1. If you installed Trident using Helm, you can use `helm upgrade trident netapp-trident/trident-operator --version 100.2602.0` to upgrade in one step. If you did not add the Helm repo or cannot use it to upgrade:

a. Download the latest Trident release from the *Assets* section on GitHub.

b. Use the `helm upgrade` command where `trident-operator-26.02.0.tgz` reflects the version that you want to upgrade to.

```
helm upgrade <name> trident-operator-26.02.0.tgz
```

> If you set custom options during the initial installation (such as specifying private, mirrored registries for Trident and CSI images), append the `helm upgrade` command using `--set` to ensure those options are included in the upgrade command, otherwise the values will reset to default.

2. Run `helm list` to verify that the chart and app version have both been upgraded. Run `tridentctl logs` to review any debug messages.

Upgrade from a `tridentctl` installation to Trident operator

You can upgrade to the latest release of the Trident operator from a `tridentctl` installation. The existing backends and PVCs will automatically be available.

> Before switching between installation methods, review Moving between installation methods.

Steps

1. Download the latest Trident release.

```
# Download the release required [26.02.0]
mkdir 26.02.0
cd 26.02.0
wget
https://github.com/NetApp/trident/releases/download/v26.02.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

2. Create the `tridentorchestrator` CRD from the manifest.

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

3. Deploy the cluster-scoped operator in the same namespace.

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

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

4. Create a `TridentOrchestrator` CR for installing Trident.

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

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m
trident-csi-xrst8                   2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0          5m41s
```

5. Confirm Trident was upgraded to the intended version.

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v26.02.0
```

Upgrade with tridentctl

You can easily upgrade an existing Trident installation using `tridentctl`.

About this task

Uninstalling and reinstalling Trident acts as an upgrade. When you uninstall Trident, the Persistent Volume Claim (PVC) and Persistent Volume (PV) used by the Trident deployment are not deleted. PVs that have already been provisioned will remain available while Trident is offline, and Trident will provision volumes for any PVCs that are created in the interim after it is back online.

Before you begin

Review Select an upgrade method before upgrading using `tridentctl`.

Steps

1. Run the uninstall command in `tridentctl` to remove all of the resources associated with Trident except for the CRDs and related objects.

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

2. Reinstall Trident. Refer to Install Trident using tridentctl.

> Do not interrupt the upgrade process. Ensure the installer runs to completion.

Manage Trident using tridentctl

The Trident installer bundle includes the `tridentctl` command-line utility to provide simple access to Trident. Kubernetes users with sufficient privileges can use it to install Trident or manage the namespace that contains the Trident pod.

Commands and global flags

You can run `tridentctl help` to get a list of available commands for `tridentctl` or append the `--help` flag to any command to get a list of options and flags for that specific command.

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

The Trident `tridentctl` utility supports the following commands and global flags.

Commands

`create`
Add a resource to Trident.

`delete`
Remove one or more resources from Trident.

`get`
Get one or more resources from Trident.

`help`
Help about any command.

`images`
Print a table of the container images Trident needs.

`import`
Import an existing resource to Trident.

`install`
Install Trident.

`logs`
Print the logs from Trident.

`send`
Send a resource from Trident.

`uninstall`
Uninstall Trident.

`update`
Modify a resource in Trident.

`update backend state`
Temporarily suspend backend operations.

`upgrade`
Upgrade a resource in Trident.

`version`
Print the version of Trident.

Global flags

-d, --debug
Debug output.

-h, --help
Help for `tridentctl`.

-k, --kubeconfig string
Specify the `KUBECONFIG` path to run commands locally or from one Kubernetes cluster to another.

> Alternatively, you can export the `KUBECONFIG` variable to point to a specific Kubernetes cluster and issue `tridentctl` commands to that cluster.

-n, --namespace string
Namespace of Trident deployment.

-o, --output string
Output format. One of json|yaml|name|wide|ps (default).

-s, --server string
Address/port of Trident REST interface.

> Trident REST interface can be configured to listen and serve at 127.0.0.1 (for IPv4) or [::1] (for IPv6) only.

Command options and flags

create

Use the `create` command to add a resource to Trident.

```
tridentctl create [option]
```

Options

`backend`: Add a backend to Trident.

delete

Use the `delete` command to remove one or more resources from Trident.

```
tridentctl delete [option]
```

Options

`backend`: Delete one or more storage backends from Trident.
`snapshot`: Delete one or more volume snapshots from Trident.
`storageclass`: Delete one or more storage classes from Trident.
`volume`: Delete one or more storage volumes from Trident.

get

Use the `get` command to get one or more resources from Trident.

```
tridentctl get [option]
```

Options

`backend`: Get one or more storage backends from Trident.
`snapshot`: Get one or more snapshots from Trident.
`storageclass`: Get one or more storage classes from Trident.
`volume`: Get one or more volumes from Trident.

Flags

`-h`, `--help`: Help for volumes.
`--parentOfSubordinate string`: Limit query to subordinate source volume.
`--subordinateOf string`: Limit query to subordinates of volume.

images

Use `images` flags to print a table of the container images Trident needs.

```
tridentctl images [flags]
```

Flags

`-h`, `--help`: Help for images.
`-v`, `--k8s-version string`: Semantic version of Kubernetes cluster.

import volume

Use the `import volume` command to import an existing volume to Trident.

```
tridentctl import volume <backendName> <volumeName> [flags]
```

Aliases

`volume`, `v`

Flags

`-f`, `--filename string`: Path to YAML or JSON PVC file.
`-h`, `--help`: Help for volume.
`--no-manage`: Create PV/PVC only. Don’t assume volume lifecycle management.

install

Use the `install` flags to install Trident.

```
tridentctl install [flags]
```

Flags

`--autosupport-image string`: The container image for Autosupport Telemetry (default "netapp/trident autosupport:<current-version>").
`--autosupport-proxy string`: The address/port of a proxy for sending Autosupport Telemetry.

`--enable-node-prep`: Attempt to install required packages on nodes.
`--generate-custom-yaml`: Generate YAML files without installing anything.
`-h`, `--help`: Help for install.
`--http-request-timeout`: Override the HTTP request timeout for Trident controller's REST API (default 1m30s).
`--image-registry string`: The address/port of an internal image registry.
`--k8s-timeout duration`: The timeout for all Kubernetes operations (default 3m0s).
`--kubelet-dir string`: The host location of kubelet's internal state (default "/var/lib/kubelet").
`--log-format string`: The Trident logging format (text, json) (default "text").
`--node-prep`: Enables Trident to prepare the nodes of the Kubernetes cluster to manage volumes using the specified data storage protocol. **Currently, `iscsi` is the only value supported. Beginning with OpenShift 4.19, the minimum Trident version supported for this feature is 25.06.1.**
`--pv string`: The name of the legacy PV used by Trident, makes sure this doesn't exist (default "trident").
`--pvc string`: The name of the legacy PVC used by Trident, makes sure this doesn't exist (default "trident").
`--silence-autosupport`: Don't send autosupport bundles to NetApp automatically (default true).
`--silent`: Disable most output during installation.
`--trident-image string`: The Trident image to install.
`--k8s-api-qps`: The queries per second (QPS) limit for Kubernetes API requests (default 100; optional).
`--use-custom-yaml`: Use any existing YAML files that exist in setup directory.
`--use-ipv6`: Use IPv6 for Trident's communication.

logs

Use `logs` flags to print the logs from Trident.

```
tridentctl logs [flags]
```

Flags

`-a`, `--archive`: Create a support archive with all logs unless otherwise specified.
`-h`, `--help`: Help for logs.
`-l`, `--log string`: Trident log to display. One of trident|auto|trident-operator|all (default "auto").
`--node string`: The Kubernetes node name from which to gather node pod logs.
`-p`, `--previous`: Get the logs for the previous container instance if it exists.
`--sidecars`: Get the logs for the sidecar containers.

send

Use the `send` command to send a resource from Trident.

```
tridentctl send [option]
```

Options

`autosupport`: Send an Autosupport archive to NetApp.

uninstall

Use `uninstall` flags to uninstall Trident.

```
tridentctl uninstall [flags]
```

Flags

`-h, --help`: Help for uninstall.
`--silent`: Disable most output during uninstall.

update

Use the `update` command to modify a resource in Trident.

```
tridentctl update [option]
```

Options

`backend`: Update a backend in Trident.

update backend state

Use the `update backend state` command to suspend or resume backend operations.

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

Points to consider

- If a backend is created using a TridentBackendConfig (tbc), the backend cannot be updated using a `backend.json` file.
- If the `userState` has been set in a tbc, it cannot be modified using the `tridentctl update backend state <backend-name> --user-state suspended/normal` command.
- To regain the ability to set the `userState` via tridentctl after it has been set via tbc, the `userState` field must be removed from the tbc. This can be done using the `kubectl edit tbc` command. After the `userState` field is removed, you can use the `tridentctl update backend state` command to change the `userState` of a backend.
- Use the `tridentctl update backend state` to change the `userState`. You can also update the `userState` using `TridentBackendConfig` or `backend.json` file; this triggers a complete re-initialization of the backend and can be time-consuming.

  #### Flags

  `-h, --help`: Help for backend state.
  `--user-state`: Set to `suspended` to pause backend operations. Set to `normal` to resume backend operations. When set to `suspended`:

- `AddVolume` and `Import Volume` are paused.
- `CloneVolume`, `ResizeVolume`, `PublishVolume`, `UnPublishVolume`, `CreateSnapshot`, `GetSnapshot`, `RestoreSnapshot`, `DeleteSnapshot`, `RemoveVolume`, `GetVolumeExternal`, `ReconcileNodeAccess` remain available.

You can also update the backend state using `userState` field in the backend configuration file `TridentBackendConfig` or `backend.json`.
For more information, refer to Options for managing backends and Perform backend management with kubectl.

Example:

JSON

Follow these steps to update the `userState` using the `backend.json` file:

1. Edit the `backend.json` file to include the `userState` field with its value set to 'suspended'.
2. Update the backend using the `tridentctl update backend` command and the path to the updated `backend.json` file.

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

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "<redacted>",
  "svm": "nas-svm",
  "backendName": "customBackend",
  "username": "<redacted>",
  "password": "<redacted>",
  "userState": "suspended"
}
```

YAML

You can edit the tbc after it has been applied using the `kubectl edit <tbc-name> -n <namespace>` command.
The following example updates the backend state to suspend using the `userState: suspended` option:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
  userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

version

Use `version` flags to print the version of `tridentctl` and the running Trident service.

```
tridentctl version [flags]
```

Flags

`--client`: Client version only (no server required).
`-h, --help`: Help for version.

Plugin support

Tridentctl supports plugins similar to kubectl. Tridentctl detects a plugin if the plugin binary file name follows the scheme "tridentctl-<plugin>", and the binary is located in a folder listed the PATH environment variable. All the detected plugins are listed in the plugin section of the tridentctl help. Optionally, you can also limit the search by specifying a plugin folder in the the enviornment variable TRIDENTCTL_PLUGIN_PATH (Example: `TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`). If the variable is used, tridenctl searches only in the specified folder.

Monitor Trident

Trident provides a set of Prometheus metrics endpoints that you can use to monitor Trident performance.

Overview

The metrics provided by Trident enable you to do the following:

- Keep tabs on Trident's health and configuration. You can examine how successful operations are and if it can communicate with the backends as expected.
- Examine backend usage information and understand how many volumes are provisioned on a backend and the amount of space consumed, and so on.
- Maintain a mapping of the amount of volumes provisioned on available backends.
- Track performance. You can take a look at how long it takes for Trident to communicate to backends and perform operations.

By default, Trident's metrics are exposed on the target port `8001` at the `/metrics` endpoint. These metrics are **enabled by default** when Trident is installed. You can configure to consume Trident metrics over HTTPS on port `8444` as well.

What you'll need

- A Kubernetes cluster with Trident installed.
- A Prometheus instance. This can be a containerized Prometheus deployment or you can choose to run Prometheus as a native application.

Step 1: Define a Prometheus target

You should define a Prometheus target to gather the metrics and obtain information about the backends Trident manages, the volumes it creates, and so on. See Prometheus Operator documentation.

Step 2: Create a Prometheus ServiceMonitor

To consume the Trident metrics, you should create a Prometheus ServiceMonitor that watches the `trident-csi` service and listens on the `metrics` port. A sample ServiceMonitor looks like this:

```
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
```

This ServiceMonitor definition retrieves metrics returned by the `trident-csi` service and specifically looks for the `metrics` endpoint of the service. As a result, Prometheus is now configured to understand Trident's metrics.

In addition to metrics available directly from Trident, kubelet exposes many `kubelet_volume_*` metrics via it's own metrics endpoint. Kubelet can provide information about the volumes that are attached, and pods and other internal operations it handles. Refer to here.

Consume Trident metrics over HTTPS

To consume Trident metrics over HTTPS (port 8444), you must modify the ServiceMonitor definition to include TLS configuration. You also need to copy the `trident-csi` secret from the `trident` namespace to the namespace where Prometheus is running. You can do this using the following command:

```
kubectl get secret trident-csi -n trident -o yaml | sed 's/namespace:
trident/namespace: monitoring/' | kubectl apply -f -
```

A sample ServiceMonitor for HTTPS metrics looks like this:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - interval: 15s
      path: /metrics
      port: https-metrics
      scheme: https
      tlsConfig:
        ca:
          secret:
            key: caCert
            name: trident-csi
        cert:
          secret:
            key: clientCert
            name: trident-csi
        keySecret:
          key: clientKey
          name: trident-csi
        serverName: trident-csi
```

Trident support HTTPS metrics in all the installation methods: tridentctl, Helm chart, and Operator:

- If you are using the `tridentctl install` command, you can pass the `--https-metrics` flag to enable HTTPS metrics.
- If you are using the Helm chart, you can set the `httpsMetrics` parameter to enable HTTPS metrics.
- If you are using YAML files, you can add the `--https_metrics` flag to the `trident-main` container in the `trident-deployment.yaml` file.

Step 3: Query Trident metrics with PromQL

PromQL is good for creating expressions that return time-series or tabular data.

Here are some PromQL queries that you can use:

Get Trident health information

- **Percentage of HTTP 2XX responses from Trident**

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Percentage of REST responses from Trident via status code**

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Average duration in ms of operations performed by Trident**

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

Get Trident usage information

- **Average volume size**

```
trident_volume_allocated_bytes/trident_volume_count
```

- **Total volume space provisioned by each backend**

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

Get individual volume usage

This is enabled only if kubelet metrics are also gathered.

- **Percentage of used space for each volume**

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Learn about Trident AutoSupport telemetry

By default, Trident sends Prometheus metrics and basic backend information to NetApp on a daily cadence.

- To stop Trident from sending Prometheus metrics and basic backend information to NetApp, pass the `--silence-autosupport` flag during Trident installation.
- Trident can also send container logs to NetApp Support on-demand via `tridentctl send autosupport`. You will need to trigger Trident to upload it's logs. Before you submit logs, you should accept NetApp's
  privacy policy.
- Unless specified, Trident fetches the logs from the past 24 hours.
- You can specify the log retention time frame with the `--since` flag. For example: `tridentctl send autosupport --since=1h`. This information is collected and sent via a `trident-autosupport` container
  that is installed alongside Trident. You can obtain the container image at Trident AutoSupport.
- Trident AutoSupport does not gather or transmit Personally Identifiable Information (PII) or Personal Information. It comes with a EULA that is not applicable to the Trident container image itself. You can learn more about NetApp's commitment to data security and trust here.

An example payload sent by Trident looks like this:

```
---
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
```

- The AutoSupport messages are sent to NetApp's AutoSupport endpoint. If you are using a private registry to store container images, you can use the `--image-registry` flag.
- You can also configure proxy URLs by generating the installation YAML files. This can be done by using `tridentctl install --generate-custom-yaml` to create the YAML files and adding the `--proxy-url` argument for the `trident-autosupport` container in `trident-deployment.yaml`.

Disable Trident metrics

To **disable** metrics from being reported, you should generate custom YAMLs (using the `--generate-custom-yaml` flag) and edit them to remove the `--metrics` flag from being invoked for the `trident-main`

container.

Uninstall Trident

You should use the same method to uninstall Trident that you used to install Trident.

About this task

- If you need a fix for bugs observed after an upgrade, dependency issues, or an unsuccessful or incomplete upgrade, you should uninstall Trident and reinstall the earlier version using the specific instructions for that version. This is the only recommended way to *downgrade* to an earlier version.
- For easy upgrade and reinstallation, uninstalling Trident does not remove the CRDs or related objects created by Trident. If you need to completely remove Trident and all of its data, refer to Completely remove Trident and CRDs.

Before you begin

If you are decommissioning Kubernetes clusters, you must delete all applications that use volumes created by Trident prior to uninstalling. This ensures that PVCs are unpublished on Kubernetes nodes before they are deleted.

Determine the original installation method

You should use the same method to uninstall Trident that you used to install it. Before uninstalling, verify which version you used to originally install Trident.

1. Use `kubectl get pods -n trident` to examine the pods.
   - If there is no operator pod, Trident was installed using `tridentctl`.
   - If there is an operator pod, Trident was installed using the Trident operator either manually or using Helm.
2. If there is an operator pod, use `kubectl describe tproc trident` to determine if Trident was installed using Helm.
   - If there is a Helm label, Trident was installed using Helm.
   - If there is no Helm label, Trident was installed manually using the Trident operator.

Uninstall a Trident operator installation

You can uninstall a trident operator installation manually or using Helm.

Uninstall manual installation

If you installed Trident using the operator, you can uninstall it by doing one of the following:

1. **Edit `TridentOrchestrator` CR and set the uninstall flag**:

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

   When the `uninstall` flag is set to `true`, the Trident operator uninstalls Trident, but does not remove the TridentOrchestrator itself. You should clean up the TridentOrchestrator and create a new one if you want to

install Trident again.

2. **Delete `TridentOrchestrator`**: By removing the `TridentOrchestrator` CR that was used to deploy Trident, you instruct the operator to uninstall Trident. The operator processes the removal of `TridentOrchestrator` and proceeds to remove the Trident deployment and daemonset, deleting the Trident pods it had created as part of the installation.

```
kubectl delete -f deploy/<bundle.yaml> -n <namespace>
```

Uninstall Helm installation

If you installed Trident by using Helm, you can uninstall it by using `helm uninstall`.

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

Uninstall a `tridentctl` installation

Use the `uninstall` command in `tridentctl` to remove all of the resources associated with Trident except for the CRDs and related objects:

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

Trident for Docker

Prerequisites for deployment

You have to install and configure the necessary protocol prerequisites on your host before you can deploy Trident.

Verify the requirements

- Verify that your deployment meets all of the requirements.
- Verify that you have a supported version of Docker installed. If your Docker version is out of date, install or update it.

```
docker --version
```

- Verify that the protocol prerequisites are installed and configured on your host.

NFS tools

Install the NFS tools using the commands for your operating system.

RHEL 8+

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

Reboot your worker nodes after installing the NFS tools to prevent failure when attaching volumes to containers.

iSCSI tools

Install the iSCSI tools using the commands for your operating system.

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-
mapper-multipath
```

2. Check that iscsi-initiator-utils version is 6.2.0.874-2.el7 or later:

```
rpm -q iscsi-initiator-utils
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

Ensure `etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `iscsid` and `multipathd` are running:

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

6. Enable and start `iscsi`:

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. Install the following system packages:

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

2. Check that open-iscsi version is 2.0.874-5ubuntu2.10 or later (for bionic) or 2.0.874-7.1ubuntu6.1 or later (for focal):

```
dpkg -l open-iscsi
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
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
```

> Ensure `etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `open-iscsi` and `multipath-tools` are enabled and running:

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

NVMe tools

Install the NVMe tools using the commands for your operating system.

- NVMe requires RHEL 9 or later.
- If the kernel version of your Kubernetes node is too old or if the NVMe package is not available for your kernel version, you might have to update the kernel version of your node to one with the NVMe package.

RHEL 9

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

Ubuntu

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

FC tools

Install the FC tools using the commands for your operating system.

- When using worker nodes that run RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) with FC PVs, specify the `discard` mountOption in the StorageClass to perform inline space reclamation. Refer to Red Hat documentation.

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

Ensure `etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipathd` is running:

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

Ubuntu

1. Install the following system packages:

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

2. Enable multipathing:

```
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
```

Ensure `etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipath-tools` is enabled and running:

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

Deploy Trident

Trident for Docker provides direct integration with the Docker ecosystem for NetApp storage platforms. It supports the provisioning and management of storage resources from the storage platform to Docker hosts, with a framework for adding additional platforms in the future.

Multiple instances of Trident can run concurrently on the same host. This allows simultaneous connections to multiple storage systems and storage types, with the ablity to customize the storage used for the Docker volumes.

What you'll need

See the prerequisites for deployment. After you ensure the prerequisites are met, you are ready to deploy Trident.

Docker managed plugin method (version 1.13/17.03 and later)

Before you begin

If you have used Trident pre Docker 1.13/17.03 in the traditional daemon method, ensure that you stop the Trident process and restart your Docker daemon before using the managed plugin method.

1. Stop all running instances:

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

2. Restart Docker.

```
systemctl restart docker
```

3. Ensure that you have Docker Engine 17.03 (new 1.13) or later installed.

```
docker --version
```

If your version is out of date, install or update your installation.

Steps

1. Create a configuration file and specify the options as follows:
   - `config`: The default filename is `config.json`, however you can use any name you choose by specifying the `config` option with the filename. The configuration file must be located in the `/etc/netappdvp` directory on the host system.
   - `log-level`: Specify the logging level (`debug`, `info`, `warn`, `error`, `fatal`). The default is `info`.
   - `debug`: Specify whether debug logging is enabled. Default is false. Overrides log-level if true.

a. Create a location for the configuration file:

```
sudo mkdir -p /etc/netappdvp
```

b. Create the configuration file:

```
cat << EOF > /etc/netappdvp/config.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. Start Trident using the managed plugin system. Replace `<version>` with the plugin version (xxx.xx.x) you are using.

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. Begin using Trident to consume storage from the configured system.
   a. Create a volume named "firstVolume":

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

   b. Create a default volume when the container starts:

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

   c. Remove the volume "firstVolume":

```
docker volume rm firstVolume
```

Traditional method (version 1.12 or earlier)

Before you begin

1. Ensure that you have Docker version 1.10 or later.

   ```
   docker --version
   ```

   If your version is out of date, update your installation.

   ```
   curl -fsSL https://get.docker.com/ | sh
   ```

   Or, follow the instructions for your distribution.

2. Ensure that NFS and/or iSCSI is configured for your system.

Steps

1. Install and configure the NetApp Docker Volume Plugin:
   a. Download and unpack the application:

      ```
      wget
      https://github.com/NetApp/trident/releases/download/10.0/trident-
      installer-26.02.0.tar.gz
      tar zxf trident-installer-26.02.0.tar.gz
      ```

   b. Move to a location in the bin path:

      ```
      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. Create a location for the configuration file:

      ```
      sudo mkdir -p /etc/netappdvp
      ```

   d. Create the configuration file:

      ```
      cat << EOF > /etc/netappdvp/ontap-nas.json
      ```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. After placing the binary and creating the configuration file, start the Trident daemon using the desired configuration file.

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

> Unless specified, the default name for the volume driver is "netapp".

After the daemon is started, you can create and manage volumes by using the Docker CLI interface.

3. Create a volume:

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

4. Provision a Docker volume when starting a container:

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

5. Remove a Docker volume:

```
docker volume rm trident_1
```

```
docker volume rm trident_2
```

Start Trident at system startup

A sample unit file for systemd based systems can be found at `contrib/trident.service.example` in the Git repo. To use the file with RHEL, do the following:

1. Copy the file to the correct location.

   You should use unique names for the unit files if you have more than one instance running.

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

2. Edit the file, change the description (line 2) to match the driver name and the configuration file path (line 9) to reflect your environment.
3. Reload systemd for it to ingest changes:

   ```
   systemctl daemon-reload
   ```

4. Enable the service.

   This name varies depending on what you named the file in the `/usr/lib/systemd/system` directory.

   ```
   systemctl enable trident
   ```

5. Start the service.

   ```
   systemctl start trident
   ```

6. View the status.

   ```
   systemctl status trident
   ```

Any time you modify the unit file, run the `systemctl daemon-reload` command for it to be aware of the changes.

Upgrade or uninstall Trident

You can safely upgrade Trident for Docker without any impact to volumes that are in use. During the upgrade process there will be a brief period where `docker volume` commands directed at the plugin will not succeed, and applications will be unable to mount volumes until the plugin is running again. Under most circumstances, this is a matter of seconds.

Upgrade

Perform the steps below to upgrade Trident for Docker.

Steps

1. List the existing volumes:

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

2. Disable the plugin:

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. Upgrade the plugin:

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```

> The 18.01 release of Trident replaces the nDVP. You should upgrade directly from the `netapp/ndvp-plugin` image to the `netapp/trident-plugin` image.

4. Enable the plugin:

```
docker plugin enable netapp:latest
```

5. Verify that the plugin is enabled:

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. Verify that the volumes are visible:

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

> If you are upgrading from an old version of Trident (pre-20.10) to Trident 20.10 or later, you might run into an error. For more information, refer to Known Issues. If you run into the error, you should first disable the plugin, then remove the plugin, and then install the required Trident version by passing an extra config parameter: `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

Uninstall

Perform the steps below to uninstall Trident for Docker.

Steps

1. Remove any volumes that the plugin created.
2. Disable the plugin:

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. Remove the plugin:

```
docker plugin rm netapp:latest
```

Work with volumes

You can easily create, clone, and remove volumes using the standard `docker volume` commands with the Trident driver name specified when needed.

Create a volume

- Create a volume with a driver using the default name:

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

- Create a volume with a specific Trident instance:

```
docker volume create -d ntap_bronze --name bronzeVolume
```

If you do not specify any options, the defaults for the driver are used.

- Override the default volume size. See the following example to create a 20 GiB volume with a driver:

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

Volume sizes are expressed as strings containing an integer value with optional units (example: 10G, 20GB, 3TiB). If no units are specified, the default is G. Size units can be expressed either as powers of 2 (B, KiB, MiB, GiB, TiB) or powers of 10 (B, KB, MB, GB, TB). Shorthand units use powers of 2 (G = GiB, T = TiB, …).

Remove a volume

- Remove the volume just like any other Docker volume:

```
docker volume rm firstVolume
```

When using the `solidfire-san` driver, the above example deletes and purges the volume.

Perform the steps below to upgrade Trident for Docker.

Clone a volume

When using the `ontap-nas`, `ontap-san`, and `solidfire-san` storage drivers, Trident can clone volumes. When using the `ontap-nas-flexgroup` or `ontap-nas-economy` drivers, cloning is not supported. Creating a new volume from an existing volume will result in a new snapshot being created.

- Inspect the volume to enumerate snapshots:

```
docker volume inspect <volume_name>
```

- Create a new volume from an existing volume. This will result in a new snapshot being created:

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume>
```

- Create a new volume from an existing snapshot on a volume. This will not create a new snapshot:

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

Example

```
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
```

Access externally created volumes

You can access externally created block devices (or their clones) by containers using Trident **only** if they have no partitions and if their filesystem is supported by Trident (for example: an `ext4`-formatted `/dev/sdc1` will not be accessible via Trident).

Driver-specific volume options

Each storage driver has a different set of options, which you can specify at volume creation time to customize the outcome. See below for options that apply to your configured storage system.

Using these options during the volume create operation is simple. Provide the option and the value using the `-o` operator during the CLI operation. These override any equivalent values from the JSON configuration file.

ONTAP volume options

Volume create options for both NFS, iSCSI, and FC include the following:

Option	Description
`size`	The size of the volume, defaults to 1 GiB.
`spaceReserve`	Thin or thick provision the volume, defaults to thin. Valid values are `none` (thin provisioned) and `volume` (thick provisioned).
`snapshotPolicy`	This will set the snapshot policy to the desired value. The default is `none`, meaning no snapshots will automatically be created for the volume. Unless modified by your storage administrator, a policy named "default" exists on all ONTAP systems which creates and retains six hourly, two daily, and two weekly snapshots. The data preserved in a snapshot can be recovered by browsing to the `.snapshot` directory in any directory in the volume.
`snapshotReserve`	This will set the snapshot reserve to the desired percentage. The default is no value, meaning ONTAP will select the snapshotReserve (usually 5%) if you have selected a snapshotPolicy, or 0% if the snapshotPolicy is none. You can set the default snapshotReserve value in the config file for all ONTAP backends, and you can use it as a volume creation option for all ONTAP backends except ontap-nas-economy.
`splitOnClone`	When cloning a volume, this will cause ONTAP to immediately split the clone from its parent. The default is `false`. Some use cases for cloning volumes are best served by splitting the clone from its parent immediately upon creation, because there is unlikely to be any opportunity for storage efficiencies. For example, cloning an empty database can offer large time savings but little storage savings, so it’s best to split the clone immediately.

Option	Description
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.
`tieringPolicy`	Sets the tiering policy to be used for the volume. This decides whether data is moved to the cloud tier when it becomes inactive (cold).

The following additional options are for NFS **only**:

Option	Description
`unixPermissions`	This controls the permission set for the volume itself. By default the permissions will be set to `---rwxr-xr-x`, or in numerical notation 0755, and `root` will be the owner. Either the text or numerical format will work.
`snapshotDir`	Setting this to `true` will make the `.snapshot` directory visible to clients accessing the volume. The default value is `false`, meaning that visibility of the `.snapshot` directory is disabled by default. Some images, for example the official MySQL image, don't function as expected when the `.snapshot` directory is visible.
`exportPolicy`	Sets the export policy to be used for the volume. The default is `default`.
`securityStyle`	Sets the security style to be used for access to the volume. The default is `unix`. Valid values are `unix` and `mixed`.

The following additional options are for iSCSI **only**:

Option	Description
`fileSystemType`	Sets the file system used to format iSCSI volumes. The default is `ext4`. Valid values are `ext3`, `ext4`, and `xfs`.

Option	Description
`spaceAllocation`	Setting this to `false` will turn off the LUN's space-allocation feature. The default value is `true`, meaning ONTAP notifies the host when the volume has run out of space and the LUN in the volume cannot accept writes. This option also enables ONTAP to reclaim space automatically when your host deletes data.

Examples

See the examples below:

- Create a 10 GiB volume:

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

- Create a 100 GiB volume with snapshots:

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- Create a volume which has the setUID bit enabled:

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

The minimum volume size is 20 MiB.

If the snapshot reserve is not specified and the snapshot policy is `none`, Trident use a snapshot reserve of 0%.

- Create a volume with no snapshot policy and no snapshot reserve:

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- Create a volume with no snapshot policy and a custom snapshot reserve of 10%:

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- Create a volume with a snapshot policy and a custom snapshot reserve of 10%:

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- Create a volume with a snapshot policy, and accept the ONTAP's default snapshot reserve (usually 5%):

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element software volume options

The Element software options expose the size and quality of service (QoS) policies associated with the volume. When the volume is created, the QoS policy associated with it is specified using the `-o type=service_level` nomenclature.

The first step to defining a QoS service level with the Element driver is to create at least one type and specify the minimum, maximum, and burst IOPS associated with a name in the configuration file.

Other Element software volume create options include the following:

Option	Description
`size`	The size of the volume, defaults to 1 GiB or config entry … "defaults": {"size": "5G"}.
`blocksize`	Use either 512 or 4096, defaults to 512 or config entry DefaultBlockSize.

Example

See the following sample configuration file with QoS definitions:

```
{
  "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
      }
    }
  ]
}
```

In the above configuration, we have three policy definitions: Bronze, Silver, and Gold. These names are arbitrary.

- Create a 10 GiB Gold volume:

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- Create a 100 GiB Bronze volume:

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

Collect logs

You can collect logs for help with troubleshooting. The method you use to collect the logs varies based on how you are running the Docker plugin.

Collect logs for troubleshooting

Steps

1. If you are running Trident using the recommended managed plugin method (i.e., using `docker plugin` commands), view them as follows:

```
docker plugin ls
```

```
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
journalctl -u docker | grep 4fb97d2b956b
```

   The standard logging level should allow you to diagnose most issues. If you find that's not enough, you can enable debug logging.

2. To enable debug logging, install the plugin with debug logging enabled:

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

   Or, enable debug logging when the plugin is already installed:

```
docker plugin disable <plugin>
```

```
docker plugin set <plugin> debug=true
```

```
docker plugin enable <plugin>
```

3. If you are running the binary itself on the host, logs are available in the host's `/var/log/netappdvp` directory. To enable debug logging, specify `-debug` when you run the plugin.

General troubleshooting tips

- The most common problem new users run into is a misconfiguration that prevents the plugin from

initializing. When this happens you will likely see a message such as this when you try to install or enable the plugin:

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

This means that the plugin failed to start. Luckily, the plugin has been built with a comprehensive logging capability that should help you diagnose most of the issues you are likely to come across.

- If there are problems with mounting a PV to a container, ensure that `rpcbind` is installed and running. Use the required package manager for the host OS and check if `rpcbind` is running. You can check the status of the rpcbind service by running a `systemctl status rpcbind` or its equivalent.

Manage multiple Trident instances

Multiple instances of Trident are needed when you desire to have multiple storage configurations available simultaneously. The key to multiple instances is to give them different names using the `--alias` option with the containerized plugin, or `--volume-driver` option when instantiating Trident on the host.

Steps for Docker managed plugin (version 1.13/17.03 or later)

1. Launch the first instance specifying an alias and configuration file.

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. Launch the second instance, specifying a different alias and configuration file.

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. Create volumes specifying the alias as the driver name.

   For example, for gold volume:

```
docker volume create -d gold --name ntapGold
```

   For example, for silver volume:

```
docker volume create -d silver --name ntapSilver
```

Steps for traditional (version 1.12 or earlier)

1. Launch the plugin with an NFS configuration using a custom driver ID:

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. Launch the plugin with an iSCSI configuration using a custom driver ID:

```
sudo trident --volume-driver=netapp-san --config=/path/to/config
-iscsi.json
```

3. Provision Docker volumes for each driver instance:

   For example, for NFS:

```
docker volume create -d netapp-nas --name my_nfs_vol
```

   For example, for iSCSI:

```
docker volume create -d netapp-san --name my_iscsi_vol
```

Storage configuration options

See the configuration options available for your Trident configurations.

Global configuration options

These configuration options apply to all Trident configurations, regardless of the storage platform being used.

Option	Description	Example
`version`	Config file version number	`1`
`storageDriverName`	Name of storage driver	`ontap-nas`, `ontap-san`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `solidfire-san`
`storagePrefix`	Optional prefix for volume names. Default: `netappdvp_`.	`staging_`

Option	Description	Example
`limitVolumeSize`	Optional restriction on volume sizes. Default: "" (not enforced)	`10g`

Do not use `storagePrefix` (including the default) for Element backends. By default, the `solidfire-san` driver will ignore this setting and not use a prefix. NetApp recommends using either a specific tenantID for Docker volume mapping or using the attribute data which is populated with the Docker version, driver info, and raw name from Docker in cases where any name munging may have been used.

Default options are available to avoid having to specify them on every volume you create. The `size` option is available for all the controller types. See the ONTAP configuration section for an example of how to set the default volume size.

Option	Description	Example
`size`	Optional default size for new volumes. Default: `1G`	`10G`

ONTAP configuration

In addition to the global configuration values above, when using ONTAP, the following top-level options are available.

Option	Description	Example
`managementLIF`	IP address of ONTAP management LIF. You can specify a fully-qualified domain name (FQDN).	`10.0.0.1`

Option	Description	Example
`dataLIF`	IP address of protocol LIF. **ONTAP NAS drivers**: NetApp recommends specifying `dataLIF`. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. **ONTAP SAN drivers**: Do not specify for iSCSI or FC. Trident uses ONTAP Selective LUN Map to discover the iSCSI or FC LIFs needed to establish a multi path session. A warning is generated if `dataLIF` is explicitly defined.	`10.0.0.2`
`svm`	Storage virtual machine to use (required, if management LIF is a cluster LIF)	`svm_nfs`
`username`	Username to connect to the storage device	`vsadmin`
`password`	Password to connect to the storage device	`secret`
`aggregate`	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the `ontap-nas-flexgroup` driver, this option is ignored. All aggregates assigned to the SVM are used to provision a FlexGroup volume.	`aggr1`
`limitAggregateUsage`	Optional, fail provisioning if usage is above this percentage	`75%`
`nfsMountOptions`	Fine grained control of NFS mount options; defaults to "-o nfsvers=3". **Available only for the `ontap-nas` and `ontap-nas-economy` drivers**. See NFS host configuration information here.	`-o nfsvers=4`

Option	Description	Example
`igroupName`	Trident creates and manages per-node `igroups` as `netappdvp`. This value cannot be changed or omitted. **Available only for the `ontap-san` driver**.	`netappdvp`
`limitVolumeSize`	Maximum requestable volume size.	`300g`
`qtreesPerFlexvol`	Maximum qtrees per FlexVol, must be in range [50, 300], default is 200. **For the `ontap-nas-economy` driver, this option allows customizing the maximum number of qtrees per FlexVol**.	`300`
`sanType`	**Supported for `ontap-san` driver only.** Use to select `iscsi` for iSCSI, `nvme` for NVMe/TCP or `fcp` for SCSI over Fibre Channel (FC).	`iscsi` if blank
`limitVolumePoolSize`	**Supported for `ontap-san-economy` and `ontap-san-economy` drivers only.** Limits FlexVol sizes in ONTAP ontap-nas-economy and ontap-SAN-economy drivers.	`300g`

Default options are available to avoid having to specify them on every volume you create:

Option	Description	Example
`spaceReserve`	Space reservation mode; `none` (thin provisioned) or `volume` (thick)	`none`
`snapshotPolicy`	Snapshot policy to use, default is `none`	`none`
`snapshotReserve`	Snapshot reserve percentage, default is "" to accept the ONTAP default	`10`

Option	Description	Example
`splitOnClone`	Split a clone from its parent upon creation, defaults to `false`	`false`
`encryption`	Enables NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	true
`unixPermissions`	NAS option for provisioned NFS volumes, defaults to `777`	`777`
`snapshotDir`	NAS option for access to the `.snapshot` directory.	Enables or disables access to the `.snapshot` directory. Valid values: `true`, `false`. Default behavior might vary by ONTAP and Trident version. Set explicitly to ensure consistent behavior.
`exportPolicy`	NAS option for the NFS export policy to use, defaults to `default`	`default`
`securityStyle`	NAS option for access to the provisioned NFS volume. NFS supports `mixed` and `unix` security styles. The default is `unix`.	`unix`
`fileSystemType`	SAN option to select the file system type, defaults to `ext4`	`xfs`
`tieringPolicy`	Tiering policy to use, default is `none`.	`none`
`skipRecoveryQueue`	During volume deletion, bypass the recovery queue in storage and delete the volume immediately.	``

Scaling options

The `ontap-nas` and `ontap-san` drivers create an ONTAP FlexVol for each Docker volume. ONTAP supports up to 1000 FlexVols per cluster node with a cluster maximum of 12,000 FlexVol volumes If your Docker volume requirements fit within that limitation, the `ontap-nas` driver is the preferred NAS solution due to the additional

features offered by FlexVols, such as Docker-volume-granular snapshots and cloning.

If you need more Docker volumes than can be accommodated by the FlexVol limits, choose the `ontap-nas-economy` or the `ontap-san-economy` driver.

The `ontap-nas-economy` driver creates Docker volumes as ONTAP Qtrees within a pool of automatically managed FlexVol volumes. Qtrees offer far greater scaling, up to 100,000 per cluster node and 2,400,000 per cluster, at the expense of some features. The `ontap-nas-economy` driver does not support Docker-volume-granular snapshots or cloning.

The `ontap-nas-economy` driver is not currently supported in Docker Swarm, because Docker Swarm does not orchestrate volume creation across multiple nodes.

The `ontap-san-economy` driver creates Docker volumes as ONTAP LUNs within a shared pool of automatically managed FlexVol volumes. This way, each FlexVol is not restricted to only one LUN and it offers better scalability for SAN workloads. Depending on the storage array, ONTAP supports up to 16384 LUNs per cluster. Because the volumes are LUNs underneath, this driver supports Docker-volume-granular snapshots and cloning.

Choose the `ontap-nas-flexgroup` driver to increase parallelism to a single volume that can grow into the petabyte range with billions of files. Some ideal use cases for FlexGroups include AI/ML/DL, big data and analytics, software builds, streaming, file repositories, and so on. Trident uses all aggregates assigned to an SVM when provisioning a FlexGroup volume. FlexGroup support in Trident also has the following considerations:

- Requires ONTAP version 9.2 or greater.
- As of this writing, FlexGroups only support NFS v3.
- Recommended to enable the 64-bit NFSv3 identifiers for the SVM.
- The minimum recommended FlexGroup member/volume size is 100 GiB.
- Cloning is not supported for FlexGroup volumes.

For information about FlexGroups and workloads that are appropriate for FlexGroups refer to the NetApp FlexGroup volume Best Practices and Implementation Guide.

To get advanced features and large scale in the same environment, you can run multiple instances of the Docker Volume Plugin, with one using `ontap-nas` and another using `ontap-nas-economy`.

Custom ONTAP role for Trident

You can create an ONTAP cluster role with minimum privileges so that you do not have to use the ONTAP admin role to perform operations in Trident. When you include the username in a Trident backend configuration, Trident uses the ONTAP cluster role you created to perform the operations.

Refer to Trident custom-role generator for more information about creating Trident custom roles.

Using ONTAP CLI

1. Create a new role using the following command:

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. Create a usename for the Trident user:

```
security login create -username <user_name\> -application ontapi
-authmethod password -role <name_of_role_in_step_1\> -vserver <svm_name\>
-comment "user_description"
security login create -username <user_name\> -application http -authmethod
password -role <name_of_role_in_step_1\> -vserver <svm_name\> -comment
"user_description"
```

3. Map the role to the user:

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

Using System Manager

Perform the following steps in ONTAP System Manager:

1. **Create a custom role**:
   a. To create a custom role at the cluster-level, select **Cluster > Settings**.

      (Or) To create a custom role at the SVM level, select **Storage > Storage VMs > `required SVM`> Settings > Users and Roles**.

   b. Select the arrow icon (→) next to **Users and Roles**.
   c. Select **+Add** under **Roles**.
   d. Define the rules for the role and click **Save**.
2. **Map the role to the Trident user**:
   + Perform the following steps on the **Users and Roles** page:
   a. Select Add icon **+** under **Users**.
   b. Select the required username, and select a role in the drop-down menu for **Role**.
   c. Click **Save**.

Refer to the following pages for more information:

- Custom roles for administration of ONTAP or Define custom roles
- Work with roles and users

Example ONTAP configuration files

NFS example for `ontap-nas` driver

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

NFS example for `ontap-nas-flexgroup` driver

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-flexgroup",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

NFS example for `ontap-nas-economy` driver

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
```

iSCSI example for `ontap-san` driver

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

NFS example for `ontap-san-economy` driver

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

NVMe/TCP example for `ontap-san` driver

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

SCSI over FC example for `ontap-san` driver

```
{
  "version": 1,
  "backendName": "ontap-san-backend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "sanType": "fcp",
  "svm": "trident_svm",
  "username": "vsadmin",
  "password": "password",
  "useREST": true
}
```

Element software configuration

In addition to the global configuration values, when using Element software (NetApp HCI/SolidFire), these options are available.

Option	Description	Example
`Endpoint`	https://<login>:<password>@<mvip>/json-rpc/<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IP address and port	10.0.0.7:3260
`TenantName`	SolidFireF Tenant to use (created if not found)	`docker`

Option	Description	Example
`InitiatorIFace`	Specify interface when restricting iSCSI traffic to non-default interface	`default`
`Types`	QoS specifications	See example below
`LegacyNamePrefix`	Prefix for upgraded Trident installs. If you used a version of Trident prior to 1.3.2 and perform an upgrade with existing volumes, you'll need to set this value to access your old volumes that were mapped via the volume-name method.	`netappdvp-`

The `solidfire-san` driver does not support Docker Swarm.

Example Element software configuration file

```
{
  "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
      }
    }
  ]
}
```

Known issues and limitations

Find information about known issues and limitations when using Trident with Docker.

Upgrading Trident Docker Volume Plugin to 20.10 and later from older versions results in upgrade failure with the no such file or directory error.

Workaround

1. Disable the plugin.

```
docker plugin disable -f netapp:latest
```

2. Remove the plugin.

```
docker plugin rm -f netapp:latest
```

3. Reinstall the plugin by providing the extra `config` parameter.

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

Volume names must be a minimum of 2 characters in length.

This is a Docker client limitation. The client will interpret a single character name as being a Windows path. See bug 25773.

Docker Swarm has certain behaviors that prevent Trident from supporting it with every storage and driver combination.

- Docker Swarm presently makes use of volume name instead of volume ID as its unique volume identifier.
- Volume requests are simultaneously sent to each node in a Swarm cluster.
- Volume plugins (including Trident) must run independently on each node in a Swarm cluster.
  Due to the way ONTAP works and how the `ontap-nas` and `ontap-san` drivers function, they are the only ones that happen to be able to operate within these limitations.

The rest of the drivers are subject to issues like race conditions that can result in the creation of a large number of volumes for a single request without a clear "winner"; for example, Element has a feature that allows volumes to have the same name but different IDs.

NetApp has provided feedback to the Docker team, but does not have any indication of future recourse.

If a FlexGroup is being provisioned, ONTAP does not provision a second FlexGroup if the second FlexGroup has one or more aggregates in common with the FlexGroup being provisioned.

Best practices and recommendations

Deployment

Use the recommendations listed here when you deploy Trident.

Deploy to a dedicated namespace

Namespaces provide administrative separation between different applications and are a barrier for resource sharing. For example, a PVC from one namespace cannot be consumed from another. Trident provides PV resources to all the namespaces in the Kubernetes cluster and consequently leverages a service account which has elevated privileges.

Additionally, access to the Trident pod might enable a user to access storage system credentials and other sensitive information. It is important to ensure that application users and management applications do not have the ability to access the Trident object definitions or the pods themselves.

Use quotas and range limits to control storage consumption

Kubernetes has two features which, when combined, provide a powerful mechanism for limiting the resource consumption by applications. The storage quota mechanism enables the administrator to implement global, and storage class specific, capacity and object count consumption limits on a per-namespace basis. Further, using a range limit ensures that the PVC requests are within both a minimum and maximum value before the request is forwarded to the provisioner.

These values are defined on a per-namespace basis, which means that each namespace should have values defined which fall in line with their resource requirements. See here for information about how to leverage quotas.

Storage configuration

Each storage platform in the NetApp portfolio has unique capabilities that benefit applications, containerized or not.

Platform overview

Trident works with ONTAP and Element. There is not one platform which is better suited for all applications and scenarios than another, however, the needs of the application and the team administering the device should be taken into account when choosing a platform.

You should follow the baseline best practices for the host operating system with the protocol that you are leveraging. Optionally, you might want to consider incorporating application best practices, when available, with backend, storage class, and PVC settings to optimize storage for specific applications.

ONTAP and Cloud Volumes ONTAP best practices

Learn the best practices for configuring ONTAP and Cloud Volumes ONTAP for Trident.

The following recommendations are guidelines for configuring ONTAP for containerized workloads, which consume volumes that are dynamically provisioned by Trident. Each should be considered and evaluated for appropriateness in your environment.

Use SVM(s) dedicated to Trident

Storage Virtual Machines (SVMs) provide isolation and administrative separation between tenants on an ONTAP system. Dedicating an SVM to applications enables the delegation of privileges and enables applying best practices for limiting resource consumption.

There are several options available for the management of the SVM:

- Provide the cluster management interface in the backend configuration, along with appropriate credentials, and specify the SVM name.
- Create a dedicated management interface for the SVM by using ONTAP System Manager or the CLI.
- Share the management role with an NFS data interface.

In each case, the interface should be in DNS, and the DNS name should be used when configuring Trident. This helps to facilitate some DR scenarios, for example, SVM-DR without the use of network identity retention.

There is no preference between having a dedicated or shared management LIF for the SVM, however, you should ensure that your network security policies align with the approach you choose. Regardless, the management LIF should be accessible via DNS to facilitate maximum flexibility should SVM-DR be used in conjunction with Trident.

Limit the maximum volume count

ONTAP storage systems have a maximum volume count, which varies based on the software version and hardware platform. Refer to NetApp Hardware Universe for your specific platform and ONTAP version to determine the exact limits. When the volume count is exhausted, provisioning operations fail not only for Trident, but for all the storage requests.

Trident's `ontap-nas` and `ontap-san` drivers provision a FlexVolume for each Kubernetes Persistent Volume (PV) that is created. The `ontap-nas-economy` driver creates approximately one FlexVolume for every 200 PVs (configurable between 50 and 300). The `ontap-san-economy` driver creates approximately one FlexVolume for every 100 PVs (configurable between 50 and 200). To prevent Trident from consuming all the available volumes on the storage system, you should set a limit on the SVM. You can do this from the command line:

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

The value for `max-volumes` varies based on several criteria specific to your environment:

- The number of existing volumes in the ONTAP cluster
- The number of volumes you expect to provision outside of Trident for other applications
- The number of persistent volumes expected to be consumed by Kubernetes applications

The `max-volumes` value is the total volumes provisioned across all the nodes in the ONTAP cluster, and not on an individual ONTAP node. As a result, you might encounter some conditions where an ONTAP cluster node might have far more or less Trident provisioned volumes than another node.

For example, a two-node ONTAP cluster has the ability to host a maximum of 2000 FlexVol volumes. Having the maximum volume count set to 1250 appears very reasonable. However, if only aggregates from one node are assigned to the SVM, or the aggregates assigned from one node are unable to be provisioned against (for example, due to capacity), then the other node becomes the target for all Trident provisioned volumes. This

means that the volume limit might be reached for that node before the `max-volumes` value is reached, resulting in impacting both Trident and other volume operations that use that node. **You can avoid this situation by ensuring that aggregates from each node in the cluster are assigned to the SVM used by Trident in equal numbers.**

Clone a volume

NetApp Trident supports cloning volumes when using the `ontap-nas`, `ontap-san`, and `solidfire-san` storage drivers. When using the `ontap-nas-flexgroup` or `ontap-nas-economy` drivers, cloning is not supported. Creating a new volume from an existing volume will result in a new snapshot being created.

Avoid cloning a PVC that is associated with a different StorageClass. Perform cloning operations within the same StorageClass to ensure compatibility and prevent unexpected behavior.

Limit the maximum size of volumes created by Trident

To configure the maximum size for volumes that can be created by Trident, use the `limitVolumeSize` parameter in your `backend.json` definition.

In addition to controlling the volume size at the storage array, you should also leverage Kubernetes capabilities.

Limit the maximum size of FlexVols created by Trident

To configure the maximum size for FlexVols used as pools for ontap-san-economy and ontap-nas-economy drivers, use the `limitVolumePoolSize` parameter in your `backend.json` definition.

Configure Trident to use bidirectional CHAP

You can specify the CHAP initiator and target usernames and passwords in your backend definition and have Trident enable CHAP on the SVM. Using the `useCHAP` parameter in your backend configuration, Trident authenticates iSCSI connections for ONTAP backends with CHAP.

Create and use an SVM QoS policy

Leveraging an ONTAP QoS policy, applied to the SVM, limits the number of IOPS consumable by the Trident provisioned volumes. This helps to prevent a bully or out-of-control container from affecting workloads outside of the Trident SVM.

You can create a QoS policy for the SVM in a few steps. See the documentation for your version of ONTAP for the most accurate information. The example below creates a QoS policy that limits the total IOPS available to the SVM to 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>
```

Additionally, if your version of ONTAP supports it, you can consider using a QoS minimum to guarantee an amount of throughput to containerized workloads. Adaptive QoS is not compatible with an SVM level policy.

The number of IOPS dedicated to the containerized workloads depends on many aspects. Among other things, these include:

- Other workloads using the storage array. If there are other workloads, not related to the Kubernetes deployment, utilizing the storage resources, care should be taken to ensure that those workloads are not accidentally adversely impacted.
- Expected workloads running in containers. If workloads which have high IOPS requirements will be running in containers, a low QoS policy results in a bad experience.

It's important to remember that a QoS policy assigned at the SVM level results in all the volumes provisioned to the SVM sharing the same IOPS pool. If one, or a small number, of the containerized applications have a high IOPS requirement, it could become a bully to the other containerized workloads. If this is the case, you might want to consider using external automation to assign per-volume QoS policies.

You should assign the QoS policy group to the SVM **only** if your ONTAP version is earlier than 9.8.

Create QoS policy groups for Trident

Quality of service (QoS) guarantees that performance of critical workloads is not degraded by competing workloads. ONTAP QoS policy groups provide QoS options for volumes, and enable users to define the throughput ceiling for one or more workloads. For more information about QoS, refer to Guaranteeing throughput with QoS.
You can specify QoS policy groups in the backend or in a storage pool, and they are applied to each volume created in that pool or backend.

ONTAP has two kinds of QoS policy groups: traditional and adaptive. Traditional policy groups provide a flat maximum (or minimum, in later versions) throughput in IOPS. Adaptive QoS automatically scales the throughput to workload size, maintaining the ratio of IOPS to TBs|GBs as the size of the workload changes. This provides a significant advantage when you are managing hundreds or thousands of workloads in a large deployment.

Consider the following when you create QoS policy groups:

- You should set the `qosPolicy` key in the `defaults` block of the backend configuration. See the following backend configuration example:

```
---
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
```

- You should apply the policy groups per volume, so that each volume gets the entire throughput as specified by the policy group. Shared policy groups are not supported.

For more information about QoS policy groups, refer to ONTAP command reference.

Limit storage resource access to Kubernetes cluster members

Limiting access to the NFS volumes, iSCSI LUNs, and FC LUNs created by Trident is a critical component of the security posture for your Kubernetes deployment. Doing so prevents hosts that are not a part of the Kubernetes cluster from accessing the volumes and potentially modifying data unexpectedly.

It's important to understand that namespaces are the logical boundary for resources in Kubernetes. The assumption is that resources in the same namespace are able to be shared, however, importantly, there is no cross-namespace capability. This means that even though PVs are global objects, when bound to a PVC they are only accessible by pods which are in the same namespace. **It is critical to ensure that namespaces are used to provide separation when appropriate.**

The primary concern for most organizations with regard to data security in a Kubernetes context is that a process in a container can access storage mounted to the host, but which is not intended for the container. Namespaces are designed to prevent this type of compromise. However, there is one exception: privileged containers.

A privileged container is one that is run with substantially more host-level permissions than normal. These are not denied by default, so ensure that you disable the capability by using pod security policies.

For volumes where access is desired from both Kubernetes and external hosts, the storage should be managed in a traditional manner, with the PV introduced by the administrator and not managed by Trident. This ensures that the storage volume is destroyed only when both the Kubernetes and external hosts have disconnected and are no longer using the volume. Additionally, a custom export policy can be applied, which enables access from the Kubernetes cluster nodes and targeted servers outside of the Kubernetes cluster.

For deployments which have dedicated infrastructure nodes (for example, OpenShift) or other nodes which are unable to schedule user applications, separate export policies should be used to further limit access to storage resources. This includes creating an export policy for services which are deployed to those infrastructure nodes (for example, the OpenShift Metrics and Logging services), and standard applications which are deployed to non-infrastructure nodes.

Use a dedicated export policy

You should ensure that an export policy exists for each backend that only allows access to the nodes present in the Kubernetes cluster. Trident can automatically create and manage export policies. This way, Trident limits access to the volumes it provisions to the nodes in the Kubernetes cluster and simplifies the addition/deletion of nodes.

Alternatively, you can also create an export policy manually and populate it with one or more export rules that process each node access request:

- Use the `vserver export-policy create` ONTAP CLI command to create the export policy.
- Add rules to the export policy by using the `vserver export-policy rule create` ONTAP CLI command.

Running these commands enables you to restrict which Kubernetes nodes have access to the data.

Disable `showmount` for the application SVM

The `showmount` feature enables an NFS client to query the SVM for a list of available NFS exports. A pod deployed to the Kubernetes cluster can issue the `showmount -e` command against the and receive a list of available mounts, including those which it does not have access to. While this, by itself, is not a security compromise, it does provide unnecessary information potentially aiding an unauthorized user with connecting to an NFS export.

You should disable `showmount` by using the SVM-level ONTAP CLI command:

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire best practices

Learn the best practices for configuring SolidFire storage for Trident.

Create Solidfire Account

Each SolidFire account represents a unique volume owner and receives its own set of Challenge-Handshake Authentication Protocol (CHAP) credentials. You can access volumes assigned to an account either by using the account name and the relative CHAP credentials or through a volume access group. An account can have up to two-thousand volumes assigned to it, but a volume can belong to only one account.

Create a QoS policy

Use SolidFire Quality of Service (QoS) policies if you want to create and save a standardized quality of service setting that can be applied to many volumes.

You can set QoS parameters on a per-volume basis. Performance for each volume can be assured by setting

three configurable parameters that define the QoS: Min IOPS, Max IOPS, and Burst IOPS.

Here are the possible minimum, maximum, and burst IOPS values for the 4Kb block size.

IOPS parameter	Definition	Min. value	Default value	Max. value(4Kb)
Min IOPS	The guaranteed level of performance for a volume.	50	50	15000
Max IOPS	The performance will not exceed this limit.	50	15000	200,000
Burst IOPS	Maximum IOPS allowed in a short burst scenario.	50	15000	200,000

Although the Max IOPS and Burst IOPS can be set as high as 200,000, the real-world maximum performance of a volume is limited by cluster usage and per-node performance.

Block size and bandwidth have a direct influence on the number of IOPS. As block sizes increase, the system increases bandwidth to a level necessary to process the larger block sizes. As bandwidth increases, the number of IOPS the system is able to attain decreases. Refer to SolidFire Quality of Service for more information about QoS and performance.

SolidFire authentication

Element supports two methods for authentication: CHAP and Volume Access Groups (VAG). CHAP uses the CHAP protocol to authenticate the host to the backend. Volume Access Groups controls access to the volumes it provisions. NetApp recommends using CHAP for authentication as it's simpler and has no scaling limits.

Trident with the enhanced CSI provisioner supports the use of CHAP authentication. VAGs should only be used in the traditional non-CSI mode of operation.

CHAP authentication (verification that the initiator is the intended volume user) is supported only with account-based access control. If you are using CHAP for authentication, two options are available: unidirectional CHAP and bidirectional CHAP. Unidirectional CHAP authenticates volume access by using the SolidFire account name and initiator secret. The bidirectional CHAP option provides the most secure way of authenticating the volume because the volume authenticates the host through the account name and the initiator secret, and then the host authenticates the volume through the account name and the target secret.

However, if CHAP cannot be enabled and VAGs are required, create the access group and add the host initiators and volumes to the access group. Each IQN that you add to an access group can access each volume in the group with or without CHAP authentication. If the iSCSI initiator is configured to use CHAP authentication, account-based access control is used. If the iSCSI initiator is not configured to use CHAP authentication, then Volume Access Group access control is used.

Where to find more information?

Some of the best practices documentation is listed below. Search the NetApp library for the most current versions.

ONTAP

- NFS Best Practice and Implementation Guide
- SAN Administration (for iSCSI)
- iSCSI Express Configuration for RHEL

Element software

- Configuring SolidFire for Linux

NetApp HCI

- NetApp HCI deployment prerequisites
- Access the NetApp Deployment Engine

Application best practices information

- Best practices for MySQL on ONTAP
- Best practices for MySQL on SolidFire
- NetApp SolidFire and Cassandra
- Oracle best practices on SolidFire
- PostgreSQL best practices on SolidFire

Not all applications have specific guidelines, it's important to work with your NetApp team and to use the NetApp library to find the most up-to-date documentation.

Integrate Trident

To integrate Trident, the following design and architectural elements require integration: driver selection and deployment, storage class design, virtual pool design, Persistent Volume Claim (PVC) impacts on storage provisioning, volume operations, and OpenShift services deployment using Trident.

Driver selection and deployment

Select and deploy a backend driver for your storage system.

ONTAP backend drivers

ONTAP backend drivers are differentiated by the protocol used and how the volumes are provisioned on the storage system. Therefore, give careful consideration when deciding which driver to deploy.

At a higher level, if your application has components which need shared storage (multiple pods accessing the same PVC), NAS-based drivers would be the default choice, while the block-based iSCSI drivers meet the needs of non-shared storage. Choose the protocol based on the requirements of the application and the comfort level of the storage and infrastructure teams. Generally speaking, there is little difference between them for most applications, so often the decision is based upon whether or not shared storage (where more than one pod will need simultaneous access) is needed.

The available ONTAP backend drivers are:

- `ontap-nas`: Each PV provisioned is a full ONTAP FlexVolume.
- `ontap-nas-economy`: Each PV provisioned is a qtree, with a configurable number of qtrees per FlexVolume (default is 200).
- `ontap-nas-flexgroup`: Each PV provisioned as a full ONTAP FlexGroup, and all aggregates assigned to a SVM are used.
- `ontap-san`: Each PV provisioned is a LUN within its own FlexVolume.
- `ontap-san-economy`: Each PV provisioned is a LUN, with a configurable number of LUNs per FlexVolume (default is 100).

Choosing between the three NAS drivers has some ramifications to the features, which are made available to the application.

Note that, in the tables below, not all of the capabilities are exposed through Trident. Some must be applied by the storage administrator after provisioning if that functionality is desired. The superscript footnotes distinguish the functionality per feature and driver.

ONTAP NAS drivers	Snapshots	Clones	Dynamic export policies	Multi-attach	QoS	Resize	Replication
`ontap-nas`	Yes	Yes	Yes [5]	Yes	Yes [1]	Yes	Yes [1]
`ontap-nas-economy`	NO [3]	NO [3]	Yes [5]	Yes	NO [3]	Yes	NO [3]
`ontap-nas-flexgroup`	Yes [1]	NO	Yes [5]	Yes	Yes [1]	Yes	Yes [1]

Trident offers 2 SAN drivers for ONTAP, whose capabilities are shown below.

ONTAP SAN drivers	Snapshots	Clones	Multi-attach	Bi-directional CHAP	QoS	Resize	Replication
`ontap-san`	Yes	Yes	Yes [4]	Yes	Yes [1]	Yes	Yes [1]
`ontap-san-economy`	Yes	Yes	Yes [4]	Yes	NO [3]	Yes	NO [3]

> Footnote for the above tables:
>
> Yes [1]: Not managed by Trident
>
> Yes [2]: Managed by Trident, but not PV granular
>
> NO [3]: Not managed by Trident and not PV granular
>
> Yes [4]: Supported for raw-block volumes
>
> Yes [5]: Supported by Trident

The features that are not PV granular are applied to the entire FlexVolume and all of the PVs (that is, qtrees or

LUNs in shared FlexVols) will share a common schedule.

As we can see in the above tables, much of the functionality between the `ontap-nas` and `ontap-nas-economy` is the same. However, because the `ontap-nas-economy` driver limits the ability to control the schedule at per-PV granularity, this can affect your disaster recovery and backup planning in particular. For development teams which desire to leverage PVC clone functionality on ONTAP storage, this is only possible when using the `ontap-nas`, `ontap-san` or `ontap-san-economy` drivers.

The `solidfire-san` driver is also capable of cloning PVCs.

Cloud Volumes ONTAP backend drivers

Cloud Volumes ONTAP provides data control along with enterprise-class storage features for various use cases, including file shares and block-level storage serving NAS and SAN protocols (NFS, SMB / CIFS, and iSCSI). The compatible drivers for Cloud Volume ONTAP are `ontap-nas`, `ontap-nas-economy`, `ontap-san` and `ontap-san-economy`. These are applicable for Cloud Volume ONTAP for Azure, Cloud Volume ONTAP for GCP.

Amazon FSx for ONTAP backend drivers

Amazon FSx for NetApp ONTAP lets you leverage NetApp features, performance, and administrative capabilities you're familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx for ONTAP supports many ONTAP file system features and administration APIs. The compatible drivers for Cloud Volume ONTAP are `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san` and `ontap-san-economy`.

NetApp HCI/SolidFire backend drivers

The `solidfire-san` driver used with the NetApp HCI/SolidFire platforms, helps the admin configure an Element backend for Trident on the basis of QoS limits. If you would like to design your backend to set the specific QoS limits on the volumes provisioned by Trident, use the `type` parameter in the backend file. The admin also can restrict the volume size that could be created on the storage using the `limitVolumeSize` parameter. Currently, Element storage features like volume resize and volume replication are not supported through the `solidfire-san` driver. These operations should be done manually through Element Software web UI.

SolidFire Driver	Snapshots	Clones	Multi-attach	CHAP	QoS	Resize	Replication
`solidfire-san`	Yes	Yes	Yes [2]	Yes	Yes	Yes	Yes [1]

> Footnote:
>
> Yes [1]: Not managed by Trident
>
> Yes [2]: Supported for raw-block volumes

Azure NetApp Files backend drivers

Trident uses the `azure-netapp-files` driver to manage the Azure NetApp Files service.

More information about this driver and how to configure it can be found in Trident backend configuration for Azure NetApp Files.

Azure NetApp Files Driver	Snapshots	Clones	Multi-attach	QoS	Expand	Replication
`azure-netapp-files`	Yes	Yes	Yes	Yes	Yes	Yes [1]

> Footnote:
>
> Yes [1]: Not managed by Trident

Storage class design

Individual Storage classes need to be configured and applied to create a Kubernetes Storage Class object. This section discusses how to design a storage class for your application.

Specific backend utilization

Filtering can be used within a specific storage class object to determine which storage pool or set of pools are to be used with that specific storage class. Three sets of filters can be set in the Storage Class: `storagePools`, `additionalStoragePools`, and/or `excludeStoragePools`.

The `storagePools` parameter helps restrict storage to the set of pools that match any specified attributes. The `additionalStoragePools` parameter is used to extend the set of pools that Trident use for provisioning along with the set of pools selected by the attributes and `storagePools` parameters. You can use either parameter alone or both together to make sure that the appropriate set of storage pools are selected.

The `excludeStoragePools` parameter is used to specifically exclude the listed set of pools that match the attributes.

Emulate QoS policies

If you would like to design Storage Classes to emulate Quality of Service policies, create a Storage Class with the `media` attribute as `hdd` or `ssd`. Based on the `media` attribute mentioned in the storage class, Trident will select the appropriate backend that serves `hdd` or `ssd` aggregates to match the media attribute and then direct the provisioning of the volumes on to the specific aggregate. Therefore we can create a storage class PREMIUM which would have `media` attribute set as `ssd` which could be classified as the PREMIUM QoS policy. We can create another storage class STANDARD which would have the media attribute set as `hdd' which could be classified as the STANDARD QoS policy. We could also use the ``IOPS'' attribute in the storage class to redirect provisioning to an Element appliance which can be defined as a QoS Policy.

Utilize backend based on specific features

Storage classes can be designed to direct volume provisioning on a specific backend where features such as thin and thick provisioning, snapshots, clones, and encryption are enabled. To specify which storage to use, create Storage Classes that specify the appropriate backend with the required feature enabled.

Virtual pools

Virtual pools are available for all Trident backends. You can define virtual pools for any backend, using any driver that Trident provides.

Virtual pools allow an administrator to create a level of abstraction over backends which can be referenced through Storage Classes, for greater flexibility and efficient placement of volumes on backends. Different backends can be defined with the same class of service. Moreover, multiple storage pools can be created on the same backend but with different characteristics. When a Storage Class is configured with a selector with the specific labels, Trident chooses a backend which matches all the selector labels to place the volume. If the Storage Class selector labels matches multiple storage pools, Trident will choose one of them to provision the volume from.

Virtual pool design

While creating a backend, you can generally specify a set of parameters. It was impossible for the administrator to create another backend with the same storage credentials and with a different set of parameters. With the introduction of virtual pools, this issue has been alleviated. A virtual pool is a level abstraction introduced between the backend and the Kubernetes Storage Class so that the administrator can define parameters along with labels which can be referenced through Kubernetes Storage Classes as a selector, in a backend-agnostic way. Virtual pools can be defined for all supported NetApp backends with Trident. That list includes SolidFire/NetApp HCI, ONTAP, as well as Azure NetApp Files.

When defining virtual pools, it is recommended to not attempt to rearrange the order of existing virtual pools in a backend definition. It is also advisable to not edit/modify attributes for an existing virtual pool and define a new virtual pool instead.

Emulating different service levels/QoS

It is possible to design virtual pools for emulating service classes. Using the virtual pool implementation for Cloud Volume Service for Azure NetApp Files, let us examine how we can setup up different service classes. Configure the Azure NetApp Files backend with multiple labels, representing different performance levels. Set `servicelevel` aspect to the appropriate performance level and add other required aspects under each labels. Now create different Kubernetes Storage Classes that would map to different virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools may be used to host a volume.

Assigning specific set of aspects

Multiple virtual pools with a specific set of aspects can be designed from a single storage backend. For doing so, configure the backend with multiple labels and set the required aspects under each label. Now create different Kubernetes Storage Classes using the `parameters.selector` field that would map to different virtual pools. The volumes that get provisioned on the backend will have the aspects defined in the chosen virtual pool.

PVC characteristics which affect storage provisioning

Some parameters beyond the requested storage class may affect the Trident provisioning decision process when creating a PVC.

Access mode

When requesting storage via a PVC, one of the mandatory fields is the access mode. The mode desired may affect the backend selected to host the storage request.

Trident will attempt to match the storage protocol used with the access method specified according to the following matrix. This is independent of the underlying storage platform.

	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
iSCSI	Yes	Yes	Yes (Raw block)
NFS	Yes	Yes	Yes

A request for a ReadWriteMany PVC submitted to a Trident deployment without an NFS backend configured will result in no volume being provisioned. For this reason, the requestor should use the access mode which is appropriate for their application.

Volume operations

Modify persistent volumes

Persistent volumes are, with two exceptions, immutable objects in Kubernetes. Once created, the reclaim policy and the size can be modified. However, this doesn't prevent some aspects of the volume from being modified outside of Kubernetes. This may be desirable in order to customize the volume for specific applications, to ensure that capacity is not accidentally consumed, or simply to move the volume to a different storage controller for any reason.

Kubernetes in-tree provisioners do not support volume resize operations for NFS, iSCSI, or FC PVs at this time. Trident supports expanding both NFS, iSCSI, and FC volumes.

The connection details of the PV cannot be modified after creation.

Create on-demand volume snapshots

Trident supports on-demand volume snapshot creation and the creation of PVCs from snapshots using the CSI framework. Snapshots provide a convenient method of maintaining point-in-time copies of the data and have a lifecycle independent of the source PV in Kubernetes. These snapshots can be used to clone PVCs.

Create volumes from snapshots

Trident also supports the creation of PersistentVolumes from volume snapshots. To accomplish this, just create a PersistentVolumeClaim and mention the `datasource` as the required snapshot from which the volume needs to be created. Trident will handle this PVC by creating a volume with the data present on the snapshot. With this feature, it is possible to duplicate data across regions, create test environments, replace a damaged or corrupted production volume in its entirety, or retrieve specific files and directories and transfer them to another attached volume.

Move volumes in the cluster

Storage administrators have the ability to move volumes between aggregates and controllers in the ONTAP cluster non-disruptively to the storage consumer. This operation does not affect Trident or the Kubernetes cluster, as long as the destination aggregate is one which the SVM that Trident is using has access to. Importantly, if the aggregate has been newly added to the SVM, the backend will need to be refreshed by re-adding it to Trident. This will trigger Trident to reinventory the SVM so that the new aggregate is recognized.

However, moving volumes across backends is not supported automatically by Trident. This includes between SVMs in the same cluster, between clusters, or onto a different storage platform (even if that storage system is one which is connected to Trident).

If a volume is copied to another location, the volume import feature may be used to import current volumes into Trident.

Expand volumes

Trident supports resizing NFS, iSCSI, and FC PVs. This enables users to resize their volumes directly through the Kubernetes layer. Volume expansion is possible for all major NetApp storage platforms, including ONTAP, and SolidFire/NetApp HCI backends. To allow possible expansion later, set `allowVolumeExpansion` to `true` in your StorageClass associated with the volume. Whenever the Persistent Volume needs to be resized, edit the `spec.resources.requests.storage` annotation in the Persistent Volume Claim to the required volume size. Trident will automatically take care of resizing the volume on the storage cluster.

Import an existing volume into Kubernetes

Volume import provides the ability to import an existing storage volume into a Kubernetes environment. This is currently supported by the `ontap-nas`, `ontap-nas-flexgroup`, `solidfire-san`, and `azure-netapp-files` drivers. This feature is useful when porting an existing application into Kubernetes or during disaster recovery scenarios.

When using the ONTAP and `solidfire-san` drivers, use the command `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml` to import an existing volume into Kubernetes to be managed by Trident. The PVC YAML or JSON file used in the import volume command points to a storage class which identifies Trident as the provisioner. When using a NetApp HCI/SolidFire backend, ensure the volume names are unique. If the volume names are duplicated, clone the volume to a unique name so the volume import feature can distinguish between them.

If the `azure-netapp-files` driver is used, use the command `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` to import the volume into Kubernetes to be managed by Trident. This ensures a unique volume reference.

When the above command is executed, Trident will find the volume on the backend and read its size. It will automatically add (and overwrite if necessary) the configured PVC's volume size. Trident then creates the new PV and Kubernetes binds the PVC to the PV.

If a container was deployed such that it required the specific imported PVC, it would remain in a pending state until the PVC/PV pair are bound via the volume import process. After the PVC/PV pair are bound, the container should come up, provided there are no other issues.

Registry service

Deploying and managing storage for the registry has been documented on netapp.io in the blog.

Logging service

Like other OpenShift services, the logging service is deployed using Ansible with configuration parameters supplied by the inventory file, a.k.a. hosts, provided to the playbook. There are two installation methods which will be covered: deploying logging during initial OpenShift install and deploying logging after OpenShift has been
installed.

As of Red Hat OpenShift version 3.9, the official documentation recommends against NFS for the logging service due to concerns around data corruption. This is based on Red Hat testing of their products. The ONTAP NFS server does not have these issues, and can easily back a logging deployment. Ultimately, the choice of protocol for the logging service is up to you, just know that both will work great when using NetApp platforms and there is no reason to avoid NFS if that is your preference.

If you choose to use NFS with the logging service, you will need to set the Ansible variable `openshift_enable_unsupported_configurations` to `true` to prevent the installer from failing.

Get started

The logging service can, optionally, be deployed for both applications as well as for the core operations of the OpenShift cluster itself. If you choose to deploy operations logging, by specifying the variable `openshift_logging_use_ops` as `true`, two instances of the service will be created. The variables which control the logging instance for operations contain "ops" in them, whereas the instance for applications does not.

Configuring the Ansible variables according to the deployment method is important to ensure that the correct storage is utilized by the underlying services. Let's look at the options for each of the deployment methods.

The tables below contain only the variables relevant for storage configuration as it relates to the logging service. You can find other options in Red Hat OpenShift logging documentation which should be reviewed, configured, and used according to your deployment.

The variables in the below table will result in the Ansible playbook creating a PV and PVC for the logging service using the details provided. This method is significantly less flexible than using the component installation playbook after OpenShift installation, however, if you have existing volumes available, it is an option.

Variable	Details
`openshift_logging_storage_kind`	Set to `nfs` to have the installer create an NFS PV for the logging service.
`openshift_logging_storage_host`	The hostname or IP address of the NFS host. This should be set to the dataLIF for your virtual machine.
`openshift_logging_storage_nfs_directory`	The mount path for the NFS export. For example, if the volume is junctioned as `/openshift_logging`, you would use that path for this variable.
`openshift_logging_storage_volume_name`	The name, e.g. `pv_ose_logs`, of the PV to create.
`openshift_logging_storage_volume_size`	The size of the NFS export, for example `100Gi`.

If your OpenShift cluster is already running, and therefore Trident has been deployed and configured, the installer can use dynamic provisioning to create the volumes. The following variables will need to be configured.

Variable	Details
`openshift_logging_es_pvc_dynamic`	Set to true to use dynamically provisioned volumes.
`openshift_logging_es_pvc_storage_class_name`	The name of the storage class which will be used in the PVC.
`openshift_logging_es_pvc_size`	The size of the volume requested in the PVC.
`openshift_logging_es_pvc_prefix`	A prefix for the PVCs used by the logging service.
`openshift_logging_es_ops_pvc_dynamic`	Set to `true` to use dynamically provisioned volumes for the ops logging instance.

Variable	Details
`openshift_logging_es_ops_pvc_storage_class_name`	The name of the storage class for the ops logging instance.
`openshift_logging_es_ops_pvc_size`	The size of the volume request for the ops instance.
`openshift_logging_es_ops_pvc_prefix`	A prefix for the ops instance PVCs.

Deploy the logging stack

If you are deploying logging as a part of the initial OpenShift install process, then you only need to follow the standard deployment process. Ansible will configure and deploy the needed services and OpenShift objects so that the service is available as soon as Ansible completes.

However, if you are deploying after the initial installation, the component playbook will need to be used by Ansible. This process may change slightly with different versions of OpenShift, so be sure to read and follow Red Hat OpenShift Container Platform 3.11 documentation for your version.

Metrics service

The metrics service provides valuable information to the administrator regarding the status, resource utilization, and availability of the OpenShift cluster. It is also necessary for pod auto-scale functionality and many organizations use data from the metrics service for their charge back and/or show back applications.

Like with the logging service, and OpenShift as a whole, Ansible is used to deploy the metrics service. Also, like the logging service, the metrics service can be deployed during an initial setup of the cluster or after its operational using the component installation method. The following tables contain the variables which are important when configuring persistent storage for the metrics service.

> The tables below only contain the variables which are relevant for storage configuration as it relates to the metrics service. There are many other options found in the documentation which should be reviewed, configured, and used according to your deployment.

Variable	Details
`openshift_metrics_storage_kind`	Set to `nfs` to have the installer create an NFS PV for the logging service.
`openshift_metrics_storage_host`	The hostname or IP address of the NFS host. This should be set to the dataLIF for your SVM.
`openshift_metrics_storage_nfs_directory`	The mount path for the NFS export. For example, if the volume is junctioned as `/openshift_metrics`, you would use that path for this variable.
`openshift_metrics_storage_volume_name`	The name, e.g. `pv_ose_metrics`, of the PV to create.
`openshift_metrics_storage_volume_size`	The size of the NFS export, for example `100Gi`.

If your OpenShift cluster is already running, and therefore Trident has been deployed and configured, the installer can use dynamic provisioning to create the volumes. The following variables will need to be configured.

Variable	Details
`openshift_metrics_cassandra_pvc_prefix`	A prefix to use for the metrics PVCs.
`openshift_metrics_cassandra_pvc_size`	The size of the volumes to request.
`openshift_metrics_cassandra_storage_type`	The type of storage to use for metrics, this must be set to dynamic for Ansible to create PVCs with the appropriate storage class.
`openshift_metrics_cassanda_pvc_storage_class_name`	The name of the storage class to use.

Deploy the metrics service

With the appropriate Ansible variables defined in your hosts/inventory file, deploy the service using Ansible. If you are deploying at OpenShift install time, then the PV will be created and used automatically. If you're deploying using the component playbooks, after OpenShift install, then Ansible creates any PVCs which are needed and, after Trident has provisioned storage for them, deploy the service.

The variables above, and the process for deploying, may change with each version of OpenShift. Ensure you review and follow Red Hat's OpenShift deployment guide for your version so that it is configured for your environment.

Data protection and disaster recovery

Learn about protection and recovery options for Trident and volumes created using Trident. You should have a data protection and recovery strategy for each application with a persistence requirement.

Trident replication and recovery

You can create a backup to restore Trident in the event of a disaster.

Trident replication

Trident uses Kubernetes CRDs to store and manage its own state and the Kubernetes cluster etcd to store its metadata.

Steps

1. Back up the Kubernetes cluster etcd using Kubernetes: Backing up an etcd cluster.
2. Place the backup artifacts on a FlexVol volume

> NetApp recommends you protect the SVM where the FlexVol resides with a SnapMirror relationship to another SVM.

Trident recovery

Using Kubernetes CRDs and the Kubernetes cluster etcd snapshot, you can recover Trident.

Steps

1. From the destination SVM, mount the volume which contains the Kubernetes etcd data files and certificates

on to the host which will be set up as a master node.

2. Copy all required certificates pertaining to the Kubernetes cluster under `/etc/kubernetes/pki` and the etcd member files under `/var/lib/etcd`.
3. Restore the Kubernetes cluster from the etcd backup using Kubernetes: Restoring an etcd cluster.
4. Run `kubectl get crd` to verify all Trident custom resources have come up and retrieve the Trident objects to verify all data is available.

SVM replication and recovery

Trident cannot configure replication relationships, however, the storage administrator can use ONTAP SnapMirror to replicate an SVM.

In the event of a disaster, you can activate the SnapMirror destination SVM to start serving data. You can switch back to the primary when systems are restored.

About this task

Consider the following when using the SnapMirror SVM Replication feature:

- You should create a distinct backend for each SVM with SVM-DR enabled.
- Configure the storage classes to select the replicated backends only when needed to avoid having volumes which do not need replication provisioned onto the backends that support SVM-DR.
- Application administrators should understand the additional cost and complexity associated with replication and carefully consider their recovery plan prior to beginning this process.

SVM replication

You can use ONTAP: SnapMirror SVM replication to create the SVM replication relationship.

SnapMirror allows you to set options to control what to replicate. You'll need to know which options you selected when preforming SVM recovery using Trident.

- -identity-preserve true replicates the entire SVM configuration.
- -discard-configs network excludes LIFs and related network settings.
- -identity-preserve false replicates only the volumes and security configuration.

SVM recovery using Trident

Trident does not automatically detect SVM failures. In the event of a disaster, the administrator can manually initiate Trident failover to the new SVM.

Steps

1. Cancel scheduled and ongoing SnapMirror transfers, break the replication relationship, stop the source SVM and then activate the SnapMirror destination SVM.
2. If you specified `-identity-preserve false` or `-discard-config network` when configuring your SVM replication, update the `managementLIF` and `dataLIF` in the Trident backend definition file.
3. Confirm `storagePrefix` is present in the Trident backend definition file. This parameter cannot be changed. Omitting `storagePrefix` will cause the backend update to fail.
4. Update all the required backends to reflect the new destination SVM name using:

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

5. If you specified `-identity-preserve false` or `discard-config network`, you must bounce all application pods.

> If you specified `-identity-preserve true`, all volumes provisioned by Trident start serving data when the destination SVM is activated.

Volume replication and recovery

Trident cannot configure SnapMirror replication relationships, however, the storage administrator can use ONTAP SnapMirror replication and recovery to replicate volumes created by Trident.

You can then import the recovered volumes into Trident using tridentctl volume import.

> Import is not supported on `ontap-nas-economy`, `ontap-san-economy`, or `ontap-flexgroup-economy` drivers.

Snapshot data protection

You can protect and restore data using:

- An external snapshot controller and CRDs to create Kubernetes volume snapshots of Persistent Volumes (PVs).

  Volume snapshots

- ONTAP Snapshots to restore the entire contents of a volume or to recover individual files or LUNs.

  ONTAP Snapshots

Automating the failover of stateful applications with Trident

Trident's force-detach feature allows you to automatically detach volumes from unhealthy nodes in a Kubernetes cluster, preventing data corruption and ensuring application availability. This feature is particularly useful in scenarios where nodes become unresponsive or are taken offline for maintenance.

Details about force detach

Force detach is available for `ontap-san`, `ontap-san-economy`, `ontap-nas`, and `ontap-nas-economy` only. Before enabling force detach, non-graceful node shutdown (NGNS) must be enabled on the Kubernetes cluster. NGNS is enabled by default for Kubernetes 1.28 and above. For more information, refer to Kubernetes: Non Graceful node shutdown.

When using the `ontap-nas` or `ontap-nas-economy` driver, you need to set the `autoExportPolicy` parameter in the backend configuration to `true` so that Trident can restrict access from the Kubernetes node with the taint applied using managed export policies.

Because Trident relies on Kubernetes NGNS, do not remove `out-of-service` taints from an unhealthy node until all non-tolerable workloads are rescheduled. Recklessly applying or removing the taint can jeopardize backend data protection.

When the Kubernetes cluster administrator has applied the `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute` taint to the node and `enableForceDetach` is set to `true`, Trident will determine the node status and:

1. Stop backend I/O access for volumes mounted to that node.
2. Mark the Trident node object as `dirty` (not safe for new publications).

The Trident controller will reject new publish volume requests until the node is re-qualified (after having been marked as `dirty`) by the Trident node pod. Any workloads scheduled with a mounted PVC (even after the cluster node is healthy and ready) will be not be accepted until Trident can verify the node `clean` (safe for new publications).

When node health is restored and the taint is removed, Trident will:

1. Identify and clean stale published paths on the node.
2. If the node is in a `cleanable` state (the out-of-service taint has been removed and the node is in `Ready` state) and all stale, published paths are clean, Trident will readmit the node as `clean` and allow new published volumes to the node.

Details about automated failover

You can automate the force-detach process through integration with node health check (NHC) operator. When a node failure occurs, NHC triggers Trident node remediation (TNR) and force-detach automatically by creating a TridentNodeRemediation CR in Trident's namespace defining the failed node. TNR is created only upon node failure, and removed by NHC once the node comes back online or the node is deleted.

Failed node pod removal process

Automated-failover selects the workloads to remove from the failed node. When a TNR is created, the TNR controller marks the node as dirty, preventing any new volume publications and begins removing force-detach supported pods and their volume attachments.

All volumes/PVCs supported by force-detach are supported by automated-failover:

- NAS, and NAS-economy volumes using auto-export policies (SMB is not yet supported).
- SAN, and SAN-economy volumes.

Refer to Details about force detach.

Default behavior:

- Pods using volumes supported by force-detach are removed from the failed node. Kubernetes will

reschedule these onto a healthy node.

- Pods using a volume not supported by force-detach, including non-Trident volumes, are not removed from the failed node.
- Stateless pods (not PVCs) are not removed from the failed node, unless the pod annotation `trident.netapp.io/podRemediationPolicy: delete` is set.

Overriding the pod removal behavior:

Pod removal behavior can be customized using a pod annotation: `trident.netapp.io/podRemediationPolicy[retain, delete]`. These annotations are examined and used when a failover occurs.
Apply annotations to the Kubernetes deployment/replicaset pod spec to prevent the annotation from disappearing after a failover:

- `retain` - Pod WILL NOT be removed from the failed node during an automated-failover.
- `delete` - Pod WILL be removed from the failed node during an automated-failover.

These annotations can be applied to any pod.

- I/O operations will be blocked only on failed nodes for volumes that support force-detach.
- For volumes that do not support force-detach, there is a risk of data corruption and multi-attach issues.

TridentNodeRemediation CR

The TridentNodeRemediation (TNR) CR defines a failed node. The name of the TNR is the name of the failed node.

Example TNR:

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediation
metadata:
  name: <K8s-node-name>
spec: {}
```

TNR states:
Use the following commands to view the status of TNRs:
`kubectl get tnr <name> -n <trident-namespace>`

TNRs can be in one of the following states:

- *Remediating*:
  - Cease backend I/O access for volumes supported by force-detach mounted to that node.
  - The Trident node object is marked dirty (not safe for new publications).
  - Remove pods and volume attachments from the node
- *NodeRecoveryPending*:

    - The controller is waiting for the node to come back online.
    - Once the node is online, publish-enforcement will ensure the node is clean and ready for new volume publications.
- If the node is deleted from K8s, the TNR controller will remove the TNR and cease reconciliation.
- *Succeeded*:
    - All remediation and node recovery steps completed successfully. The node is clean and ready for new volume publications.
- *Failed*:
    - Unrecoverable error. Error reasons are set in the status.message field of the CR.

Enabling automated-failover

Prerequisites:

- Ensure that force detach is enabled before enabling automated-failover. For more information, refer to Details about force detach.
- Install node health check (NHC) in the Kubernetes cluster.
    - Install operator-sdk.
    - Install Operator Lifecycle Manager (OLM) in the cluster if not already installed: `operator-sdk olm install`.
    - Install Node Health check Operator: `kubectl create -f https://operatorhub.io/install/node-healthcheck-operator.yaml`.

> You can also use alternative ways to detect node failure as specified in the Integrating custom node health check solutions section below.

See Node Health Check Operator for more information.

Steps

1. Create a NodeHealthCheck (NHC) CR in the Trident namespace to monitor the worker nodes in the cluster. Example:

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: <CR name>
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  remediationTemplate:
    apiVersion: trident.netapp.io/v1
    kind: TridentNodeRemediationTemplate
    namespace: <Trident installation namespace>
    name: trident-node-remediation-template
  minHealthy: 0 # Trigger force-detach upon one or more node failures
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 0s
    - type: Ready
      status: Unknown
      duration: 0s
```

2. Apply the node health check CR in the `trident` namespace.

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

The above CR is configured to watch K8s worker nodes for node conditions Ready: false and Unknown. Automated-Failover will be triggered upon a node going into Ready: false, or Ready: Unknown state.

The `unhealthyConditions` in the CR uses a 0 second grace period. This causes automated-failover to trigger immediately upon K8s setting node condition Ready: false, which is set after K8s loses the heartbeat from a node. K8s has a default of 40sec wait after the last heartbeat before setting Ready: false. This grace-period can be customized in K8s deployment options.

For additional configuration options, refer to Node-Healthcheck-Operator documentation.

Additional setup information

When Trident is installed with force-detach enabled, two additional resources are automatically created in the Trident namespace to facilitate integration with NHC: TridentNodeRemediationTemplate (TNRT) and ClusterRole.

TridentNodeRemediationTemplate (TNRT):

The TNRT serves as a template for the NHC controller, which uses TNRT to generate TNR resources as

needed.

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

ClusterRole:

A cluster role is also added during the installation when force-detach is enabled. This gives NHC permissions to TNRs in the Trident namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.ext-remediation/aggregate-to-ext-remediation: "true"
  name: tridentnoderemediation-access
rules:
- apiGroups:
  - trident.netapp.io
  resources:
  - tridentnoderemediationtemplates
  - tridentnoderemediations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

K8s cluster upgrades and maintenance

To prevent any failovers, pause automated-failover during K8s maintenance or upgrades, where the nodes are expected to go down or reboot. You can pause the NHC CR (described above) by patching its CR:

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

This pauses the automated-failover. To re-enable automated-failover, remove the pauseRequests from the

spec after the maintenance is complete.

Limitations

- I/O operations are prevented only on the failed nodes for volumes supported by force-detach. Only pods using volumes/PVCs supported by force-detach are automatically removed.
- Automatic-failover and force-detach run inside the trident-controller pod. If the node hosting trident-controller fails, automated-failover will be delayed until K8s moves the pod to a healthy node.

Integrating custom node health check solutions

You can replace Node Healthcheck Operator with alternative node failure detection tools to trigger automatic-failover.
To ensure compatibility with the automated failover mechanism, your custom solution should:

- Create a TNR when a node failure is detected, using the failed node's name as the TNR CR name.
- Delete the TNR when the node has recovered and the TNR is in the Succeeded state.

Security

Security

Use the recommendations listed here to ensure your Trident installation is secure.

Run Trident in its own namespace

It is important to prevent applications, application administrators, users, and management applications from accessing Trident object definitions or the pods to ensure reliable storage and block potential malicious activity.

To separate the other applications and users from Trident, always install Trident in its own Kubernetes namespace (`trident`). Putting Trident in its own namespace assures that only the Kubernetes administrative personnel have access to the Trident pod and the artifacts (such as backend and CHAP secrets if applicable) stored in the namespaced CRD objects.
You should ensure that you allow only administrators access to the Trident namespace and thus access to the `tridentctl` application.

Use CHAP authentication with ONTAP SAN backends

Trident supports CHAP-based authentication for ONTAP SAN workloads (using the `ontap-san` and `ontap-san-economy` drivers). NetApp recommends using bidirectional CHAP with Trident for authentication between a host and the storage backend.

For ONTAP backends that use the SAN storage drivers, Trident can set up bidirectional CHAP and manage CHAP usernames and secrets through `tridentctl`.
Refer to Prepare to configure backend with ONTAP SAN drivers to understand how Trident configures CHAP on ONTAP backends.

Use CHAP authentication with NetApp HCI and SolidFire backends

NetApp recommends deploying bidirectional CHAP to ensure authentication between a host and the NetApp HCI and SolidFire backends. Trident uses a secret object that includes two CHAP passwords per tenant. When Trident is installed, it manages the CHAP secrets and stores them in a `tridentvolume` CR object for the

respective PV. When you create a PV, Trident uses the CHAP secrets to initiate an iSCSI session and communicate with the NetApp HCI and SolidFire system over CHAP.

> The volumes that are created by Trident are not associated with any Volume Access Group.

Use Trident with NVE and NAE

NetApp ONTAP provides data-at-rest encryption to protect sensitive data in the event a disk is stolen, returned, or repurposed. For details, refer to Configure NetApp Volume Encryption overview.

- If NAE is enabled on the backend, any volume provisioned in Trident will be NAE-enabled.
  - You can set the NVE encryption flag to `""` to create NAE-enabled volumes.
- If NAE is not enabled on the backend, any volume provisioned in Trident will be NVE-enabled unless the NVE encryption flag is set to `false` (the default value) in the backend configuration.

> Volumes created in Trident on an NAE-enabled backend must be NVE or NAE encrypted.
>
> - You can set the NVE encryption flag to `true` in the Trident backend configuration to override the NAE encryption and use a specific encryption key on a per volume basis.
> - Setting the NVE encryption flag to `false` on an NAE-enabled backend creates an NAE-enabled volume. You cannot disable NAE encryption by setting the NVE encryption flag to `false`.

- You can manually create an NVE volume in Trident by explicitly setting the NVE encryption flag to `true`.

For more information on backend configuration options, refer to:

- ONTAP SAN configuration options
- ONTAP NAS configuration options

Linux Unified Key Setup (LUKS)

You can enable Linux Unified Key Setup (LUKS) to encrypt ONTAP SAN and ONTAP SAN ECONOMY volumes on Trident. Trident supports passphrase rotation and volume expansion for LUKS-encrypted volumes.

In Trident, LUKS-encrypted volumes use the aes-xts-plain64 cypher and mode, as recommended by NIST.

> LUKS encryption is not supported for ASA r2 systems. For information about ASA r2 systems, see Learn about ASA r2 storage systems.

Before you begin

- Worker nodes must have cryptsetup 2.1 or higher (but lower than 3.0) installed. For more information, visit Gitlab: cryptsetup.
- For performance reasons, NetApp recommends that worker nodes support Advanced Encryption Standard New Instructions (AES-NI). To verify AES-NI support, run the following command:

```
grep "aes" /proc/cpuinfo
```

If nothing is returned, your processor does not support AES-NI. For more information on AES-NI, visit: Intel: Advanced Encryption Standard Instructions (AES-NI).

Enable LUKS encryption

You can enable per-volume, host-side encryption using Linux Unified Key Setup (LUKS) for ONTAP SAN and ONTAP SAN ECONOMY volumes.

Steps

1. Define LUKS encryption attributes in the backend configuration. For more information on backend configuration options for ONTAP SAN, refer to ONTAP SAN configuration options.

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

2. Use `parameters.selector` to define the storage pools using LUKS encryption. For example:

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

3. Create a secret that contains the LUKS passphrase. For example:

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

Limitations

LUKS-encrypted volumes cannot take advantage of ONTAP deduplication and compression.

Backend configuration for importing LUKS volumes

To import a LUKS volume, you must set `luksEncryption` to `true` on the backend. The `luksEncryption` option tells Trident if the volume is LUKS-compliant (`true`) or not LUKS-compliant (`false`) as shown in the following example.

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

PVC configuration for importing LUKS volumes

To import LUKS volumes dynamically, set the annotation `trident.netapp.io/luksEncryption` to `true` and include a LUKS-enabled storage class in the PVC as shown in this example.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

Rotate a LUKS passphrase

You can rotate the LUKS passphrase and confirm rotation.

> Do not forget a passphrase until you have verified it is no longer referenced by any volume, snapshot, or secret. If a referenced passphrase is lost, you might be unable to mount the volume and the data will remain encrypted and inaccessible.

About this task

LUKS passphrase rotation occurs when a pod that mounts the volume is created after a new LUKS passphrase is specified. When a new pod is created, Trident compares the LUKS passphrase on the volume to the active passphrase in the secret.

- If the passphrase on the volume does not match the active passphrase in the secret, rotation occurs.
- If the passphrase on the volume matches the active passphrase in the secret, the `previous-luks-passphrase` parameter is ignored.

Steps

1. Add the `node-publish-secret-name` and `node-publish-secret-namespace` StorageClass parameters. For example:

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

2. Identify existing passphrases on the volume or snapshot.

Volume

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

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

Snapshot

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

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

3. Update the LUKS secret for the volume to specify the new and previous passphrases. Ensure `previous-luke-passphrase-name` and `previous-luks-passphrase` match the previous passphrase.

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

4. Create a new pod mounting the volume. This is required to initiate the rotation.
5. Verify the the passphrase was rotated.

Volume

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

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

Snapshot

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

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

Results

The passphrase was rotated when only the new passphrase is returned on the volume and snapshot.

> If two passphrases are returned, for example `luksPassphraseNames: ["B", "A"]`, the rotation is incomplete. You can trigger a new pod to attempt to complete the rotation.

Enable volume expansion

You can enable volume expansion on a LUKS-encrypted volume.

Steps

1. Enable the `CSINodeExpandSecret` feature gate (beta 1.25+). Refer to Kubernetes 1.25: Use Secrets for Node-Driven Expansion of CSI Volumes for details.
2. Add the `node-expand-secret-name` and `node-expand-secret-namespace` StorageClass parameters. For example:

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

Results

When you initiate online storage expansion, the kubelet passes the appropriate credentials to the driver.

Kerberos in-flight encryption

Using Kerberos in-flight encryption, you can improve data access security by enabling encryption for the traffic between your managed cluster and the storage backend.

Trident supports Kerberos encryption for ONTAP as a storage backend:

- **On-premise ONTAP** - Trident supports Kerberos encryption over NFSv3 and NFSv4 connections from Red Hat OpenShift and upstream Kubernetes clusters to on-premise ONTAP volumes.

You can create, delete, resize, snapshot, clone, read-only clone, and import volumes that use NFS encryption.

Configure in-flight Kerberos encryption with on-premise ONTAP volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and an on-premise ONTAP storage backend.

Kerberos encryption for NFS traffic with on-premise ONTAP storage backends is only supported using the `ontap-nas` storage driver.

Before you begin

- Ensure that you have access to the `tridentctl` utility.
- Ensure you have administrator access to the ONTAP storage backend.
- Ensure you know the name of the volume or volumes you will be sharing from the ONTAP storage backend.
- Ensure that you have prepared the ONTAP storage VM to support Kerberos encryption for NFS volumes. Refer to Enable Kerberos on a dataLIF for instructions.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Add or modify ONTAP export policies

You need to add rules to existing ONTAP export policies or create new export polices that support Kerberos encryption for the ONTAP storage VM root volume as well as any ONTAP volumes shared with the upstream Kubernetes cluster. The export policy rules you add, or new export policies you create, need to support the following access protocols and access permissions:

Access protocols

Configure the export policy with NFS, NFSv3, and NFSv4 access protocols.

Access details

You can configure one of three different versions of Kerberos encryption, depending on your needs for the volume:

- **Kerberos 5** - (authentication and encryption)
- **Kerberos 5i** - (authentication and encryption with identity protection)
- **Kerberos 5p** - (authentication and encryption with identity and privacy protection)

Configure the ONTAP export policy rule with the appropriate access permissions. For example, if clusters will

be mounting the NFS volumes with a mixture of Kerberos 5i and Kerberos 5p encryption, use the following access settings:

Type	Read-only access	Read/Write access	Superuser access
UNIX	Enabled	Enabled	Enabled
Kerberos 5i	Enabled	Enabled	Enabled
Kerberos 5p	Enabled	Enabled	Enabled

Refer to the following documentation for how to create ONTAP export policies and export policy rules:

- Create an export policy
- Add a rule to an export policy

Create a storage backend

You can create a Trident storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can specify one of three different versions of Kerberos encryption using the `spec.nfsMountOptions` parameter:

- `spec.nfsMountOptions: sec=krb5` (authentication and encryption)
- `spec.nfsMountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `spec.nfsMountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used.

Steps

1. On the managed cluster, create a storage backend configuration file using the following example. Replace values in brackets <> with information from your environment:

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. Use the configuration file you created in the previous step to create the backend:

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

If the backend creation fails, something is wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can run the create command again.

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

About this task

When you create a storage class object, you can specify one of three different versions of Kerberos encryption using the `mountOptions` parameter:

- `mountOptions: sec=krb5` (authentication and encryption)
- `mountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `mountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used. If the level of encryption you specified in the storage backend configuration is different than the level you specify in the storage class object, the storage class object takes precedence.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc ontap-nas-sc
```

   You should see output similar to the following:

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions,

refer to Provision a volume.

Configure in-flight Kerberos encryption with Azure NetApp Files volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and a single Azure NetApp Files storage backend or a virtual pool of Azure NetApp Files storage backends.

Before you begin

- Ensure that you have enabled Trident on the managed Red Hat OpenShift cluster.
- Ensure that you have access to the `tridentctl` utility.
- Ensure that you have prepared the Azure NetApp Files storage backend for Kerberos encryption by noting the requirements and following the instructions in Azure NetApp Files documentation.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Create a storage backend

You can create an Azure NetApp Files storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can define it so that it should be applied at one of two possible levels:

- The **storage backend level** using the `spec.kerberos` field
- The **virtual pool level** using the `spec.storage.kerberos` field

When you define the configuration at the virtual pool level, the pool is selected using the label in the storage class.

At either level, you can specify one of three different versions of Kerberos encryption:

- `kerberos: sec=krb5` (authentication and encryption)
- `kerberos: sec=krb5i` (authentication and encryption with identity protection)
- `kerberos: sec=krb5p` (authentication and encryption with identity and privacy protection)

Steps

1. On the managed cluster, create a storage backend configuration file using one of the following examples, depending on where you need to define the storage backend (storage backend level or virtual pool level). Replace values in brackets <> with information from your environment:

Storage backend level example

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

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

Virtual pool level example

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

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

2. Use the configuration file you created in the previous step to create the backend:

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

If the backend creation fails, something is wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can run the create command again.

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc -sc-nfs
```

You should see output similar to the following:

```
NAME          PROVISIONER             AGE
sc-nfs        csi.trident.netapp.io   15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions, refer to Provision a volume.

Protect applications with Trident Protect

Learn about Trident Protect

NetApp Trident Protect provides advanced application data management capabilities that enhance the functionality and availability of stateful Kubernetes applications backed by NetApp ONTAP storage systems and the NetApp Trident CSI storage provisioner. Trident Protect simplifies the management, protection, and movement of containerized workloads across public clouds and on-premises environments. It also offers automation capabilities through its API and CLI.

You can protect applications with Trident Protect by creating custom resources (CRs) or by using the Trident Protect CLI.

What's next?

You can learn about Trident Protect requirements before you install it:

- Trident Protect requirements

Install Trident Protect

Trident Protect requirements

Get started by verifying the readiness of your operational environment, application clusters, applications, and licenses. Ensure that your environment meets these requirements to deploy and operate Trident Protect.

Trident Protect Kubernetes cluster compatibility

Trident Protect is compatible with a wide range of fully managed and self-managed Kubernetes offerings, including:

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes Service (AKS)
- Red Hat OpenShift
- SUSE Harvester 1.7.0 (ONTAP iSCSI)
- SUSE Rancher
- VMware Tanzu Portfolio
- Upstream Kubernetes

- Trident Protect backups are supported on Linux compute nodes only. Windows compute nodes are not supported for backup operations.
- Ensure that the cluster on which you install Trident Protect is configured with a running snapshot controller and the related CRDs. To install a snapshot controller, refer to these instructions.
- Ensure that at least one VolumeSnapshotClass exists. For more information, refer to VolumeSnapshotClass.
- Helm 4.x or later is required to install Trident Protect.

Trident Protect storage backend compatibility

Trident Protect supports the following storage backends:

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP storage arrays
- Google Cloud NetApp Volumes
- Azure NetApp Files

Ensure that your storage backend meets the following requirements:

- Ensure that NetApp storage connected to the cluster is using Trident 24.02 or newer (Trident 24.10 is recommended).
- Ensure that you have a NetApp ONTAP storage backend.
- Ensure that you have configured an object storage bucket for storing backups.
- Create any application namespaces that you plan to use for applications or application data management operations. Trident Protect does not create these namespaces for you; if you specify a nonexistent namespace in a custom resource, the operation will fail.

Requirements for nas-economy volumes

Trident Protect supports backup and restore operations to nas-economy volumes. Snapshots, clones, and SnapMirror replication to nas-economy volumes are not currently supported. You need to enable a snapshot directory for each nas-economy volume you plan to use with Trident Protect.

Some applications are not compatible with volumes that use a snapshot directory. For these applications, you need to hide the snapshot directory by running the following command on the ONTAP storage system:

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

You can enable the snapshot directory by running the following command for each nas-economy volume, replacing `<volume-UUID>` with the UUID of the volume you want to change:

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

You can enable snapshot directories by default for new volumes by setting the Trident backend configuration option `snapshotDir` to `true`. Existing volumes are not affected.

Protecting data with KubeVirt VMs

Trident Protect provides filesystem freeze and unfreeze capabilities for KubeVirt virtual machines during data protection operations to ensure data consistency. The configuration method and default behavior for VM freeze operations varies across Trident Protect versions, with newer releases offering simplified configuration through Helm chart parameters.

During restore operations, any `VirtualMachineSnapshots` created for a virtual machine (VM) are not restored.

Trident Protect 25.10 and newer

Trident Protect automatically freezes and unfreezes KubeVirt filesystems during data protection operations to ensure consistency. Beginning with Trident Protect 25.10, you can disable this behavior using the `vm.freeze` parameter during Helm chart installation. The parameter is enabled by default.

```
helm install ... --set vm.freeze=false ...
```

Trident Protect 24.10.1 to 25.06

Beginning with Trident Protect 24.10.1, Trident Protect automatically freezes and unfreezes KubeVirt filesystems during data protection operations. Optionally, you can disable this automatic behavior using the following command:

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

Trident Protect 24.10

Trident Protect 24.10 does not automatically ensure a consistent state for KubeVirt VM filesystems during data protection operations. If you want to protect your KubeVirt VM data using Trident Protect 24.10, you need to manually enable the freeze/unfreeze functionality for the filesystems before the data protection operation. This ensures that the filesystems are in a consistent state.

You can configure Trident Protect 24.10 to manage the freezing and unfreezing of the VM filesystem during data protection operations by configuring virtualization and then using the following command:

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

Requirements for SnapMirror replication

NetApp SnapMirror replication is available for use with Trident Protect for the following ONTAP solutions:

- On-premises NetApp FAS, AFF, and ASA systems. SnapMirror replication with Trident protect is not currently supported for ASA r2 systems.
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

ONTAP cluster requirements for SnapMirror replication

Ensure your ONTAP cluster meets the following requirements if you plan to use SnapMirror replication:

- **NetApp Trident**: NetApp Trident must exist on both the source and destination Kubernetes clusters that utilize ONTAP as a backend. Trident Protect supports replication with NetApp SnapMirror technology using storage classes backed by the following drivers:
  - `ontap-nas`: NFS
  - `ontap-san`: iSCSI
  - `ontap-san`: FC
  - `ontap-san`: NVMe/TCP (requires minimum ONTAP version 9.15.1)
- **Licenses**: ONTAP SnapMirror asynchronous licenses using the Data Protection bundle must be enabled on both the source and destination ONTAP clusters. Refer to SnapMirror licensing overview in ONTAP for more information.

  Beginning with ONTAP 9.10.1, all licenses are delivered as a NetApp license file (NLF), which is a single file that enables multiple features. Refer to Licenses included with ONTAP One for more information.

  Only SnapMirror asynchronous protection is supported.

Peering considerations for SnapMirror replication

Ensure your environment meets the following requirements if you plan to use storage backend peering:

- **Cluster and SVM**: The ONTAP storage backends must be peered. Refer to Cluster and SVM peering overview for more information.

  Ensure that the SVM names used in the replication relationship between two ONTAP clusters are unique.

- **NetApp Trident and SVM**: The peered remote SVMs must be available to NetApp Trident on the destination cluster.
- **Managed backends**: You need to add and manage ONTAP storage backends in Trident Protect to create a replication relationship.

Trident / ONTAP configuration for SnapMirror replication

Trident Protect requires that you configure at least one storage backend that supports replication for both the source and destination clusters. If the source and destination clusters are the same, the destination application should use a different storage backend than the source application for the best resiliency.

Kubernetes cluster requirements for SnapMirror replication

Ensure your Kubernetes clusters meet the following requirements:

- **AppVault accessibility**: Both source and destination clusters must have network access to read from and write to the AppVault for application object replication.
- **Network connectivity**: Configure firewall rules, bucket permissions, and IP allowlists to enable communication between both clusters and the AppVault across WANs.

Many enterprise environments implement strict firewall policies across WAN connections. Verify these network requirements with your infrastructure team before configuring replication.

Install and configure Trident Protect

If your environment meets the requirements for Trident Protect, you can follow these steps to install Trident Protect on your cluster. You can obtain Trident Protect from NetApp, or install it from your own private registry. Installing from a private registry is helpful if your cluster cannot access the Internet.

Install Trident Protect

Install Trident Protect from NetApp

Steps

1. Add the Trident Helm repository:

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

2. Use Helm to install Trident Protect. Replace `<name-of-cluster>` with a cluster name, which will be assigned to the cluster and used to identify the cluster's backups and snapshots:

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2602.0 --create
-namespace --namespace trident-protect
```

3. Optionally, to enable debug logging (recommended for troubleshooting), use:

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2602.0 --create-namespace --namespace trident-protect
```

   Debug logging helps NetApp support troubleshoot issues without requiring log level changes or problem reproduction.

Install Trident Protect from a private registry

You can install Trident Protect from a private image registry if your Kubernetes cluster is unable to access the Internet. In these examples, replace values in brackets with information from your environment:

Steps

1. Pull the following images to your local machine, update the tags, and then push them to your private registry:

```
docker.io/netapp/controller:26.02.0
docker.io/netapp/restic:26.02.0
docker.io/netapp/kopia:26.02.0
docker.io/netapp/kopiablockrestore:26.02.0
docker.io/netapp/trident-autosupport:26.02.0
docker.io/netapp/exechook:26.02.0
docker.io/netapp/resourcebackup:26.02.0
docker.io/netapp/resourcerestore:26.02.0
docker.io/netapp/resourcedelete:26.02.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

   For example:

```
docker pull docker.io/netapp/controller:26.02.0
```

```
docker tag docker.io/netapp/controller:26.02.0 <private-registry-url>/controller:26.02.0
```

```
docker push <private-registry-url>/controller:26.02.0
```

> To obtain the Helm chart, first download the Helm chart on a machine with internet access using `helm pull trident-protect --version 100.2602.0 --repo https://netapp.github.io/trident-protect-helm-chart`, then copy the resulting `trident-protect-100.2602.0.tgz` file to your offline environment and install using `helm install trident-protect ./trident-protect-100.2602.0.tgz` instead of the repository reference in the final step.

2. Create the Trident Protect system namespace:

```
kubectl create ns trident-protect
```

3. Log in to the registry:

```
helm registry login <private-registry-url> -u <account-id> -p <api-token>
```

4. Create a pull secret to use for private registry authentication:

```
kubectl create secret docker-registry regcred --docker-username=<registry-username> --docker-password=<api-token> -n trident-protect --docker-server=<private-registry-url>
```

5. Add the Trident Helm repository:

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

6. Create a file named `protectValues.yaml`. Ensure that it contains the following Trident Protect settings:

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

> The `imageRegistry` and `imagePullSecrets` values apply to all component images including `resourcebackup` and `resourcerestore`. If you push images to a specific repository path within your registry (for example, `example.com:443/my-repo`), include the full path in the registry field. This will ensure that all images are pulled from `<private-registry-url>/<image-name>:<tag>`.

7. Use Helm to install Trident Protect. Replace `<name_of_cluster>` with a cluster name, which will be assigned to the cluster and used to identify the cluster's backups and snapshots:

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2602.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

8. Optionally, to enable debug logging (recommended for troubleshooting), use:

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2602.0 --create-namespace --namespace trident-protect -f
protectValues.yaml
```

Debug logging helps NetApp support troubleshoot issues without requiring log level changes or problem reproduction.

> For additional Helm chart configuration options, including AutoSupport settings and namespace filtering, refer to Customize Trident Protect installation.

Install the Trident Protect CLI plugin

You can use the Trident Protect command line plugin, which is an extension of the Trident `tridentctl` utility, to create and interact with Trident Protect custom resources (CRs).

Install the Trident Protect CLI plugin

Before using the command line utility, you need to install it on the machine you use to access your cluster. Follow these steps, depending on if your machine uses an x64 or ARM CPU.

Download plugin for Linux AMD64 CPUs

Steps

1. Download the Trident Protect CLI plugin:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/26.02.0/tridentctl-protect-linux-amd64
```

Download plugin for Linux ARM64 CPUs

Steps

1. Download the Trident Protect CLI plugin:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/26.02.0/tridentctl-protect-linux-arm64
```

Download plugin for Mac AMD64 CPUs

Steps

1. Download the Trident Protect CLI plugin:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/26.02.0/tridentctl-protect-macos-amd64
```

Download plugin for Mac ARM64 CPUs

Steps

1. Download the Trident Protect CLI plugin:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/26.02.0/tridentctl-protect-macos-arm64
```

2. Enable execute permissions for the plugin binary:

```
chmod +x tridentctl-protect
```

3. Copy the plugin binary to a location that is defined in your PATH variable. For example, `/usr/bin` or `/usr/local/bin` (you might need elevated privileges):

```
cp ./tridentctl-protect /usr/local/bin/
```

4. Optionally, you can copy the plugin binary to a location in your home directory. In this case, it is recommended to ensure the location is part of your PATH variable:

```
cp ./tridentctl-protect ~/bin/
```

Copying the plugin to a location in your PATH variable enables you to use the plugin by typing `tridentctl-protect` or `tridentctl protect` from any location.

View Trident CLI plugin help

You can use the built-in plugin help features to get detailed help on the capabilities of the plugin:

Steps

1. Use the help function to view usage guidance:

```
tridentctl-protect help
```

Enable command auto-completion

After you have installed the Trident Protect CLI plugin, you can enable auto-completion for certain commands.

Enable auto-completion for the Bash shell

Steps

1. Create the completion script:

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. Make a new directory in your home directory to contain the script:

```
mkdir -p ~/.bash/completions
```

3. Move the downloaded script to the `~/.bash/completions` directory:

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. Add the following line to the `~/.bashrc` file in your home directory:

```
source ~/.bash/completions/tridentctl-completion.bash
```

Enable auto-completion for the Z shell

Steps

1. Create the completion script:

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. Make a new directory in your home directory to contain the script:

```
mkdir -p ~/.zsh/completions
```

3. Move the downloaded script to the `~/.zsh/completions` directory:

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. Add the following line to the `~/.zprofile` file in your home directory:

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

Result

Upon your next shell login, you can use command auto-completion with the tridentctl-protect plugin.

Customize Trident Protect installation

You can customize the default configuration of Trident Protect to meet the specific requirements of your environment.

Specify Trident Protect container resource limits

You can use a configuration file to specify resource limits for Trident Protect containers after you install Trident Protect. Setting resource limits enables you to control how much of the cluster's resources are consumed by Trident Protect operations.

Steps

1. Create a file named `resourceLimits.yaml`.
2. Populate the file with resource limit options for Trident Protect containers according to the needs of your environment.

   The following example configuration file shows the available settings and contains the default values for each resource limit:

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

```
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. Apply the values from the `resourceLimits.yaml` file:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

Customize security context constraints

You can use a configuration file to modify OpenShift security context constraint (SCCs) for Trident Protect containers after you install Trident Protect. These constraints define security restrictions for pods in a Red Hat OpenShift cluster.

Steps

1. Create a file named `sccconfig.yaml`.
2. Add the SCC option to the file and modify the parameters according to the needs of your environment.

   The following example shows the default values of the parameters for the SCC option:

```
scc:
  create: true
  name: trident-protect-job
  priority: 1
```

This table describes the parameters for the SCC option:

Parameter	Description	Default
create	Determines whether an SCC resource can be created. An SCC resource will be created only if `scc.create` is set to `true` and the Helm installation process identifies an OpenShift environment. If not operating on OpenShift, or if `scc.create` is set to `false`, no SCC resource will be created.	true
name	Specifies the name of the SCC.	trident-protect-job
priority	Defines the priority of the SCC. SCCs with higher priority values are assessed before those with lower values.	1

3. Apply the values from the `sccconfig.yaml` file:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f sccconfig.yaml --reuse-values
```

This will replace the default values with those specified in the `sccconfig.yaml` file.

Configure additional Trident Protect helm chart settings

You can customize AutoSupport settings and namespace filtering to meet your specific requirements. The following table describes the available configuration parameters:

Parameter	Type	Description
autoSupport.proxy	string	Configures a proxy URL for NetApp AutoSupport connections. Use this to route support bundle uploads through a proxy server. Example: `http://my.proxy.url`.
autoSupport.insecure	boolean	Skips TLS verification for AutoSupport proxy connections when set to `true`. Use only for insecure proxy connections. (default: `false`)

Parameter	Type	Description
autoSupport.enabled	boolean	Enables or disables daily Trident Protect AutoSupport bundle uploads. When set to `false`, scheduled daily uploads are disabled, but you can still manually generate support bundles. (default: `true`)
restoreSkipNamespaceAnnotations	string	Comma-separated list of namespace annotations to exclude from backup and restore operations. Allows you to filter namespaces based on annotations.
restoreSkipNamespaceLabels	string	Comma-separated list of namespace labels to exclude from backup and restore operations. Allows you to filter namespaces based on labels.

You can configure these options using either a YAML configuration file or command-line flags:

Use YAML file

Steps

1. Create a configuration file and name it `values.yaml`.
2. In the file you created, add the configuration options you want to customize.

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
```

3. After you populate the `values.yaml` file with the correct values, apply the configuration file:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

Use CLI flag

Steps

1. Use the following command with the `--set` flag to specify individual parameters:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set-string
restoreSkipNamespaceAnnotations="{annotation1,annotation2}" \
  --set-string restoreSkipNamespaceLabels="{label1,label2}" \
  --reuse-values
```

Restrict Trident Protect pods to specific nodes

You can use the Kubernetes nodeSelector node selection constraint to control which of your nodes are eligible to run Trident Protect pods, based on node labels. By default, Trident Protect is restricted to nodes that are running Linux. You can further customize these constraints depending on your needs.

Steps

1. Create a file named `nodeSelectorConfig.yaml`.
2. Add the nodeSelector option to the file and modify the file to add or change node labels to restrict according to the needs of your environment. For example, the following file contains the default OS restriction, but also targets a specific region and app name:

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. Apply the values from the `nodeSelectorConfig.yaml` file:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

This replaces the default restrictions with those you specified in the `nodeSelectorConfig.yaml` file.

Manage Trident Protect

Manage Trident Protect authorization and access control

Trident Protect uses the Kubernetes model of role-based access control (RBAC). By default, Trident Protect provides a single system namespace and its associated default service account. If you have an organization with many users or specific security needs, you can use the RBAC features of Trident Protect to gain more granular control over access to resources and namespaces.

The cluster administrator always has access to resources in the default `trident-protect` namespace, and can also access resources in all other namespaces. To control access to resources and applications, you need to create additional namespaces and add resources and applications to those namespaces.

Note that no users can create application data management CRs in the default `trident-protect` namespace. You need to create application data management CRs in an application namespace (as a best practice, create application data management CRs in the same namespace as their associated application).

Only administrators should have access to privileged Trident Protect custom resource objects, which include:

- **AppVault**: Requires bucket credential data
- **AutoSupportBundle**: Collects metrics, logs, and other sensitive Trident Protect data
- **AutoSupportBundleSchedule**: Manages log collection schedules

As a best practice, use RBAC to restrict access to privileged objects to administrators.

For more information about how RBAC regulates access to resources and namespaces, refer to the Kubernetes RBAC documentation.

Fore information about service accounts, refer to the Kubernetes service account documentation.

Example: Manage access for two groups of users

For example, an organization has a cluster administrator, a group of engineering users, and a group of marketing users. The cluster administrator would complete the following tasks to create an environment where the engineering group and the marketing group each have access to only the resources assigned to their respective namespaces.

Step 1: Create a namespace to contain resources for each group

Creating a namespace enables you to logically separate resources and better control who has access to those resources.

Steps

1. Create a namespace for the engineering group:

```
kubectl create ns engineering-ns
```

2. Create a namespace for the marketing group:

```
kubectl create ns marketing-ns
```

Step 2: Create new service accounts to interact with resources in each namespace

Each new namespace you create comes with a default service account, but you should create a service account for each group of users so that you can further divide privileges between groups in the future if necessary.

Steps

1. Create a service account for the engineering group:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. Create a service account for the marketing group:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

Step 3: Create a secret for each new service account

A service account secret is used to authenticate with the service account, and can easily be deleted and recreated if compromised.

Steps

1. Create a secret for the engineering service account:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. Create a secret for the marketing service account:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

Step 4: Create a RoleBinding object to bind the ClusterRole object to each new service account

A default ClusterRole object is created when you install Trident Protect. You can bind this ClusterRole to the service account by creating and applying a RoleBinding object.

Steps

1. Bind the ClusterRole to the engineering service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. Bind the ClusterRole to the marketing service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

Step 5: Test permissions

Test that the permissions are correct.

Steps

1. Confirm that engineering users can access engineering resources:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. Confirm that engineering users cannot access marketing resources:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

Step 6: Grant access to AppVault objects

To perform data management tasks such as backups and snapshots, the cluster administrator needs to grant access to AppVault objects to individual users.

Steps

1. Create and apply an AppVault and secret combination YAML file that grants a user access to an AppVault. For example, the following CR grants access to an AppVault to the user `eng-user`:

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. Create and apply a Role CR to enable cluster administrators to grant access to specific resources in a namespace. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. Create and apply a RoleBinding CR to bind the permissions to the user eng-user. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. Verify that the permissions are correct.
   a. Attempt to retrieve AppVault object information for all namespaces:

```
kubectl get appvaults -n trident-protect
--as=system:serviceaccount:engineering-ns:eng-user
```

   You should see output similar to the following:

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. Test to see if the user can get the AppVault information that they now have permission to access:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

You should see output similar to the following:

```
yes
```

Result

The users you have granted AppVault permissions to should be able to use authorized AppVault objects for application data management operations, and should not be able to access any resources outside of the assigned namespaces or create new resources that they do not have access to.

Monitor Trident Protect resources

You can use the kube-state-metrics, Prometheus, and Alertmanager open source tools to monitor the health of the resources protected by Trident Protect.

The kube-state-metrics service generates metrics from Kubernetes API communication. Using it with Trident Protect exposes useful information about the state of resources in your environment.

Prometheus is a toolkit that can ingest the data generated by kube-state-metrics and present it as easily readable information about these objects. Together, kube-state-metrics and Prometheus provide a way for you to monitor the health and status of the resources you are managing with Trident Protect.

Alertmanager is a service that ingests the alerts sent by tools such as Prometheus and routes them to destinations that you configure.

The configurations and guidance included in these steps are only examples; you need to customize them to match your environment. Refer to the following official documentation for specific instructions and support:

- kube-state-metrics documentation
- Prometheus documentation
- Alertmanager documentation

Step 1: Install the monitoring tools

To enable resource monitoring in Trident Protect, you need to install and configure kube-state-metrics, Promethus, and Alertmanager.

Install kube-state-metrics

You can install kube-state-metrics using Helm.

Steps

1. Add the kube-state-metrics Helm chart. For example:

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. Apply the Prometheus ServiceMonitor CRD to the cluster:

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. Create a configuration file for the Helm chart (for example, `metrics-config.yaml`). You can customize the following example configuration to match your environment:

metrics-config.yaml: kube-state-metrics Helm chart configuration

```
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: protect.trident.netapp.io
          kind: "Backup"
          version: "v1"
        labelsFromPath:
          backup_uid: [metadata, uid]
          backup_name: [metadata, name]
          creation_time: [metadata, creationTimestamp]
        metrics:
        - name: backup_info
          help: "Exposes details about the Backup state"
          each:
            type: Info
            info:
              labelsFromPath:
                appVaultReference: ["spec", "appVaultRef"]
                appReference: ["spec", "applicationRef"]
rbac:
  extraRules:
  - apiGroups: ["protect.trident.netapp.io"]
    resources: ["backups"]
    verbs: ["list", "watch"]

# Collect metrics from all namespaces
namespaces: ""

# Ensure that the metrics are collected by Prometheus
prometheus:
  monitor:
    enabled: true
```

4. Install kube-state-metrics by deploying the Helm chart. For example:

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. Configure kube-state-metrics to generate metrics for the custom resources used by Trident Protect by following the instructions in the kube-state-metrics custom resource documentation.

Install Prometheus

You can install Prometheus by following the instructions in the Prometheus documentation.

Install Alertmanager

You can install Alertmanager by following the instructions in the Alertmanager documentation.

Step 2: Configure the monitoring tools to work together

After you install the monitoring tools, you need to configure them to work together.

Steps

1. Integrate kube-state-metrics with Prometheus. Edit the Prometheus configuration file (`prometheus.yaml`) and add the kube-state-metrics service information. For example:

   prometheus.yaml: kube-state-metrics service integration with Prometheus

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: trident-protect
data:
  prometheus.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.trident-protect.svc:8080']
```

2. Configure Prometheus to route alerts to Alertmanager. Edit the Prometheus configuration file (`prometheus.yaml`) and add the following section:

prometheus.yaml: Send alerts to Alertmanager

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

Result

Prometheus can now gather metrics from kube-state-metrics, and can send alerts to Alertmanager. You are now ready to configure what conditions trigger an alert and where the alerts should be sent.

Step 3: Configure alerts and alert destinations

After you configure the tools to work together, you need to configure what type of information triggers alerts, and where the alerts should be sent.

Alert example: backup failure

The following example defines a critical alert that is triggered when the status of the backup custom resource is set to `Error` for 5 seconds or longer. You can customize this example to match your environment, and include this YAML snippet in your `prometheus.yaml` configuration file:

rules.yaml: Define a Prometheus alert for failed backups

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

Configure Alertmanager to send alerts to other channels

You can configure Alertmanager to send notifications to other channels, such as e-mail, PagerDuty, Microsoft Teams, or other notification services by specifying the respective configuration in the `alertmanager.yaml` file.

The following example configures Alertmanager to send notifications to a Slack channel. To customize this example to your environment, replace the value of the `api_url` key with the Slack webhook URL used in your environment:

alertmanager.yaml: Send alerts to a Slack channel

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

Generate a Trident Protect support bundle

Trident Protect enables administrators to generate bundles that include information useful to NetApp Support, including logs, metrics, and topology information about the clusters and apps under management. If you are connected to the internet, you can upload support bundles to the NetApp Support Site (NSS) using a custom resource (CR) file.

Create a support bundle using a CR

Steps

1. Create the custom resource (CR) file and name it (for example, `trident-protect-support-bundle.yaml`).
2. Configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.triggerType**: (*Required*) Determines whether the support bundle is generated immediately, or scheduled. Scheduled bundle generation happens at 12AM UTC. Possible values:
     - Scheduled
     - Manual
   - **spec.uploadEnabled**: (*Optional*) Controls whether the support bundle should be uploaded to the NetApp Support Site after it is generated. If not specified, defaults to `false`. Possible values:
     - true
     - false (default)
   - **spec.dataWindowStart**: (*Optional*) A date string in RFC 3339 format that specifies the date and time that the window of included data in the support bundle should begin. If not specified, defaults to 24 hours ago. The earliest window date you can specify is 7 days ago.

     Example YAML:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: AutoSupportBundle
     metadata:
       name: trident-protect-support-bundle
     spec:
       triggerType: Manual
       uploadEnabled: true
       dataWindowStart: 2024-05-05T12:30:00Z
     ```

3. After you populate the `trident-protect-support-bundle.yaml` file with the correct values, apply the CR:

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

Create a support bundle using the CLI

Steps

1. Create the support bundle, replacing values in brackets with information from your environment. The `trigger-type` determines whether the bundle is created immediately or if creation time is dictated by the schedule, and can be `Manual` or `Scheduled`. The default setting is `Manual`.

For example:

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

Monitor and retrieve the support bundle

After creating a support bundle using either method, you can monitor its generation progress and retrieve it to your local system.

Steps

1. Wait for the `status.generationState` to reach `Completed` state. You can monitor the generation progress with the following command:

   ```
   kubectl get autosupportbundle trident-protect-support-bundle -n trident-protect
   ```

2. Retrieve the support bundle to your local system. Get the copy command from the completed AutoSupport bundle:

   ```
   kubectl describe autosupportbundle trident-protect-support-bundle -n trident-protect
   ```

   Find the `kubectl cp` command from the output and run it, replacing the destination argument with your preferred local directory.

Upgrade Trident Protect

You can upgrade Trident Protect to the latest version to take advantage of new features or bug fixes.

- When you upgrade from version 24.10, snapshots running during the upgrade might fail. This failure does not prevent future snapshots, whether manual or scheduled, from being created. If a snapshot fails during the upgrade, you can manually create a new snapshot to ensure your application is protected.

  To avoid potential failures, you can disable all snapshot schedules before the upgrade and re-enable them afterward. However, this results in missing any scheduled snapshots during the upgrade period.

- For private registry installations, ensure the required Helm chart and images for the target version are available in your private registry, and verify your custom Helm values are compatible with the new chart version. For more information, refer to Install Trident Protect from a private registry.

Step 1: Select a version

Trident Protect versions follow a date-based `YY.MM` naming convention, where "YY" is the last two digits of the year and "MM" is the month. Dot releases follow a `YY.MM.X` convention, where "X" is the patch level. You will select the version to upgrade to based on the version you are upgrading from.

- You can perform a direct upgrade to any target release that is within a four-release window of your installed version. For example, you can directly upgrade from 24.10 (or any 24.10 dot release) to 25.10.
- If you are upgrading from a release outside of the four-release window, perform a multi-step upgrade. Use the upgrade instructions for the earlier version you are upgrading from to upgrade to the most recent release that fits the four-release window. For example, if you are running 24.10 and want to upgrade to 26.02:
  1. First upgrade from 24.10 to 25.02.
  2. Then upgrade from 25.02 to 26.02.

Step 2: Upgrade Trident Protect

To upgrade Trident Protect, perform the following steps.

Steps

1. Update the Trident Helm repository:

```
helm repo update
```

2. Upgrade the Trident Protect CRDs:

> This step is required if you are upgrading from a version earlier than 25.06, as the CRDs are now included in the Trident Protect Helm chart.

   a. Run this command to shift management of CRDs from `trident-protect-crds` to `trident-protect`:

```
kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
{"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
```

   b. Run this command to delete the Helm secret for the `trident-protect-crds` chart:

> Do not uninstall the `trident-protect-crds` chart using Helm, as this could remove your CRDs and any related data.

```
kubectl delete secret -n trident-protect -l name=trident-protect-
crds,owner=helm
```

3. Upgrade Trident Protect:

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2602.0 --namespace trident-protect
```

You can configure the logging level during upgrade by adding `--set logLevel=debug` to the upgrade command. The default logging level is `warn`. Debug logging is recommended for troubleshooting as it helps NetApp support diagnose issues without requiring log level changes or problem reproduction.

Manage and protect applications

Use Trident Protect AppVault objects to manage buckets

The bucket custom resource (CR) for Trident Protect is known as an AppVault. AppVault objects are the declarative Kubernetes workflow representation of a storage bucket. An AppVault CR contains the configurations necessary for a bucket to be used in protection operations, such as backups, snapshots, restore operations, and SnapMirror replication. Only administrators can create AppVaults.

You need to create an AppVault CR manually or from the command line when you perform data protection operations on an application. The AppVault CR is specific to your environment, and you can use the examples on this page as a guide when creating AppVault CRs.

Ensure the AppVault CR is on the cluster where Trident Protect is installed. If the AppVault CR does not exist or you cannot access it, the command line shows an error.

Configure AppVault authentication and passwords

Before you create an AppVault CR, ensure the AppVault and the data mover you choose can authenticate with the provider and any related resources.

Data mover repository passwords

When you create AppVault objects using CRs or the Trident Protect CLI plugin, you can specify a Kubernetes secret with custom passwords for Restic and Kopia encryption. If you don’t specify a secret, Trident Protect uses a default password.

- When manually creating AppVault CRs, use the **spec.dataMoverPasswordSecretRef** field to specify the secret.
- When creating AppVault objects using the Trident Protect CLI, use the `--data-mover-password-secret-ref` argument to specify the secret.

Create a data mover repository password secret

Use the following examples to create the password secret. When you create AppVault objects, you can instruct Trident Protect to use this secret to authenticate with the data mover repository.

- Depending on which data mover you are using, you only need to include the corresponding password for that data mover. For example, if you are using Restic and do not plan to use Kopia in the future, you can include only the Restic password when you create the secret.
- Keep the password in a safe place. You will need it to restore data on the same cluster or a different one. If the cluster or the `trident-protect` namespace is deleted, you will not be able to restore your backups or snapshots without the password.

Use a CR

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

Use the CLI

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

S3-compatible storage IAM permissions

When you access S3-compatible storage such as Amazon S3, Generic S3, StorageGrid S3, or ONTAP S3 using Trident Protect, you need to ensure that the user credentials you provide have the necessary permissions to access the bucket. The following is an example of a policy that grants the minimum required permissions for access with Trident Protect. You can apply this policy to the user that manages S3-compatible bucket policies.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

For more information about Amazon S3 policies, refer to the examples in the Amazon S3 documentation.

EKS Pod Identity for Amazon S3 (AWS) authentication

Trident Protect supports EKS Pod Identity for Kopia data mover operations. This feature enables secure access to S3 buckets without storing AWS credentials in Kubernetes secrets.

Requirements for EKS Pod Identity with Trident Protect

Before using EKS Pod Identity with Trident Protect, ensure the following:

- Your EKS cluster has Pod Identity enabled.
- You have created an IAM role with the necessary S3 bucket permissions. To learn more, refer to S3-compatible storage IAM permissions.
- The IAM role is associated with the following Trident Protect service accounts:
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

For detailed instructions on enabling Pod Identity and associating IAM roles with service accounts, refer to the AWS EKS Pod Identity documentation.

AppVault Configuration
When using EKS Pod Identity, configure your AppVault CR with the `useIAM: true` flag instead of explicit credentials:

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: eks-protect-vault
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-aws
      endpoint: s3.example.com
      useIAM: true
```

AppVault key generation examples for cloud providers

When defining an AppVault CR, you need to include credentials to access the resources hosted by the provider, unless you are using IAM authentication. How you generate the keys for the credentials will differ depending on the provider. The following are command line key generation examples for several providers. You can use the following examples to create keys for the credentials of each cloud provider.

Google Cloud

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

Amazon S3 (AWS)

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

Microsoft Azure

```
kubectl create secret generic <secret-name> \
--from-literal=accountKey=<secret-name> \
-n trident-protect
```

Generic S3

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

ONTAP S3

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

StorageGrid S3

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

AppVault creation examples

The following are example AppVault definitions for each provider.

AppVault CR examples

You can use the following CR examples to create AppVault objects for each cloud provider.

- You can optionally specify a Kubernetes secret that contains custom passwords for the Restic and Kopia repository encryption. Refer to Data mover repository passwords for more information.
- For Amazon S3 (AWS) AppVault objects, you can optionally specify a sessionToken, which is useful if you are using single sign-on (SSO) for authentication. This token is created when you generate keys for the provider in AppVault key generation examples for cloud providers.
- For S3 AppVault objects, you can optionally specify an egress proxy URL for outbound S3 traffic using the `spec.providerConfig.S3.proxyURL` key.

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Amazon S3 (AWS)

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3-secret
```

For EKS environments using Pod Identity with Kopia data mover, you can remove the `providerCredentials` section and add `useIAM: true` under the `s3` configuration instead.

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

Generic S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGrid S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

AppVault creation examples using the Trident Protect CLI

You can use the following CLI command examples to create AppVault CRs for each provider.

- You can optionally specify a Kubernetes secret that contains custom passwords for the Restic and Kopia repository encryption. Refer to Data mover repository passwords for more information.
- For S3 AppVault objects, you can optionally specify an egress proxy URL for outbound S3 traffic using the `--proxy-url <ip_address:port>` argument.

Google Cloud

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Amazon S3 (AWS)

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Generic S3

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

StorageGrid S3

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Supported `providerConfig.s3` configuration options

See the following table for the S3 provider configuration options:

Parameter	Description	Default	Example
`providerConfig.s3.skipCertValidation`	Disable SSL/TLS certificate verification.	false	"true", "false"
`providerConfig.s3.secure`	Enable secure HTTPS communication with the S3 endpoint.	true	"true", "false"
`providerConfig.s3.proxyURL`	Specify the URL of the proxy server used to connect to S3.	None	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	Provide a custom root CA certificate for SSL/TLS verification.	None	"CN=MyCustomCA"
`providerConfig.s3.useIAM`	Enable IAM authentication for accessing S3 buckets. Applicable for EKS Pod Identity.	false	true, false

View AppVault information

You can use the Trident Protect CLI plugin to view information about AppVault objects that you have created on the cluster.

Steps

1. View the contents of an AppVault object:

```
tridentctl-protect get appvaultcontent gcp-vault \
--show-resources all \
-n trident-protect
```

Example output:

```
+-------------+-------+----------+----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+----------------------------
+---------------------------+
```

2. Optionally, to see the AppVaultPath for each resource, use the flag `--show-paths`.

   The cluster name in the first column of the table is only available if a cluster name was specified in the Trident Protect helm installation. For example: `--set clusterName=production1`.

Remove an AppVault

You can remove an AppVault object at any time.

Do not remove the `finalizers` key in the AppVault CR before deleting the AppVault object. If you do so, it can result in residual data in the AppVault bucket and orphaned resources in the cluster.

Before you begin

Ensure that you have deleted all snapshot and backup CRs being used by the AppVault you want to delete.

Remove an AppVault using the Kubernetes CLI

1. Remove the AppVault object, replacing `appvault-name` with the name of the AppVault object to remove:

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

Remove an AppVault using the Trident Protect CLI

1. Remove the AppVault object, replacing `appvault-name` with the name of the AppVault object to remove:

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

Define an application for management with Trident Protect

You can define an application that you want to manage with Trident Protect by creating an application CR and an associated AppVault CR.

Create an AppVault CR

You need to create an AppVault CR that will be used when performing data protection operations on the application, and the AppVault CR needs to reside on the cluster where Trident Protect is installed. The AppVault CR is specific to your environment; for examples of AppVault CRs, refer to AppVault custom resources.

Define an application

You need to define each application that you want to manage with Trident Protect. You can define an application for management by either manually creating an application CR or by using the Trident Protect CLI.

Add an application using a CR

Steps

1. Create the destination application CR file:
   a. Create the custom resource (CR) file and name it (for example, `maria-app.yaml`).
   b. Configure the following attributes:
      - **metadata.name**: (*Required*) The name of the application custom resource. Note the name you choose because other CR files needed for protection operations refer to this value.
      - **spec.includedNamespaces**: (*Required*) Use namespace and label selector to specify the namespaces and resources that the application uses. The application namespace must be part of this list. The label selector is optional and can be used to filter resources within each specified namespace.
      - **spec.includedClusterScopedResources**: (*Optional*) Use this attribute to specify cluster-scoped resources to be included in the application definition. This attribute allows you to select these resources based on their group, version, kind, and labels.
        - **groupVersionKind**: (*Required*) Specifies the API group, version, and kind of the cluster-scoped resource.
        - **labelSelector**: (*Optional*) Filters the cluster-scoped resources based on their labels.
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (*Optional*) This annotation is only applicable to applications defined from virtual machines, such as in KubeVirt environments, where filesystem freezes occur before snapshots. Specify whether this application can write to the filesystem during a snapshot. If set to true, the application ignores the global setting and can write to the filesystem during a snapshot. If set to false, the application ignores the global setting and the filesystem is frozen during a snapshot. If specified but the application has no virtual machines in the application definition, the annotation is ignored. If not specified, the application follows the global Trident Protect freeze setting.

If you need to apply this annotation after an application has already been created, you can use the following command:

```
kubectl annotate application -n <application CR
namespace> <application CR name>
protect.trident.netapp.io/skip-vm-freeze="true"
```

Example YAML:

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

2. (*Optional*) If needed, you can add resource filtering to the same CR to include or exclude specific resources:
   - **Generic filter example:**
     - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
       - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
       - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For

example: `"trident.netapp.io/os=linux"`.

> When both `resourceFilter` and `labelSelector` are used, `resourceFilter` runs first, and then `labelSelector` is applied to the resulting resources.

For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

- **PVC-only filter example:**

  To define a PVC-only application, you must also include `PersistentVolume` and `VolumeSnapshotClass` in the resource filter. Snapshot and backup operations depend on `PersistentVolume` (the cluster-scoped volume bound to each PVC) and `VolumeSnapshotClass` (the snapshot driver), and will fail without them. For example:

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-pvc-app
  namespace: my-app-namespace
spec:
  includedNamespaces:
  - namespace: my-app-namespace
  resourceFilter:
    resourceMatchers:
    - kind: PersistentVolumeClaim
      version: v1
    - kind: PersistentVolume
      version: v1
    - kind: VolumeSnapshotClass
      version: v1
    resourceSelectionCriteria: Include
```

3. After you create the application CR to match your environment, apply the CR. For example:

```
kubectl apply -f maria-app.yaml
```

Add an application using the CLI

Steps

1. Create and apply the application definition using one of the following examples, replacing values in brackets with information from your environment. You can include namespaces and resources in the application definition using comma-separated lists with the arguments shown in the examples.

   You can optionally use an annotation when you create an app to specify whether the application can write to the filesystem during a snapshot. This is only applicable to applications defined from virtual machines, such as in KubeVirt environments, where filesystem freezes occur before snapshots. If you set the annotation to `true`, the application ignores the global setting and can write to the filesystem during a snapshot. If you set it to `false`, the application ignores the global setting and the filesystem is frozen during a snapshot. If you use the annotation but the application has no virtual machines in the application definition, the annotation is ignored. If you don't use the annotation, the application follows the global Trident Protect freeze setting.

   To specify the annotation when you use the CLI to create an application, you can use the `--annotation` flag.

   - Create the application and use the global setting for filesystem freeze behavior:

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

- Create the application and configure the local application setting for filesystem freeze behavior:

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --annotation protect.trident.netapp.io/skip-vm-freeze
=<"true"|"false">
```

- You can use `--resource-filter-include` and `--resource-filter-exclude` flags to include or exclude resources based on `resourceSelectionCriteria` such as group, kind, version, labels, names, and namespaces, as shown in the following example:

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my
-deployment"],"Namespaces":["my
-namespace"],"LabelSelectors":["app=my-app"]}]'
```

- To define a PVC-only application, you must also include `PersistentVolume` and `VolumeSnapshotClass` in the resource filter. Snapshot and backup operations depend on `PersistentVolume` (the cluster-scoped volume bound to each PVC) and `VolumeSnapshotClass` (the snapshot driver), and will fail without them. For example:

```
tridentctl-protect create app my-pvc-app --namespaces <my-app-
namespace> --resource-filter-include
'[{"Kind":"PersistentVolumeClaim","Version":"v1"},{"Kind":"Persis
tentVolume","Version":"v1"},{"Kind":"VolumeSnapshotClass","Versio
n":"v1"}]' -n <my-app-namespace>
```

Protect applications using Trident Protect

You can protect all apps managed by Trident Protect by taking snapshots and backups using an automated protection policy or on an ad-hoc basis.

You can configure Trident Protect to freeze and unfreeze filesystems during data protection operations. Learn more about configuring filesystem freezing with Trident Protect.

Create an on-demand snapshot

You can create an on-demand snapshot at any time.

Cluster-scoped resources are included in a backup, snapshot, or clone if they are explicitly referenced in the application definition or if they have references to any of the application namespaces.

Create a snapshot using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: The Kubernetes name of the application to snapshot.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents (metadata) should be stored.
   - **spec.reclaimPolicy**: (*Optional*) Defines what happens to the AppArchive of a snapshot when the snapshot CR is deleted. This means that even when set to `Retain`, the snapshot will be deleted. Valid options:
     - `Retain` (default)
     - `Delete`

     ```
     apiVersion: protect.trident.netapp.io/v1
     kind: Snapshot
     metadata:
       namespace: my-app-namespace
       name: my-cr-name
     spec:
       applicationRef: my-application
       appVaultRef: appvault-name
       reclaimPolicy: Delete
     ```
3. After you populate the `trident-protect-snapshot-cr.yaml` file with the correct values, apply the CR:

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

Create a snapshot using the CLI

Steps

1. Create the snapshot, replacing values in brackets with information from your environment. For example:

   ```
   tridentctl-protect create snapshot <my_snapshot_name> --appvault
   <my_appvault_name> --app <name_of_app_to_snapshot> -n
   <application_namespace>
   ```

Create an on-demand backup

You can back up an app at any time.

> Cluster-scoped resources are included in a backup, snapshot, or clone if they are explicitly referenced in the application definition or if they have references to any of the application namespaces.

Before you begin

Ensure that the AWS session token expiration is sufficient for any long-running s3 backup operations. If the token expires during the backup operation, the operation can fail.

- Refer to the AWS API documentation for more information about checking the current session token expiration.
- Refer to the AWS IAM documentation for more information about credentials with AWS resources.

Create a backup using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) The Kubernetes name of the application to back up.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents should be stored.
   - **spec.dataMover**: (*Optional*) A string indicating which backup tool to use for the backup operation. Possible values (case sensitive):
     - `Restic`
     - `Kopia` (default)
   - **spec.reclaimPolicy**: (*Optional*) Defines what happens to a backup when released from its claim. Possible values:
     - `Delete`
     - `Retain` (default)
   - **spec.snapshotRef**: (*Optional*): Name of the snapshot to use as the source of the backup. If not provided, a temporary snapshot will be created and backed up.

     Example YAML:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: Backup
     metadata:
       namespace: my-app-namespace
       name: my-cr-name
     spec:
       applicationRef: my-application
       appVaultRef: appvault-name
       dataMover: Kopia
     ```

3. After you populate the `trident-protect-backup-cr.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-backup-cr.yaml
   ```

Create a backup using the CLI

Steps

1. Create the backup, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create backup <my_backup_name> --appvault <my-vault-name> --app <name_of_app_to_back_up> --data-mover <Kopia_or_Restic> -n <application_namespace>
```

You can optionally use the `--full-backup` flag to specify whether a backup should be non-incremental. By default, all backups are incremental. When this flag is used, the backup becomes non-incremental. It is best practice to perform a full backup periodically and then perform incremental backups in between full backups to minimize the risk associated with restores.

Supported backup annotations

The following table describes the annotations you can use when creating a backup CR:

Annotation	Type	Description	Default value
protect.trident.netapp.io/full-backup	string	Specifies whether a backup should be non-incremental. Set to `true` to create a non-incremental backup. It is best practice to perform a full backup periodically and then perform incremental backups in between full backups to minimize the risk associated with restores.	"false"
protect.trident.netapp.io/snapshot-completion-timeout	string	The maximum time allowed for the overall snapshot operation to complete.	"60m"
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	The maximum time allowed for volume snapshots to reach the ready-to-use state.	"30m"
protect.trident.netapp.io/volume-snapshots-created-timeout	string	The maximum time allowed for volume snapshots to be created.	"5m"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	Maximum time (in seconds) to wait for any newly created PersistentVolumeClaims (PVCs) to reach the `Bound` phase before the operations fails.	"1200" (20 minutes)

Create a data protection schedule

A protection policy protects an app by creating snapshots, backups, or both at a defined schedule. You can choose to create snapshots and backups hourly, daily, weekly, and monthly, and you can specify the number of copies to retain. You can schedule a non-incremental full backup by using the full-backup-rule annotation. By default, all backups are incremental. Performing a full backup periodically, along with incremental backups in between, helps reduce the risk associated with restores.

- You can create schedules for snapshots only by setting `backupRetention` to zero and `snapshotRetention` to a value greater than zero. Setting `snapshotRetention` to zero means any scheduled backups will still create snapshots, but those are temporary and get deleted immediately after the backup is completed.
- Cluster-scoped resources are included in a backup, snapshot, or clone if they are explicitly referenced in the application definition or if they have references to any of the application namespaces.

Create a schedule using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-schedule-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.dataMover**: (*Optional*) A string indicating which backup tool to use for the backup operation. Possible values (case sensitive):
     - `Restic`
     - `Kopia` (default)
   - **spec.applicationRef**: The Kubernetes name of the application to back up.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents should be stored.
   - **spec.backupRetention**: (*Required*) The number of backups to retain. Zero indicates that no backups should be created (snapshots only).
   - **spec.backupReclaimPolicy**: (*Optional*) Determines what happens to a backup if the backup CR is deleted during its retention period. After the retention period, backups are always deleted. Possible values (case sensitive):
     - `Retain` (default)
     - `Delete`
   - **spec.snapshotRetention**: (*Required*) The number of snapshots to retain. Zero indicates that no snapshots should be created.
   - **spec.snapshotReclaimPolicy**: (*Optional*) Determines what happens to a snapshot if the snapshot CR is deleted during its retention period. After the retention period, snapshots are always deleted. Possible values (case sensitive):
     - `Retain`
     - `Delete` (default)
   - **spec.granularity**: The frequency at which the schedule should run. Possible values, along with required associated fields:
     - `Hourly` (requires that you specify `spec.minute`)
     - `Daily` (requires that you specify `spec.minute` and `spec.hour`)
     - `Weekly` (requires that you specify `spec.minute, spec.hour`, and `spec.dayOfWeek`)
     - `Monthly` (requires that you specify `spec.minute, spec.hour`, and `spec.dayOfMonth`)
     - `Custom`
   - **spec.dayOfMonth**: (*Optional*) The day of the month (1 - 31) that the schedule should run. This field is required if the granularity is set to `Monthly`. The value must be provided as a string.
   - **spec.dayOfWeek**: (*Optional*) The day of the week (0 - 7) that the schedule should run. Values of 0 or 7 indicate Sunday. This field is required if the granularity is set to `Weekly`. The value must be provided as a string.

- **spec.hour**: (*Optional*) The hour of the day (0 - 23) that the schedule should run. This field is required if the granularity is set to `Daily`, `Weekly`, or `Monthly`. The value must be provided as a string.
- **spec.minute**: (*Optional*) The minute of the hour (0 - 59) that the schedule should run. This field is required if the granularity is set to `Hourly`, `Daily`, `Weekly`, or `Monthly`. The value must be provided as a string.
- **spec.runImmediately**: (*Optional*) Set to `true` to trigger a one-time, immediate baseline run (backup and/or snapshot per retention settings) when the schedule is created. Defaults to `false`. This does not modify subsequent recurrence.

  Example YAML for backup and snapshot schedule:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: Daily
  hour: "0"
  minute: "0"
```

  Example YAML for snapshot-only schedule:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

Example YAML for schedule with immediate run:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-daily-schedule-run-immediately
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "7"
  snapshotRetention: "7"
  granularity: Daily
  hour: "3"
  minute: "0"
  runImmediately: true
```

3. After you populate the `trident-protect-schedule-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

Create a schedule using the CLI

Steps

1. Create the protection schedule, replacing values in brackets with information from your environment. For example:

> You can use `tridentctl-protect create schedule --help` to view detailed help information for this command.

```
tridentctl-protect create schedule <my_schedule_name> \
  --appvault <my_appvault_name> \
  --app <name_of_app_to_snapshot> \
  --backup-retention <how_many_backups_to_retain> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_to_run_schedule> \
  --granularity <frequency_to_run> \
  --hour <hour_of_day_to_run> \
  --minute <minute_of_hour_to_run> \
  --recurrence-rule <recurrence> \
  --snapshot-retention <how_many_snapshots_to_retain> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

The following flags provide additional control over your schedule:

- **Full backup scheduling**: Use the `--full-backup-rule` flag to schedule non-incremental full backups. This flag only works with `--granularity Daily`. Possible values:
  - `Always`: Create a full backup every day.
  - Specific weekdays: Specify one or more days separated by commas (for example, `"Monday,Thursday"`). Valid values: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.

  > The `--full-backup-rule` flag does not work with Hourly, Weekly, or Monthly granularity.

- **Immediate baseline protection**: Use `--run-immediately true` to create an initial backup or snapshot immediately when the schedule is created, rather than waiting for the first scheduled run time. Default is `false`.
- **Snapshot-only schedules**: Set `--backup-retention 0` and specify a value greater than zero for `--snapshot-retention`.

Supported schedule annotations

The following table describes the annotations you can use when creating a schedule CR:

Annotation	Type	Description	Default value
protect.trident.netapp.io/full-backup-rule	string	Specifies the rule for scheduling full backups. You can set it to `Always` for constant full backup or customize it based on your requirements. For example, if you choose daily granularity, you can specify the weekdays on which full backup should occur (for example, `"Monday,Thursday"`). Valid weekday values are: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday. Note that this annotation can only be used with schedules that have `granularity` set to `Daily`.	Not set (all backups are incremental)
protect.trident.netapp.io/snapshot-completion-timeout	string	The maximum time allowed for the overall snapshot operation to complete.	"60m"
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	The maximum time allowed for volume snapshots to reach the ready-to-use state.	"30m"
protect.trident.netapp.io/volume-snapshots-created-timeout	string	The maximum time allowed for volume snapshots to be created.	"5m"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	Maximum time (in seconds) to wait for any newly created PersistentVolumeClaims (PVCs) to reach the `Bound` phase before the operations fails.	"1200" (20 minutes)

Delete a snapshot

Delete the scheduled or on-demand snapshots that you no longer need.

Steps

1. Remove the snapshot CR associated with the snapshot:

```
kubectl delete snapshot <snapshot_name> -n my-app-namespace
```

Delete a backup

Delete the scheduled or on-demand backups that you no longer need.

Ensure the reclaim policy is set to `Delete` to remove all backup data from object storage. The default setting of the policy is `Retain` to avoid accidental data loss. If the policy is not changed to `Delete`, the backup data will remain in object storage and will require manual deletion.

Steps

1. Remove the backup CR associated with the backup:

```
kubectl delete backup <backup_name> -n my-app-namespace
```

Check the status of a backup operation

You can use the command line to check the status of a backup operation that is in progress, has completed, or has failed.

Steps

1. Use the following command to retrieve status of the backup operation, replacing values in brackes with information from your environment:

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

Enable backup and restore for azure-netapp-files (ANF) operations

If you have installed Trident Protect, you can enable space-efficient backup and restore functionality for storage backends that use the azure-netapp-files storage class and were created prior to Trident 24.06. This funtionality works with NFSv4 volumes and does not consume additional space from the capacity pool.

Before you begin

Ensure the following:

- You have installed Trident Protect.
- You have defined an application in Trident Protect. This application will have limited protection functionality until you complete this procedure.
- You have `azure-netapp-files` selected as the default storage class for your storage backend.

Expand for configuration steps

1. Do the following in Trident if the ANF volume was created prior to upgrading to Trident 24.10:
   a. Enable the snapshot directory for each PV that is azure-netapp-files based and associated with the application:

```
tridentctl update volume <pv name> --snapshot-dir=true -n trident
```

   b. Confirm that the snapshot directory has been enabled for each associated PV:

```
tridentctl get volume <pv name> -n trident -o yaml | grep
snapshotDir
```

   Response:

```
snapshotDirectory: "true"
```

   When the snapshot directory is not enabled, Trident Protect chooses the regular backup functionality, which temporarily consumes space in the capacity pool during the backup process. In this case, ensure that sufficient space is available in the capacity pool to create a temporary volume of the size of the volume being backed up.

Result

The application is ready for backup and restore using Trident Protect. Each PVC is also available to be used by other applications for backups and restores.

Restore applications

Restore applications using Trident Protect

You can use Trident Protect to restore your application from a snapshot or backup. Restoring from an existing snapshot will be faster when restoring the application to the same cluster.

- When you restore an application, all execution hooks configured for the application are restored with the app. If a post-restore execution hook is present, it runs automatically as part of the restore operation.
- Restoring from a backup to a different namespace or to the original namespace is supported for qtree volumes. However, restoring from a snapshot to a different namespace or to the original namespace is not supported for qtree volumes.
- You can use advanced settings to customize restore operations. To learn more, refer to Use advanced Trident Protect restore settings.

Restore from a backup to a different namespace

When you restore a backup to a different namespace using a BackupRestore CR, Trident Protect restores the application in a new namespace and creates an application CR for the restored application. To protect the restored application, create on-demand backups or snapshots, or establish a protection schedule.

- Restoring a backup to a different namespace with existing resources will not alter any resources that share names with those in the backup. To restore all resources in the backup, either delete and re-create the target namespace, or restore the backup to a new namespace.
- When using a CR to restore to a new namespace, you must manually create the destination namespace before applying the CR. Trident Protect automatically creates namespaces only when using the CLI.

Before you begin

Ensure that the AWS session token expiration is sufficient for any long-running s3 restore operations. If the token expires during the restore operation, the operation can fail.

- Refer to the AWS API documentation for more information about checking the current session token expiration.
- Refer to the AWS IAM documentation for more information about credentials with AWS resources.

When you restore backups using Kopia as the data mover, you can optionally specify annotations in the CR or using the CLI to control the behavior of the temporary storage used by Kopia. Refer to the Kopia documentation for more information about the options you can configure. Use the `tridentctl-protect create --help` command for more information about specifying annotations with the Trident Protect CLI.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

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

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.
   - **spec.destinationApplicationName**: (*Optional*) The name for the restored application. If provided, the restored application uses this name. If not provided, the restored application uses the source application name.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       destinationApplicationName: my-new-app-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:

   > Trident Protect selects some resources automatically because of their relationship with resources that you select. For example, if you select a persistent volume claim resource and it has an associated pod, Trident Protect will also restore the associated pod.

   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers

parameters to define the resources to be included or excluded:

- **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
  - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
  - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
  - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
  - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
  - **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
  - **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

    For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-backup-restore-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

Use the CLI

Steps

1. Restore the backup to a different namespace, replacing values in brackets with information from your environment. The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format

`source1:dest1,source2:dest2`. For example:

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

Restore from a backup to the original namespace

You can restore a backup to the original namespace at any time. When you perform an in-place restore, Trident Protect automatically manages protection schedules and in-progress operations to prevent invalid recovery points:

- All enabled protection schedules for the application are disabled before the restore begins. This prevents scheduled backups or snapshots from running while the application resources are being restored.
- After the restore completes successfully, only the schedules that were enabled before the restore are re-enabled. Schedules that were already disabled remain disabled.
- Any in-progress backup or snapshot operations are cancelled before the restore begins. If an operation does not cancel within 5 minutes, the restore proceeds and logs a warning in the restore CR status.

Before you begin

Ensure that the AWS session token expiration is sufficient for any long-running s3 restore operations. If the token expires during the restore operation, the operation can fail.

- Refer to the AWS API documentation for more information about checking the current session token expiration.
- Refer to the AWS IAM documentation for more information about credentials with AWS resources.

When you restore backups using Kopia as the data mover, you can optionally specify annotations in the CR or using the CLI to control the behavior of the temporary storage used by Kopia. Refer to the Kopia documentation for more information about the options you can configure. Use the `tridentctl-protect create --help` command for more information about specifying annotations with the Trident Protect CLI.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-ipr-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

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

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.

     For example:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupInplaceRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:

   > Trident Protect selects some resources automatically because of their relationship with resources that you select. For example, if you select a persistent volume claim resource and it has an associated pod, Trident Protect will also restore the associated pod.

   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.

- **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-backup-ipr-cr.yaml` file with the correct values, apply the CR:

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

Use the CLI

Steps

1. Restore the backup to the original namespace, replacing values in brackets with information from your environment. The `backup` argument uses a namespace and backup name in the format `<namespace>/<name>`. For example:

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

Restore from a backup to a different cluster

You can restore a backup to a different cluster if there is an issue with the original cluster.

- When you restore backups using Kopia as the data mover, you can optionally specify annotations in the CR or using the CLI to control the behavior of the temporary storage used by Kopia. Refer to the Kopia documentation for more information about the options you can configure. Use the `tridentctl-protect create --help` command for more information about specifying annotations with the Trident Protect CLI.
- When using a CR to restore to a new namespace, you must manually create the destination namespace before applying the CR. Trident Protect automatically creates namespaces only when using the CLI.

Before you begin

Ensure the following prerequisites are met:

- The destination cluster has Trident Protect installed.
- The destination cluster has access to the bucket path of the same AppVault as the source cluster, where the backup is stored.
- Ensure that your local environment can connect to the object storage bucket defined in the AppVault CR when running the `tridentctl-protect get appvaultcontent` command. If network restrictions prevent access, run the Trident Protect CLI from within a pod on the destination cluster instead.
- Ensure that the AWS session token expiration is sufficient for any long-running restore operations. If the token expires during the restore operation, the operation can fail.
  - Refer to the AWS API documentation for more information about checking the current session token expiration.
  - Refer to the AWS documentation for more information about credentials with AWS resources.

Steps

1. Verify that the AppVault CR exists on the destination cluster using the Trident Protect CLI plugin:

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

   If the AppVault CR does not exist on the destination cluster, create it by following the steps in Use Trident Protect AppVault objects to manage buckets.

2. View the backup contents of the available AppVault on the destination cluster, and note `appArchivePath` of the backup you want to restore:

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

   Running this command displays the available backups in the AppVault, including their originating clusters, corresponding application names, timestamps, and archive paths.

Example output:

```
+------------+----------+--------+----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |        TIMESTAMP
|    PATH     |
+------------+----------+--------+----------------
+--------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+------------+----------+--------+----------------
+--------------------------+-------------+
```

3. Restore the application to the destination cluster using the AppVault name and archive path:

> When using a CR, ensure that the namespace intended for the application restore exists on the destination cluster.

Use a CR

4. Create the custom resource (CR) file and name it `trident-protect-backup-restore-cr.yaml`.
5. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.
   - **spec.appArchivePath**: (*Required*) The path inside AppVault where the backup contents are stored. Use the command from step 2 to view the backup contents and find `appArchivePath` for the backup you want to restore.
   - **spec.destinationApplicationName**: (*Optional*) The name for the restored application. If provided, the restored application uses this name. If not provided, the restored application uses the source application name.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

     For example:

     ```
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-backup-path
       destinationApplicationName: my-new-app-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

6. After you populate the `trident-protect-backup-restore-cr.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-backup-restore-cr.yaml
   ```

Use the CLI

4. Use the following command to restore the application, replacing values in brackets with information from your environment. The namespace-mapping argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format source1:dest1,source2:dest2. For example:

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--destination-app-name <custom_app_name> \
--context <destination_cluster_name> \
-n <application_namespace>
```

Restore from a snapshot to a different namespace

You can restore data from a snapshot using a custom resource (CR) file either to a different namespace or the original source namespace. When you restore a snapshot to a different namespace using a SnapshotRestore CR, Trident Protect restores the application in a new namespace and creates an application CR for the restored application. To protect the restored application, create on-demand backups or snapshots, or establish a protection schedule.

- SnapshotRestore supports the `spec.storageClassMapping` attribute, but only when the source and destination storage classes use the same storage backend. If you attempt to restore to a `StorageClass` that uses a different storage backend, the restore operation will fail.
- When using a CR to restore to a new namespace, you must manually create the destination namespace before applying the CR. Trident Protect automatically creates namespaces only when using the CLI.

Before you begin

Ensure that the AWS session token expiration is sufficient for any long-running s3 restore operations. If the token expires during the restore operation, the operation can fail.

- Refer to the AWS API documentation for more information about checking the current session token expiration.
- Refer to the AWS IAM documentation for more information about credentials with AWS resources.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

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

   - **spec.destinationApplicationName**: (*Optional*) The name for the restored application. If provided, the restored application uses this name. If not provided, the restored application uses the source application name.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

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

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:

   > Trident Protect selects some resources automatically because of their relationship with resources that you select. For example, if you select a persistent volume claim resource and it has an associated pod, Trident Protect will also restore the associated pod.

   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers

parameters to define the resources to be included or excluded:

- **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
  - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
  - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
  - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
  - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
  - **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
  - **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

    For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-restore-cr.yaml` file with the correct values, apply the CR:

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

Use the CLI

Steps

1. Restore the snapshot to a different namespace, replacing values in brackets with information from your environment.

- The `snapshot` argument uses a namespace and snapshot name in the format `<namespace>/<name>`.
- The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format `source1:dest1,source2:dest2`.

  For example:

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name <custom_app_name> \
-n <application_namespace>
```

Restore from a snapshot to the original namespace

You can restore a snapshot to the original namespace at any time. When you perform an in-place restore, Trident Protect automatically manages protection schedules and in-progress operations to prevent invalid recovery points:

- All enabled protection schedules for the application are disabled before the restore begins. This prevents scheduled backups or snapshots from running while the application resources are being restored.
- After the restore completes successfully, only the schedules that were enabled before the restore are re-enabled. Schedules that were already disabled remain disabled.
- Any in-progress backup or snapshot operations are cancelled before the restore begins. If an operation does not cancel within 5 minutes, the restore proceeds and logs a warning in the restore CR status.

Before you begin

Ensure that the AWS session token expiration is sufficient for any long-running s3 restore operations. If the token expires during the restore operation, the operation can fail.

- Refer to the AWS API documentation for more information about checking the current session token expiration.
- Refer to the AWS IAM documentation for more information about credentials with AWS resources.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-ipr-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

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

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotInplaceRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:

   > Trident Protect selects some resources automatically because of their relationship with resources that you select. For example, if you select a persistent volume claim resource and it has an associated pod, Trident Protect will also restore the associated pod.

   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
       - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.

- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-ipr-cr.yaml` file with the correct values, apply the CR:

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

Use the CLI

Steps

1. Restore the snapshot to the original namespace, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
-n <application_namespace>
```

Check the status of a restore operation

You can use the command line to check the status of a restore operation that is in progress, has completed, or has failed.

Steps

1. Use the following command to retrieve status of the restore operation, replacing values in brackes with information from your environment:

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

Use advanced Trident Protect restore settings

You can customize restore operations using advanced settings such as annotations, namespace settings, and storage options to meet your specific requirements.

Namespace annotations and labels during restore and failover operations

During restore and failover operations, labels and annotations in the destination namespace are made to match the labels and annotations in the source namespace. Labels or annotations from the source namespace that don't exist in the destination namespace are added, and any labels or annotations that already exist are overwritten to match the value from the source namespace. Labels or annotations that exist only on the destination namespace remain unchanged.

If you use Red Hat OpenShift, it's important to note the critical role of namespace annotations in OpenShift environments. Namespace annotations ensure that restored pods adhere to the appropriate permissions and security configurations defined by OpenShift security context constraints (SCCs) and can access volumes without permission issues. For more information, refer to the OpenShift security context constraints documentation.

You can prevent specific annotations in the destination namespace from being overwritten by setting the Kubernetes environment variable `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` before you perform the restore or failover operation. For example:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

When performing restore or failover operation, any namespace annotations and labels specified in `restoreSkipNamespaceAnnotations` and `restoreSkipNamespaceLabels` are excluded from the restore or failover operation. Ensure these settings are configured during the initial Helm installation. To learn more, refer to Configure additional Trident Protect helm chart settings.

If you installed the source application using Helm with the `--create-namespace` flag, special treatment is given to the `name` label key. During the restore or failover process, Trident Protect copies this label to the destination namespace, but updates the value to the destination namespace value if the value from source matches the source namespace. If this value doesn't match the source namespace it is copied to the

destination namespace with no changes.

Example

The following example presents a source and destination namespace, each with different annotations and labels. You can see the state of the destination namespace before and after the operation, and how the annotations and labels are combined or overwritten in the destination namespace.

Before the restore or failover operation

The following table illustrates the state of the example source and destination namespaces before the restore or failover operation:

Namespace	Annotations	Labels
Namespace ns-1 (source)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• environment=production • compliance=hipaa • name=ns-1
Namespace ns-2 (destination)	• annotation.one/key: "true" • annotation.three/key: "false"	• role=database

After the restore operation

The following table illustrates the state of the example destination namespace after the restore or failover operation. Some keys have been added, some have been overwritten, and the `name` label has been updated to match the destination namespace:

Namespace	Annotations	Labels
Namespace ns-2 (destination)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• name=ns-2 • compliance=hipaa • environment=production • role=database

Supported fields

This section describes additional fields available for restore operations.

Storage class mapping

The `spec.storageClassMapping` attribute defines a mapping from a storage class present in the source application to a new storage class on the target cluster. You can use this when migrating applications between clusters with different storage classes or when changing the storage backend for BackupRestore operations.

Example:

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

Supported annotations

This section lists the supported annotations for configuring various behaviors in the system. If an annotation is not explicitly set by the user, the system will use the default value.

Annotation	Type	Description	Default value
protect.trident.netapp.io/data-mover-timeout-sec	string	The maximum time (in seconds) allowed for data mover operation to be stalled.	"300"
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	string	The maximum size limit (in megabytes) for the Kopia content cache.	"1000"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	Maximum time (in seconds) to wait for any newly created PersistentVolumeClaims (PVCs) to reach the `Bound` phase before the operations fails. Applies to all restore CR types (BackupRestore, BackupInplaceRestore, SnapshotRestore, SnapshotInplaceRestore). Use a higher value if your storage backend or cluster often requires more time.	"1200" (20 minutes)

Replicate applications using NetApp SnapMirror and Trident Protect

Using Trident Protect, you can use the asynchronous replication capabilities of NetApp SnapMirror technology to replicate data and application changes from one storage backend to another, on the same cluster or between different clusters.

Namespace annotations and labels during restore and failover operations

During restore and failover operations, labels and annotations in the destination namespace are made to match the labels and annotations in the source namespace. Labels or annotations from the source namespace that don't exist in the destination namespace are added, and any labels or annotations that already exist are overwritten to match the value from the source namespace. Labels or annotations that exist only on the destination namespace remain unchanged.

If you use Red Hat OpenShift, it's important to note the critical role of namespace annotations in OpenShift environments. Namespace annotations ensure that restored pods adhere to the appropriate permissions and security configurations defined by OpenShift security context constraints (SCCs) and can access volumes without permission issues. For more information, refer to the OpenShift security context constraints documentation.

You can prevent specific annotations in the destination namespace from being overwritten by setting the Kubernetes environment variable `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` before you perform the restore or failover operation. For example:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

When performing restore or failover operation, any namespace annotations and labels specified in `restoreSkipNamespaceAnnotations` and `restoreSkipNamespaceLabels` are excluded from the restore or failover operation. Ensure these settings are configured during the initial Helm installation. To learn more, refer to Configure additional Trident Protect helm chart settings.

If you installed the source application using Helm with the `--create-namespace` flag, special treatment is given to the `name` label key. During the restore or failover process, Trident Protect copies this label to the destination namespace, but updates the value to the destination namespace value if the value from source matches the source namespace. If this value doesn't match the source namespace it is copied to the destination namespace with no changes.

Example

The following example presents a source and destination namespace, each with different annotations and labels. You can see the state of the destination namespace before and after the operation, and how the annotations and labels are combined or overwritten in the destination namespace.

Before the restore or failover operation

The following table illustrates the state of the example source and destination namespaces before the restore or failover operation:

Namespace	Annotations	Labels
Namespace ns-1 (source)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• environment=production • compliance=hipaa • name=ns-1
Namespace ns-2 (destination)	• annotation.one/key: "true" • annotation.three/key: "false"	• role=database

After the restore operation

The following table illustrates the state of the example destination namespace after the restore or failover operation. Some keys have been added, some have been overwritten, and the `name` label has been updated to match the destination namespace:

Namespace	Annotations	Labels
Namespace ns-2 (destination)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• name=ns-2 • compliance=hipaa • environment=production • role=database

You can configure Trident Protect to freeze and unfreeze filesystems during data protection operations. Learn more about configuring filesystem freezing with Trident Protect.

Execution hooks during failover and reverse operations

When using AppMirror relationship to protect your application, there are specific behaviors related to execution hooks that you should be aware of during failover and reverse operations.

- During failover, the execution hooks are automatically copied from the source cluster to the destination cluster. You do not need to manually recreate them. After failover, execution hooks are present on the application and will execute during any relevant actions.
- During reverse or reverse resync, any existing execution hooks on the application are removed. When the source application becomes the destination application, these execution hooks are not valid and are deleted to prevent their execution.

To learn more about execution hooks, refer to Manage Trident Protect execution hooks.

Set up a replication relationship

Setting up a replication relationship involves the following:

- Choosing how frequently you want Trident Protect to take an app snapshot (which includes the app's Kubernetes resources as well as the volume snapshots for each of the app's volumes)
- Choosing the replication schedule (includes Kubernetes resources as well as persistent volume data)
- Setting the time for the snapshot to be taken

Steps

1. On the source cluster, create an AppVault for the source application. Depending on your storage provider, modify an example in AppVault custom resources to fit your environment:

Create an AppVault using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-appvault-primary-source.yaml`).

b. Configure the following attributes:

- **metadata.name**: (*Required*) The name of the AppVault custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
- **spec.providerConfig**: (*Required*) Stores the configuration necessary to access the AppVault using the specified provider. Choose a bucketName and any other necessary details for your provider. Make note of the values you choose, because other CR files needed for a replication relationship refer to these values. Refer to AppVault custom resources for examples of AppVault CRs with other providers.
- **spec.providerCredentials**: (*Required*) Stores references to any credential required to access the AppVault using the specified provider.
  - **spec.providerCredentials.valueFromSecret**: (*Required*) Indicates that the credential value should come from a secret.
    - **key**: (*Required*) The valid key of the secret to select from.
    - **name**: (*Required*) Name of the secret containing the value for this field. Must be in the same namespace.
  - **spec.providerCredentials.secretAccessKey**: (*Required*) The access key used to access the provider. The **name** should match **spec.providerCredentials.valueFromSecret.name**.
- **spec.providerType**: (*Required*) Determines what provides the backup; for example, NetApp ONTAP S3, generic S3, Google Cloud, or Microsoft Azure. Possible values:
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. After you populate the `trident-protect-appvault-primary-source.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

Create an AppVault using the CLI

a. Create the AppVault, replacing values in brackets with information from your environment:

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

2. On the source cluster, create the source application CR:

Create the source application using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-app-source.yaml`).

b. Configure the following attributes:

- **metadata.name**: (*Required*) The name of the application custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
- **spec.includedNamespaces**: (*Required*) An array of namespaces and associated labels. Use namespace names and optionally narrow the scope of the namespaces with labels to specify resources that exist in the namespaces listed here. The application namespace must be part of this array.

  Example YAML:

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

c. After you populate the `trident-protect-app-source.yaml` file with the correct values, apply the CR:

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

Create the source application using the CLI

a. Create the source application. For example:

```
tridentctl-protect create app <my-app-name> --namespaces <namespaces-to-be-included> -n <my-app-namespace>
```

3. Optionally, on the source cluster, take a snapshot of the source application. This snapshot is used as the basis for the application on the destination cluster. If you skip this step, you'll need to wait for the next scheduled snapshot to run so that you have a recent snapshot. To create an on-demand snapshot, refer to Create an on-demand snapshot.
4. On the source cluster, create the replication schedule CR:

Alongside the schedule provided below, it is recommended to create a separate daily snapshot schedule with a retention period of 7 days to maintain a common snapshot between peered ONTAP clusters. This ensures that snapshots are available for up to 7 days, but the retention period can be customized based on user requirements.

If a failover happens, the system can use these snapshots for up to 7 days for reverse operations. This approach makes the reverse process faster and more efficient because only the changes made since the last snapshot will be transferred, not all the data.

If an existing schedule for the application already meets the desired retention requirements, no additional schedules are required.

Create the replication schedule using a CR

a. Create a replication schedule for the source application:

i. Create the custom resource (CR) file and name it (for example, `trident-protect-schedule.yaml`).

ii. Configure the following attributes:

- **metadata.name**: (*Required*) The name of the schedule custom resource.
- **spec.appVaultRef**: (*Required*) This value must match the metadata.name field of the AppVault for the source application.
- **spec.applicationRef**: (*Required*) This value must match the metadata.name field of the source application CR.
- **spec.backupRetention**: (*Required*) This field is required, and the value must be set to 0.
- **spec.enabled**: Must be set to true.
- **spec.granularity**: Must be set to `Custom`.
- **spec.recurrenceRule**: Define a start date in UTC time and a recurrence interval.
- **spec.snapshotRetention**: Must be set to 2.

Example YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

iii. After you populate the `trident-protect-schedule.yaml` file with the correct values, apply the CR:

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

Create the replication schedule using the CLI

a. Create the replication schedule, replacing values in brackets with information from your environment:

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

Example:

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. On the destination cluster, create a source application AppVault CR that is identical to the AppVault CR you applied on the source cluster and name it (for example, `trident-protect-appvault-primary-destination.yaml`).
6. Apply the CR:

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

7. Create a destination AppVault CR for the destination application on the destination cluster. Depending on your storage provider, modify an example in AppVault custom resources to fit your environment:
   a. Create the custom resource (CR) file and name it (for example, `trident-protect-appvault-secondary-destination.yaml`).
   b. Configure the following attributes:
      - **metadata.name**: (*Required*) The name of the AppVault custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
      - **spec.providerConfig**: (*Required*) Stores the configuration necessary to access the AppVault using the specified provider. Choose a `bucketName` and any other necessary details for your provider. Make note of the values you choose, because other CR files needed for a replication relationship refer to these values. Refer to AppVault custom resources for examples of AppVault CRs with other providers.
      - **spec.providerCredentials**: (*Required*) Stores references to any credential required to access the AppVault using the specified provider.
        - **spec.providerCredentials.valueFromSecret**: (*Required*) Indicates that the credential value should come from a secret.
          - **key**: (*Required*) The valid key of the secret to select from.

        - **name**: (*Required*) Name of the secret containing the value for this field. Must be in the same namespace.
      - **spec.providerCredentials.secretAccessKey**: (*Required*) The access key used to access the provider. The **name** should match **spec.providerCredentials.valueFromSecret.name**.
    - **spec.providerType**: (*Required*) Determines what provides the backup; for example, NetApp ONTAP S3, generic S3, Google Cloud, or Microsoft Azure. Possible values:
      - aws
      - azure
      - gcp
      - generic-s3
      - ontap-s3
      - storagegrid-s3

c. After you populate the `trident-protect-appvault-secondary-destination.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n trident-protect
```

8. On the destination cluster, create an AppMirrorRelationship CR file.

> When using a CR, manually create the destination namespace before applying the CR. Trident Protect automatically creates namespaces only when using the CLI.

Create an AppMirrorRelationship using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-relationship.yaml`).

b. Configure the following attributes:

- **metadata.name:** (Required) The name of the AppMirrorRelationship custom resource.
- **spec.destinationAppVaultRef**: (*Required*) This value must match the name of the AppVault for the destination application on the destination cluster.
- **spec.namespaceMapping**: (*Required*) The destination and source namespaces must match the application namespace defined in the respective application CR.
- **spec.sourceAppVaultRef**: (*Required*) This value must match the name of the AppVault for the source application.
- **spec.sourceApplicationName**: (*Required*) This value must match the name of the source application you defined in the source application CR.
- **spec.sourceApplicationUID**: (Required) This value must match the UID of the source application you defined in the source application CR.
- **spec.storageClassName**: (*Optional*) Choose the name of a valid storage class on the cluster. The storage class must be linked to an ONTAP storage VM that is peered with the source environment. If the storage class is not provided, the default storage class on the cluster will be used by default.
- **spec.recurrenceRule**: Define a start date in UTC time and a recurrence interval.

  Example YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-
bucket-8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. After you populate the `trident-protect-relationship.yaml` file with the correct values, apply the CR:

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

Create an AppMirrorRelationship using the CLI

a. Create and apply the AppMirrorRelationship object, replacing values in brackets with information from your environment:

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --source-app-vault <my_vault_name> --recurrence
-rule <rule> --namespace-mapping <ns_mapping> --source-app-id
<source_app_UID> --source-app <my_source_app_name> --storage
-class <storage_class_name> -n <application_namespace>
```

Example:

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. (*Optional*) On the destination cluster, check the state and status of the replication relationship:

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

Fail over to destination cluster

Using Trident Protect, you can fail over replicated applications to a destination cluster. This procedure stops the replication relationship and brings the app online on the destination cluster. Trident Protect does not stop the app on the source cluster if it was operational.

Steps

1. On the destination cluster, edit the AppMirrorRelationship CR file (for example, `trident-protect-relationship.yaml`) and change the value of **spec.desiredState** to `Promoted`.
2. Save the CR file.
3. Apply the CR:

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

4. (*Optional*) Create any protection schedules that you need on the failed over application.
5. (*Optional*) Check the state and status of the replication relationship:

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

Resync a failed over replication relationship

The resync operation re-establishes the replication relationship. After you perform a resync operation, the original source application becomes the running application, and any changes made to the running application on the destination cluster are discarded.

The process stops the app on the destination cluster before re-establishing replication.

Any data written to the destination application during failover will be lost.

Steps

1. Optional: On the source cluster, create a snapshot of the source application. This ensures that the latest changes from the source cluster are captured.
2. On the destination cluster, edit the AppMirrorRelationship CR file (for example, `trident-protect-relationship.yaml`) and change the value of spec.desiredState to `Established`.
3. Save the CR file.
4. Apply the CR:

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

5. If you created any protection schedules on the destination cluster to protect the failed over application, remove them. Any schedules that remain cause volume snapshot failures.

Reverse resync a failed over replication relationship

When you reverse resync a failed over replication relationship, the destination application becomes the source application, and the source becomes the destination. Changes made to the destination application during failover are kept.

Steps

1. On the original destination cluster, delete the AppMirrorRelationship CR. This causes the destination to become the source. If there are any protection schedules remaining on the new destination cluster, remove them.
2. Set up a replication relationship by applying the CR files you originally used to set up the relationship to the opposite clusters.
3. Ensure the new destination (original source cluster) is configured with both AppVault CRs.
4. Set up a replication relationship on the opposite cluster, configuring values for the reverse direction.

Reverse application replication direction

When you reverse replication direction, Trident Protect moves the application to the destination storage backend while continuing to replicate back to the original source storage backend. Trident Protect stops the source application and replicates the data to the destination before failing over to the destination app.

In this situation, you are swapping the source and destination.

Steps

1. On the source cluster, create a shutdown snapshot:

Create a shutdown snapshot using a CR

a. Disable the protection policy schedules for the source application.

b. Create a ShutdownSnapshot CR file:

   i. Create the custom resource (CR) file and name it (for example, `trident-protect-shutdownsnapshot.yaml`).

   ii. Configure the following attributes:

   - **metadata.name**: (*Required*) The name of the custom resource.
   - **spec.AppVaultRef**: (*Required*) This value must match the metadata.name field of the AppVault for the source application.
   - **spec.ApplicationRef**: (*Required*) This value must match the metadata.name field of the source application CR file.

   Example YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-
86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-
04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. After you populate the `trident-protect-shutdownsnapshot.yaml` file with the correct values, apply the CR:

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

Create a shutdown snapshot using the CLI

a. Create the shutdown snapshot, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot>
--appvault <my_vault> --app <app_to_snapshot> -n
<application_namespace>
```

2. On the source cluster, after the shutdown snapshot completes, get the status of the shutdown snapshot:

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

3. On the source cluster, find the value of **shutdownsnapshot.status.appArchivePath** using the following command, and record the last part of the file path (also called the basename; this will be everything after the last slash):

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

4. Perform a fail over from the new destination cluster to the new source cluster, with the following change:

> In step 2 of the fail over procedure, include the `spec.promotedSnapshot` field in the AppMirrorRelationship CR file, and set its value to the basename you recorded in step 3 above.

5. Perform the reverse resync steps in Reverse resync a failed over replication relationship.
6. Enable protection schedules on the new source cluster.

Result

The following actions occur because of the reverse replication:

- A snapshot is taken of the original source app's Kubernetes resources.
- The original source app's pods are gracefully stopped by deleting the app's Kubernetes resources (leaving PVCs and PVs in place).
- After the pods are shut down, snapshots of the app's volumes are taken and replicated.
- The SnapMirror relationships are broken, making the destination volumes ready for read/write.
- The app's Kubernetes resources are restored from the pre-shutdown snapshot, using the volume data replicated after the original source app was shut down.
- Replication is re-established in the reverse direction.

Fail back applications to the original source cluster

Using Trident Protect, you can achieve "fail back" after a failover operation by using the following sequence of operations. In this workflow to restore the original replication direction, Trident Protect replicates (resyncs) any application changes back to the original source application before reversing the replication direction.

This process starts from a relationship that has completed a failover to a destination and involves the following steps:

- Start with a failed over state.
- Reverse resync the replication relationship.

Do not perform a normal resync operation, as this will discard data written to the destination cluster during the fail over procedure.

- Reverse the replication direction.

Steps

1. Perform the Reverse resync a failed over replication relationship steps.
2. Perform the Reverse application replication direction steps.

Delete a replication relationship

You can delete a replication relationship at any time. When you delete the application replication relationship, it results in two separate applications with no relationship between them.

Steps

1. On the current desination cluster, delete the AppMirrorRelationship CR:

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

Migrate applications using Trident Protect

You can migrate your applications between clusters or to different storage classes by restoring backup data.

When you migrate an application, all execution hooks configured for the application are migrated with the app. If a post-restore execution hook is present, it runs automatically as part of the restore operation.

Backup and restore operations

To perform backup and restore operations for the following scenarios, you can automate specific backup and restore tasks.

Clone to same cluster

To clone an application to the same cluster, create a snapshot or backup and restore the data to the same cluster.

Steps

1. Do one of the following:
   a. Create a snapshot.
   b. Create a backup.
2. On the same cluster, do one of the following, depending on if you created a snapshot or a backup:
   a. Restore your data from the snapshot.
   b. Restore your data from the backup.

Clone to different cluster

To clone an application to a different cluster (perform a cross-cluster clone), create a backup on the source cluster, and then restore the backup to a different cluster. Make sure that Trident Protect is installed on the destination cluster.

> You can replicate an application between different clusters using SnapMirror replication.

Steps

1. Create a backup.
2. Ensure that the AppVault CR for the object storage bucket that contains the backup has been configured on the destination cluster.
3. On the destination cluster, restore your data from the backup.

Migrate applications from one storage class to another storage class

You can migrate applications from one storage class to a different storage class by restoring a backup to the destination storage class.

For example (excluding the secrets from the restore CR):

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    - destination: "${destinationNamespace}"
      source: "${sourceNamespace}"
  storageClassMapping:
    - destination: "${destinationStorageClass}"
      source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

Restore the snapshot using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

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

3. Optionally, if you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `include or exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.

- **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-restore-cr.yaml` file with the correct values, apply the CR:

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

Restore the snapshot using the CLI

Steps

1. Restore the snapshot to a different namespace, replacing values in brackets with information from your environment.
   - The `snapshot` argument uses a namespace and snapshot name in the format `<namespace>/<name>`.
   - The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format `source1:dest1,source2:dest2`.

     For example:

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

Manage Trident Protect execution hooks

An execution hook is a custom action that you can configure to run in conjunction with a data protection operation of a managed app. For example, if you have a database app, you can use an execution hook to pause all database transactions before a snapshot, and resume transactions after the snapshot is complete. This ensures application-consistent snapshots.

Types of execution hooks

Trident Protect supports the following types of execution hooks, based on when they can be run:

- Pre-snapshot
- Post-snapshot
- Pre-backup
- Post-backup
- Post-restore
- Post-failover

Order of execution

When a data protection operation is run, execution hook events take place in the following order:

1. Any applicable custom pre-operation execution hooks are run on the appropriate containers. You can create and run as many custom pre-operation hooks as you need, but the order of execution of these hooks before the operation is neither guaranteed nor configurable.
2. Filesystem freezes occur, if applicable. Learn more about configuring filesystem freezing with Trident Protect.
3. The data protection operation is performed.
4. Frozen filesystems are unfrozen, if applicable.
5. Any applicable custom post-operation execution hooks are run on the appropriate containers. You can create and run as many custom post-operation hooks as you need, but the order of execution of these hooks after the operation is neither guaranteed nor configurable.

If you create multiple execution hooks of the same type (for example, pre-snapshot), the order of execution of those hooks is not guaranteed. However, the order of execution of hooks of different types is guaranteed. For example, the following is the order of execution of a configuration that has all of the different types of hooks:

1. Pre-snapshot hooks executed
2. Post-snapshot hooks executed

3. Pre-backup hooks executed
4. Post-backup hooks executed

> The preceding order example only applies when you run a backup that does not use an existing snapshot.

> You should always test your execution hook scripts before enabling them in a production environment. You can use the 'kubectl exec' command to conveniently test the scripts. After you enable the execution hooks in a production environment, test the resulting snapshots and backups to ensure they are consistent. You can do this by cloning the app to a temporary namespace, restoring the snapshot or backup, and then testing the app.

> If a pre-snapshot execution hook adds, changes, or removes Kubernetes resources, those changes are included in the snapshot or backup and in any subsequent restore operation.

Important notes about custom execution hooks

Consider the following when planning execution hooks for your apps.

- An execution hook must use a script to perform actions. Many execution hooks can reference the same script.
- Trident Protect requires the scripts that execution hooks use to be written in the format of executable shell scripts.
- Script size is limited to 96KB.
- Trident Protect uses execution hook settings and any matching criteria to determine which hooks are applicable to a snapshot, backup, or restore operation.

> Because execution hooks often reduce or completely disable the functionality of the application they are running against, you should always try to minimize the time your custom execution hooks take to run. If you start a backup or snapshot operation with associated execution hooks but then cancel it, the hooks are still allowed to run if the backup or snapshot operation has already begun. This means that the logic used in a post-backup execution hook cannot assume that the backup was completed.

Execution hook filters

When you add or edit an execution hook for an application, you can add filters to the execution hook to manage which containers the hook will match. Filters are useful for applications that use the same container image on all containers, but might use each image for a different purpose (such as Elasticsearch). Filters allow you to create scenarios where execution hooks run on some but not necessarily all identical containers. If you create multiple filters for a single execution hook, they are combined with a logical AND operator. You can have up to 10 active filters per execution hook.

Each filter you add to an execution hook uses a regular expression to match containers in your cluster. When a hook matches a container, the hook will run its associated script on that container. Regular expressions for filters use the Regular Expression 2 (RE2) syntax, which does not support creating a filter that excludes containers from the list of matches. For information on the syntax that Trident Protect supports for regular expressions in execution hook filters, see Regular Expression 2 (RE2) syntax support.

If you add a namespace filter to an execution hook that runs after a restore or clone operation and the restore or clone source and destination are in different namespaces, the namespace filter is only applied to the destination namespace.

Execution hook examples

Visit the NetApp Verda GitHub project to download real execution hooks for popular apps such as Apache Cassandra and Elasticsearch. You can also see examples and get ideas for structuring your own custom execution hooks.

Create an execution hook

You can create a custom execution hook for an app using Trident Protect. You need to have Owner, Admin, or Member permissions to create execution hooks.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-hook.yaml`.
2. Configure the following attributes to match your Trident Protect environment and cluster configuration:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) The Kubernetes name of the application for which to run the execution hook.
   - **spec.stage**: (*Required*) A string indicating which stage during the action that the execution hook should run. Possible values:
     - Pre
     - Post
   - **spec.action**: (*Required*) A string indicating which action the execution hook will take, assuming any execution hook filters specified are matched. Possible values:
     - Snapshot
     - Backup
     - Restore
     - Failover
   - **spec.enabled**: (*Optional*) Indicates whether this execution hook is enabled or disabled. If not specified, the default value is true.
   - **spec.hookSource**: (*Required*) A string containing the base64-encoded hook script.
   - **spec.timeout**: (*Optional*) A number defining how long in minutes that the execution hook is allowed to run. The minimum value is 1 minute, and the default value is 25 minutes if not specified.
   - **spec.arguments**: (*Optional*) A YAML list of arguments that you can specify for the execution hook.
   - **spec.matchingCriteria**: (*Optional*) An optional list of criteria key value pairs, each pair making up an execution hook filter. You can add up to 10 filters per execution hook.
   - **spec.matchingCriteria.type**: (*Optional*) A string identifying the execution hook filter type. Possible values:
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value**: (*Optional*) A string or regular expression identifying the execution hook filter value.

     Example YAML:

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. After you populate the CR file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-hook.yaml
```

Use the CLI

Steps

1. Create the execution hook, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

Manually run an execution hook

You can manually run an execution hook for testing purposes or if you need to re-run the hook manually after a failure. You need to have Owner, Admin, or Member permissions to manually run execution hooks.

Manually running an execution hook consists of two basic steps:

1. Create a resource backup, which collects resources and creates a backup of them, determining where the hook will run
2. Run the execution hook against the backup

Step 1: Create a resource backup

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-resource-backup.yaml`.
2. Configure the following attributes to match your Trident Protect environment and cluster configuration:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) The Kubernetes name of the application for which to create the resource backup.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

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

     Example YAML:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: ResourceBackup
     metadata:
       name: example-resource-backup
     spec:
       applicationRef: my-app-name
       appVaultRef: my-appvault-name
       appArchivePath: example-resource-backup
     ```
3. After you populate the CR file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-resource-backup.yaml
   ```

Use the CLI

Steps

1. Create the backup, replacing values in brackets with information from your environment. For example:

```
tridentctl protect create resourcebackup <my_backup_name> --app
<my_app_name> --appvault <my_appvault_name> -n
<my_app_namespace> --app-archive-path <app_archive_path>
```

2. View the status of the backup. You can use this example command repeatedly until the operation is complete:

```
tridentctl protect get resourcebackup -n <my_app_namespace>
<my_backup_name>
```

3. Verify that the backup was successful:

```
kubectl describe resourcebackup <my_backup_name>
```

Step 2: Run the execution hook

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-hook-run.yaml`.
2. Configure the following attributes to match your Trident Protect environment and cluster configuration:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) Ensure this value matches the application name from the ResourceBackup CR you created in step 1.
   - **spec.appVaultRef**: (*Required*) Ensure this value matches the appVaultRef from the ResourceBackup CR you created in step 1.
   - **spec.appArchivePath**: Ensure this value matches the appArchivePath from the ResourceBackup CR you created in step 1.

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

   - **spec.action**: (*Required*) A string indicating which action the execution hook will take, assuming any execution hook filters specified are matched. Possible values:
     - Snapshot
     - Backup
     - Restore
     - Failover
   - **spec.stage**: (*Required*) A string indicating which stage during the action that the execution hook should run. This hook run will not run hooks in any other stage. Possible values:
     - Pre
     - Post

     Example YAML:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: ExecHooksRun
     metadata:
       name: example-hook-run
     spec:
       applicationRef: my-app-name
       appVaultRef: my-appvault-name
       appArchivePath: example-resource-backup
       stage: Post
       action: Failover
     ```

3. After you populate the CR file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-hook-run.yaml
```

Use the CLI

Steps

1. Create the manual execution hook run request:

```
tridentctl protect create exechooksrun <my_exec_hook_run_name>
-n <my_app_namespace> --action snapshot --stage <pre_or_post>
--app <my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. Check the status of the execution hook run. You can run this command repeatedly until the operation is complete:

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. Describe the exechooksrun object to see the final details and status:

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

Uninstall Trident Protect

You might need to remove Trident Protect components if you are upgrading from a trial to a full version of the product.

To remove Trident Protect, perform the following steps.

Steps

1. Remove the Trident Protect CR files:

   This step is not required for version 25.06 and later.

```
helm uninstall -n trident-protect trident-protect-crds
```

2. Remove Trident Protect:

```
helm uninstall -n trident-protect trident-protect
```

3. Remove the Trident Protect namespace:

```
kubectl delete ns trident-protect
```

Trident and Trident Protect blogs

You can find some great NetApp Trident and Trident Protect blogs here:

Trident blogs

- October 16, 2025: Advanced storage solutions for Kubernetes
- August 19, 2025: Enhancing Data Consistency: Volume Group Snapshots in OpenShift virtualization with Trident
- May 09, 2025: Automatic Trident backend configuration for FSx for ONTAP with the Amazon EKS add-on
- April 15, 2025: NetApp Trident with Google Cloud NetApp Volumes for SMB Protocol
- April 14, 2025: Leveraging Fiber Channel Protocol with Trident 25.02 for Persistent Storage on Kubernetes
- April 14, 2025: Unlocking the power of NetApp ASA r2 systems for Kubernetes block storage
- March 31, 2025: Simplifying Trident Installation on Red Hat OpenShift with the New Certified Operator
- March 27, 2025: Provisioning Trident for SMB with Google Cloud NetApp Volumes
- March 05, 2025: Unlock Seamless iSCSI Storage Integration: A Guide to FSxN on ROSA Clusters for AWS
- February 27, 2025: Deploying cloud identity with Trident, GKE, and Google Cloud NetApp Volumes
- December 12, 2024: Introducing Fibre Channel support in Trident
- November 11, 2024: NetApp Trident with Google Cloud NetApp Volumes
- October 29, 2024: Amazon FSx for NetApp ONTAP with Red Hat OpenShift Service on AWS (ROSA) using Trident
- October 29, 2024: Live Migration of VMs with OpenShift Virtualization on ROSA and Amazon FSx for NetApp ONTAP
- July 08, 2024: Using NVMe/TCP to consume ONTAP storage for your modern containerized apps on Amazon EKS
- July 01, 2024: Seamless Kubernetes storage with Google Cloud NetApp Volumes Flex and Astra Trident
- June 11, 2024: ONTAP as backend storage for the integrated image registry in OpenShift

Trident Protect blogs

- May 16, 2025: Automating registry failover for disaster recovery with Trident Protect post-restore hooks
- May 16, 2025: OpenShift Virtualization Disaster Recovery with NetApp Trident Protect
- May 13, 2025: Storage class migration with Trident Protect backup & restore
- May 09, 2025: Rescale Kubernetes applications with Trident Protect post-restore hooks
- April 03, 2025: Trident Protect Power Up: Kubernetes Replication for Protection & Disaster Recovery
- Mar 13, 2025: Crash-Consistent Backup and Restore Operations for OpenShift Virtualization VMs
- Mar 11, 2025: Extending GitOps patterns to application data protection with NetApp Trident
- Mar 03, 2025: Trident 25.02: Elevating the Red Hat OpenShift Experience with Exciting New Features
- Jan 15, 2025: Introducing Trident Protect role-based access control
- Nov 11, 2024: Kubernetes-driven data management: The new era with Trident Protect

Knowledge and support

Frequently asked questions

Find answers to the frequently asked questions about installing, configuring, upgrading, and troubleshooting Trident.

General questions

How frequently is Trident released?

Beginning with the 24.02 release, Trident is released every four months: February, June, and October.

Does Trident support all the features that are released in a particular version of Kubernetes?

Trident usually does not support alpha features in Kubernetes. Trident might support beta features within the two Trident releases that follow the Kubernetes beta release.

Does Trident have any dependencies on other NetApp products for its functioning?

Trident does not have any dependencies on other NetApp software products and it works as a standalone application. However, you should have a NetApp backend storage device.

How can I obtain complete Trident configuration details?

Use the `tridentctl get` command to obtain more information about your Trident configuration.

Can I obtain metrics on how storage is provisioned by Trident?

Yes. Prometheus endpoints that can be used to gather information about Trident operation, such as the number of backends managed, the number of volumes provisioned, bytes consumed, and so on. You can also use Cloud Insights for monitoring and analysis.

Does the user experience change when using Trident as a CSI Provisioner?

No. There are no changes as far as the user experience and functionalities are concerned. The provisioner name used is `csi.trident.netapp.io`. This method of installing Trident is recommended if you want to use all the new features provided by current and future releases.

Install and use Trident on a Kubernetes cluster

Does Trident support an offline install from a private registry?

Yes, Trident can be installed offline. Refer to Learn about Trident installation.

Can I install Trident be remotely?

Yes. Trident 18.10 and later support remote installation capability from any machine that has `kubectl` access to the cluster. After `kubectl` access is verified (for example, initiate a `kubectl get nodes` command from the remote machine to verify), follow the installation instructions.

Can I configure High Availability with Trident?

Trident is installed as a Kubernetes Deployment (ReplicaSet) with one instance, and so it has HA built in. You should not increase the number of replicas in the deployment. If the node where Trident is installed is lost or the pod is otherwise inaccessible, Kubernetes automatically re-deploys the pod to a healthy node in your cluster. Trident is control-plane only, so currently mounted pods are not affected if Trident is re-deployed.

Does Trident need access to the kube-system namespace?

Trident reads from the Kubernetes API Server to determine when applications request new PVCs, so it needs access to kube-system.

What are the roles and privileges used by Trident?

The Trident installer creates a Kubernetes ClusterRole, which has specific access to the cluster's PersistentVolume, PersistentVolumeClaim, StorageClass, and Secret resources of the Kubernetes cluster. Refer to Customize tridentctl installation.

Can I locally generate the exact manifest files Trident uses for installation?

You can locally generate and modify the exact manifest files Trident uses for installation, if needed. Refer to Customize tridentctl installation.

Can I share the same ONTAP backend SVM for two separate Trident instances for two separate Kubernetes clusters?

Although it is not advised, you can use the same backend SVM for two Trident instances. Specify a unique volume name for each instance during installation and/or specify a unique `StoragePrefix` parameter in the `setup/backend.json` file. This is to ensure the same FlexVol volume is not used for both instances.

Is it possible to install Trident under ContainerLinux (formerly CoreOS)?

Trident is simply a Kubernetes pod and can be installed wherever Kubernetes is running.

Can I use Trident with NetApp Cloud Volumes ONTAP?

Yes, Trident is supported on AWS, Google Cloud, and Azure.

Troubleshooting and support

Does NetApp support Trident?

Although Trident is open source and provided for free, NetApp fully supports it provided your NetApp backend is supported.

How do I raise a support case?

To raise a support case, do one of the following:

1. Contact your Support Account Manager and get help to raise a ticket.
2. Raise a support case by contacting NetApp Support.

How do I generate a support log bundle?

You can create a support bundle by running `tridentctl logs -a`. In addition to the logs captured in the bundle, capture the kubelet log to diagnose the mount problems on the Kubernetes side. The instructions to get the kubelet log varies based on how Kubernetes is installed.

What do I do if I need to raise a request for a new feature?

Create an issue on Trident Github and mention **RFE** in the subject and description of the issue.

Where do I raise a defect?

Create an issue on Trident Github. Make sure to include all the necessary information and logs pertaining to the issue.

What happens if I have quick question on Trident that I need clarification on? Is there a community or a forum?

If you have any questions, issues, or requests, reach out to us through our Trident Discord channel or GitHub.

My storage system's password has changed and Trident no longer works, how do I recover?

Update the backend's password with `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`. Replace `myBackend` in the example with your backend name, and `/path/to_new_backend.json` with the path to the correct `backend.json` file.

Trident cannot find my Kubernetes node. How do I fix this?

There are two likely scenarios why Trident cannot find a Kubernetes node. It can be because of a networking issue within Kubernetes or a DNS issue. The Trident node daemonset that runs on each Kubernetes node must be able to communicate with the Trident controller to register the node with Trident. If networking changes occurred after Trident was installed, you encounter this problem only with new Kubernetes nodes that are added to the cluster.

If the Trident pod is destroyed, will I lose the data?

Data will not be lost if the Trident pod is destroyed. Trident metadata is stored in CRD objects. All PVs that have been provisioned by Trident will function normally.

Upgrade Trident

Can I upgrade from a older version directly to a newer version (skipping a few versions)?

NetApp supports upgrading Trident from one major release to the next immediate major release. You can upgrade from version 18.xx to 19.xx, 19.xx to 20.xx, and so on. You should test upgrading in a lab before production deployment.

Is it possible to downgrade Trident to a previous release?

If you need a fix for bugs observed after an upgrade, dependency issues, or an unsuccessful or incomplete upgrade, you should uninstall Trident and reinstall the earlier version using the specific instructions for that version. This is the only recommended way to downgrade to an earlier version.

Manage backends and volumes

Do I need to define both Management and DataLIFs in an ONTAP backend definition file?

The management LIF is mandatory. DataLIF varies:

- ONTAP SAN: Do not specify for iSCSI. Trident uses ONTAP Selective LUN Map to discover the iSCI LIFs needed to establish a multi path session. A warning is generated if `dataLIF` is explicitly defined. Refer to ONTAP SAN configuration options and examples for details.
- ONTAP NAS: NetApp recommends specifying `dataLIF`. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. Refer to ONTAP NAS configuration options and examples for details

Can Trident configure CHAP for ONTAP backends?

Yes. Trident supports bidirectional CHAP for ONTAP backends. This requires setting `useCHAP=true` in your backend configuration.

How do I manage export policies with Trident?

Trident can dynamically create and manage export policies from version 20.04 onwards. This enables the storage administrator to provide one or more CIDR blocks in their backend configuration and have Trident add node IPs that fall within these ranges to an export policy it creates. In this manner, Trident automatically manages the addition and deletion of rules for nodes with IPs within the given CIDRs.

Can IPv6 addresses be used for the Management and DataLIFs?

Trident supports defining IPv6 addresses for:

- `managementLIF` and `dataLIF` for ONTAP NAS backends.
- `managementLIF` for ONTAP SAN backends. You cannot specify `dataLIF` on an ONTAP SAN backend.

Trident must be installed using the flag `--use-ipv6` (for `tridentctl` installation), `IPv6` (for Trident operator), or `tridentTPv6` (for Helm installation) for it to function over IPv6.

Is it possible to update the Management LIF on the backend?

Yes, it is possible to update the backend Management LIF using the `tridentctl update backend` command.

Is it possible to update the DataLIF on the backend?

You can update the DataLIF on `ontap-nas` and `ontap-nas-economy` only.

Can I create multiple backends in Trident for Kubernetes?

Trident can support many backends simultaneously, either with the same driver or different drivers.

How does Trident store backend credentials?

Trident stores the backend credentials as Kubernetes Secrets.

How does Trident select a specific backend?

If the backend attributes cannot be used to automatically select the right pools for a class, the `storagePools` and `additionalStoragePools` parameters are used to select a specific set of pools.

How do I ensure that Trident will not provision from a specific backend?

The `excludeStoragePools` parameter is used to filter the set of pools that Trident uses for provisioning and will remove any pools that match.

If there are multiple backends of the same kind, how does Trident select which backend to use?

If there are multiple configured backends of the same type, Trident selects the appropriate backend based on the parameters present in `StorageClass` and `PersistentVolumeClaim`. For example, if there are multiple ontap-nas driver backends, Trident tries to match parameters in the `StorageClass` and `PersistentVolumeClaim` combined and match a backend which can deliver the requirements listed in `StorageClass` and `PersistentVolumeClaim`. If there are multiple backends that match the request, Trident selects from one of them at random.

Does Trident support bi-directional CHAP with Element/SolidFire?

Yes.

How does Trident deploy Qtrees on an ONTAP volume? How many Qtrees can be deployed on a single volume?

The `ontap-nas-economy` driver creates up to 200 Qtrees in the same FlexVol volume (configurable between 50 and 300), 100,000 Qtrees per cluster node, and 2.4M per cluster. When you enter a new `PersistentVolumeClaim` that is serviced by the economy driver, the driver looks to see if a FlexVol volume already exists that can service the new Qtree. If the FlexVol volume does not exist that can service the Qtree, a new FlexVol volume is created.

How can I set Unix permissions for volumes provisioned on ONTAP NAS?

You can set Unix permissions on the volume provisioned by Trident by setting a parameter in the backend definition file.

How can I configure an explicit set of ONTAP NFS mount options while provisioning a volume?

By default, Trident does not set mount options to any value with Kubernetes. To specify the mount options in the Kubernetes Storage Class, follow the example given here.

How do I set the provisioned volumes to a specific export policy?

To allow the appropriate hosts access to a volume, use the `exportPolicy` parameter configured in the backend definition file.

How do I set volume encryption through Trident with ONTAP?

You can set encryption on the volume provisioned by Trident by using the encryption parameter in the backend definition file. For more information, refer to: How Trident works with NVE and NAE

What is the best way to implement QoS for ONTAP through Trident?

Use `StorageClasses` to implement QoS for ONTAP.

How do I specify thin or thick provisioning through Trident?

The ONTAP drivers support either thin or thick provisioning. The ONTAP drivers default to thin provisioning. If thick provisioning is desired, you should configure either the backend definition file or the `StorageClass`. If both are configured, `StorageClass` takes precedence. Configure the following for ONTAP:

1. On `StorageClass`, set the `provisioningType` attribute as thick.
2. In the backend definition file, enable thick volumes by setting `backend spaceReserve parameter` as volume.

How do I make sure that the volumes being used are not deleted even if I accidentally delete the PVC?

PVC protection is automatically enabled on Kubernetes starting from version 1.10.

Can I grow NFS PVCs that were created by Trident?

Yes. You can expand a PVC that has been created by Trident. Note that volume autogrow is an ONTAP feature that is not applicable to Trident.

Can I import a volume while it is in SnapMirror Data Protection (DP) or offline mode?

The volume import fails if the external volume is in DP mode or is offline. You receive the following error message:

```
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.
```

How is resource quota translated to a NetApp cluster?

Kubernetes Storage Resource Quota should work as long as NetApp storage has capacity. When the NetApp storage cannot honor the Kubernetes quota settings due to lack of capacity, Trident tries to provision but errors out.

Can I create Volume Snapshots using Trident?

Yes. Creating on-demand volume snapshots and Persistent Volumes from Snapshots are supported by Trident. To create PVs from snapshots, ensure that the `VolumeSnapshotDataSource` feature gate has been enabled.

What are the drivers that support Trident volume snapshots?

As of today, on-demand snapshot support is available for our `ontap-nas`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, and `azure-netapp-files` backend drivers.

How do I take a snapshot backup of a volume provisioned by Trident with ONTAP?

This is available on `ontap-nas`, `ontap-san`, and `ontap-nas-flexgroup` drivers. You can also specify a `snapshotPolicy` for the `ontap-san-economy` driver at the FlexVol level.

This is also available on the `ontap-nas-economy` drivers but on the FlexVol volume level granularity and not on the qtree level granularity. To enable the ability to snapshot volumes provisioned by Trident, set the backend parameter option `snapshotPolicy` to the desired snapshot policy as defined on the ONTAP backend. Any snapshots taken by the storage controller are not known by Trident.

Can I set a snapshot reserve percentage for a volume provisioned through Trident?

Yes, you can reserve a specific percentage of disk space for storing the snapshot copies through Trident by setting the `snapshotReserve` attribute in the backend definition file. If you have configured `snapshotPolicy` and `snapshotReserve` in the backend definition file, snapshot reserve percentage is set according to the `snapshotReserve` percentage mentioned in the backend file. If the `snapshotReserve` percentage number is not mentioned, ONTAP by default takes the snapshot reserve percentage as 5. If the `snapshotPolicy` option is set to none, the snapshot reserve percentage is set to 0.

Can I directly access the volume snapshot directory and copy files?

Yes, you can access the snapshot directory on the volume provisioned by Trident by setting the `snapshotDir` parameter in the backend definition file.

Can I set up SnapMirror for volumes through Trident?

Currently, SnapMirror has to be set externally by using ONTAP CLI or OnCommand System Manager.

How do I restore Persistent Volumes to a specific ONTAP snapshot?

To restore a volume to an ONTAP snapshot, perform the following steps:

1. Quiesce the application pod which is using the Persistent volume.
2. Revert to the required snapshot through ONTAP CLI or OnCommand System Manager.
3. Restart the application pod.

Can Trident provision volumes on SVMs that have a Load-Sharing Mirror configured?

Load-sharing mirrors can be created for root volumes of SVMs that serve data over NFS. ONTAP automatically updates load-sharing mirrors for volumes that have been created by Trident. This may result in delays in mounting volumes. When multiple volumes are created using Trident, provisioning a volume is dependent on ONTAP updating the load-sharing mirror.

How can I separate out storage class usage for each customer/tenant?

Kubernetes does not allow storage classes in namespaces. However, you can use Kubernetes to limit usage of a specific storage class per namespace by using Storage Resource Quotas, which are per namespace. To deny a specific namespace access to specific storage, set the resource quota to 0 for that storage class.

Troubleshooting

Use the pointers provided here for troubleshooting issues you might encounter while

installing and using Trident.

> For help with Trident, create a support bundle using `tridentctl logs -a -n trident` and send it to NetApp Support.

General troubleshooting

- If the Trident pod fails to come up properly (for example, when the Trident pod is stuck in the `ContainerCreating` phase with fewer than two ready containers), running `kubectl -n trident describe deployment trident` and `kubectl -n trident describe pod trident-**` can provide additional insights. Obtaining kubelet logs (for example, via `journalctl -xeu kubelet`) can also be helpful.
- If there is not enough information in the Trident logs, you can try enabling the debug mode for Trident by passing the `-d` flag to the install parameter based on your installation option.

  Then confirm debug is set using `./tridentctl logs -n trident` and searching for `level=debug msg` in the log.

  Installed with Operator

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

  This will restart all Trident pods, which can take several seconds. You can check this by observing the 'AGE' column in the output of `kubectl get pod -n trident`.

  For Trident 20.07 and 20.10 use `tprov` in place of `torc`.

  Installed with Helm

  ```
  helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
  tridentDebug=true`
  ```

  Installed with tridentctl

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

- You can also obtain debug logs for each backend by including `debugTraceFlags` in your backend definition. For example, include `debugTraceFlags: {"api":true, "method":true,}` to obtain API calls and method traversals in the Trident logs. Existing backends can have `debugTraceFlags` configured with a `tridentctl backend update`.
- When using Red Hat Enterprise Linux CoreOS (RHCOS), ensure that `iscsid` is enabled on the worker nodes and started by default. This can be done using OpenShift MachineConfigs or by modifying the ignition templates.
- A common problem you could encounter when using Trident with Azure NetApp Files is when the tenant and client secrets come from an app registration with insufficient permissions. For a complete list of Trident requirements, Refer to Azure NetApp Files configuration.

- If there are problems with mounting a PV to a container, ensure that `rpcbind` is installed and running. Use the required package manager for the host OS and check if `rpcbind` is running. You can check the status of the `rpcbind` service by running a `systemctl status rpcbind` or its equivalent.
- If a Trident backend reports that it is in the `failed` state despite having worked before, it is likely caused by changing the SVM/admin credentials associated with the backend. Updating the backend information using `tridentctl update backend` or bouncing the Trident pod will fix this issue.
- If you encounter permission issues when installing Trident with Docker as the container runtime, attempt the installation of Trident with the `--in cluster=false` flag. This will not use an installer pod and avoid permission troubles seen due to the `trident-installer` user.
- Use the `uninstall parameter <Uninstalling Trident>` for cleaning up after a failed run. By default, the script does not remove the CRDs that have been created by Trident, making it safe to uninstall and install again even in a running deployment.
- If you want to downgrade to an earlier version of Trident, first run the `tridentctl uninstall` command to remove Trident. Download the desired Trident version and install using the `tridentctl install` command.
- After a successful install, if a PVC is stuck in the `Pending` phase, running `kubectl describe pvc` can provide additional information about why Trident failed to provision a PV for this PVC.

Unsuccessful Trident deployment using the operator

If you are deploying Trident using the operator, the status of `TridentOrchestrator` changes from `Installing` to `Installed`. If you observe the `Failed` status, and the operator is unable to recover by itself, you should check the logs of the operator by running following command:

```
tridentctl logs -l trident-operator
```

Trailing the logs of the trident-operator container can point to where the problem lies. For example, one such issue could be the inability to pull the required container images from upstream registries in an airgapped environment.

To understand why the installation of Trident was unsuccessful, you should take a look at the `TridentOrchestrator` status.

```
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'
```

This error indicates that there already exists a `TridentOrchestrator` that was used to install Trident. Since each Kubernetes cluster can only have one instance of Trident, the operator ensures that at any given time there only exists one active `TridentOrchestrator` that it can create.

In addition, observing the status of the Trident pods can often indicate if something is not right.

```
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
```

You can clearly see that the pods are not able to initialize completely because one or more container images were not fetched.

To address the problem, you should edit the `TridentOrchestrator` CR. Alternatively, you can delete `TridentOrchestrator`, and create a new one with the modified and accurate definition.

Unsuccessful Trident deployment using `tridentctl`

To help figure out what went wrong, you could run the installer again using the `-d` argument, which will turn on debug mode and help you understand what the problem is:

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

After addressing the problem, you can clean up the installation as follows, and then run the `tridentctl install` command again:

```
./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.
```

Completely remove Trident and CRDs

You can completely remove Trident and all created CRDs and associated custom resources.

This cannot be undone. Do not do this unless you want a completely fresh installation of Trident. To uninstall Trident without removing CRDs, refer to Uninstall Trident.

Trident operator

To uninstall Trident and completely remove CRDs using the Trident operator:

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

Helm

To uninstall Trident and completely remove CRDs using Helm:

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

`tridentctl`

To completely remove CRDs after uninstalling Trident using `tridentctl`

```
tridentctl obliviate crd
```

NVMe node unstaging failure with RWX raw block namespaces o Kubernetes 1.26

If you are running Kubernetes 1.26, node unstaging might fail when using NVMe/TCP with RWX raw block namespaces. The following scenarios provide workaround to the failure. Alternatively, you can upgrade Kubernetes to 1.27.

Deleted the namespace and pod

Consider a scenario where you have a Trident managed namespace (NVMe persistent volume) attached to a pod. If you delete the namespace directly from the ONTAP backend, the unstaging process gets stuck after you attempt to delete the pod. This scenario does not impact the Kubernetes cluster or other functioning.

Workaround

Unmount the persistent volume (corresponding to that namespace) from the respective node and delete it.

Blocked dataLIFs

```
If you block (or bring down) all the dataLIFs of the NVMe Trident backend,
the unstaging process gets stuck when you attempt to delete the pod. In
this scenario, you cannot run any NVMe CLI commands on the Kubernetes
node.
```

Workaround

Bring up the dataLIFS to restore full functionality.

Deleted namespace mapping

```
If you remove the `hostNQN` of the worker node from the corresponding
subsystem, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```

Workaround

Add the `hostNQN` back to the subsystem.

NFSv4.2 clients report "invalid argument" after upgrading ONTAP when when expecting "v4.2-xattrs" being enabled

After upgrading ONTAP, NFSv4.2 clients might report "invalid argument" errors when attempting to mount NFSv4.2 exports. This issue occurs when the `v4.2-xattrs` option is not enabled on the SVM.
.Workaround
Enable the `v4.2-xattrs` option on the SVM or upgrade to ONTAP 9.12.1 or later, where this option is enabled by default.

Support

NetApp provides support for Trident in a variety of ways. Extensive free self-support options are available 24x7, such as knowledgebase (KB) articles and a Discord channel.

Trident support lifecycle

Trident provides three levels of support based on your version. Refer to NetApp software version support for definitions.

Full support

Trident provides full support for twelve months from the release date.

Limited support

Trident provides limited support for months 13 - 24 from the release date.

Self-support

Trident documentation is available for months 25 - 36 from the release date.

Table 2. Trident version support schedule

Version	Full support	Limited support	Self-support
26.02	February 2026	February 2027	February 2028
25.10	October 2026	October 2027	October 2028

25.06	June 2026	June 2027	June 2028
25.02	February 2026	February 2027	February 2028
24.10	—	October 2026	October 2027
24.06	—	June 2026	June 2027
24.02	—	February 2026	February 2027
23.10	—	—	October 2026
23.07	—	—	July 2026
23.04	—	—	April 2026

Self-support

For a comprehensive list of troubleshooting articles, refer to NetApp Knowledgebase (login required).

Community support

There is a vibrant public community of container users (including Trident developers) on our Discord channel. This is a great place to ask general questions about the project and discuss related topics with like-minded peers.

NetApp technical support

For help with Trident, create a support bundle using `tridentctl logs -a -n trident` and send it to `NetApp Support <Getting Help>`.

For more information

- Trident resources
- Kubernetes Hub

Reference

Trident ports

Learn more about the ports that Trident uses for communication.

Overview

Trident uses various ports for communication inside Kubernetes clusters and with storage backends. The following is a summary of key ports, their purposes, and security considerations.

- **Outbound focus**: Kubernetes nodes (controller and worker) primarily initiate traffic to storage LIFs/IPs, so iptables rules should allow outbound from node IPs to specific storage IPs on these ports. Avoid broad "any-to-any" rules.
- **Inbound restrictions**: Limit internal Trident ports to cluster-internal traffic (for example, using CNI like Calico). No unnecessary inbound exposure on host firewalls.
- **Protocol security**:
  - Use TCP where possible (more reliable).
  - Enable CHAP/IPsec for iSCSI if sensitive; TLS/HTTPS for management (port 443/8443).
  - For NFSv4 (default in Trident), prune UDP/older NFSv3 ports (for example, 4045-4049) if not needed.
  - Restrict to trusted subnets; monitor with tools like Prometheus (optional port 8001).

Ports for controller nodes

These ports are primarily for Trident operator (backend management). All internal ports are pod-level; allow on nodes only if host firewall interferes with CNI.

Port/Protocol	Direction	Purpose	Driver/Protocol	Security Notes
TCP 8000	Inbound/Outbound (cluster-internal)	Trident REST server (operator-controller comms)	All	Restrict to pod CIDRs; no external exposure.
TCP 8443	Inbound/Outbound (cluster-internal)	Backchannel HTTPS (secure internal API)	All	TLS-encrypted; limit to Kubernetes service mesh if used.
TCP 8001	Inbound (cluster-internal, optional)	Prometheus metrics	All	Expose only to monitoring tools (for example, using RBAC); disable if unused.
TCP 443	Outbound	HTTPS to ONTAP SVM/cluster mgmt LIF	ONTAP (all), ANF	Require TLS cert validation; restrict to mgmt LIF IPs only.
TCP 8443	Outbound	HTTPS to E-Series Web Services Proxy	E-Series (iSCSI)	Default REST API; use certs; configurable in backend YAML.

Ports for worker nodes

These ports are for CSI node daemonsets and pod mounts. Data ports are outbound to storage data LIFs; include NFSv3 extras if using NFSv3 (optional for NFSv4).

Port/Protocol	Direction	Purpose	Driver/Protocol	Security Notes
TCP 17546	Inbound (local to pod)	CSI node liveness/readiness probes	All	Configurable (--probe-port); ensure no host conflicts; local-only.
TCP 8000	Inbound/Outbound (cluster-internal)	Trident REST server	All	As above; pod-internal.
TCP 8443	Inbound/Outbound (cluster-internal)	Backchannel HTTPS	All	As above.
TCP 8001	Inbound (cluster-internal, optional)	Prometheus metrics	All	As above.
TCP 443	Outbound	HTTPS to ONTAP SVM/cluster mgmt LIF	ONTAP (all), ANF	As above; used for discovery.
TCP 8443	Outbound	HTTPS to E-Series Web Services Proxy	E-Series (iSCSI)	As above.
TCP/UDP 111	Outbound	RPCBIND/portmapper	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Required for v3; optional for v4 (firewall offload); restrict if using NFSv4-only.
TCP/UDP 2049	Outbound	NFS daemon	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Core data; well-known; use TCP for reliability.
TCP/UDP 635	Outbound	Mount daemon	ONTAP-NAS (NFSv3/v4), ANF (NFS)	Mounting; bidirectional callbacks possible (allow inbound ephemeral if needed).
UDP 4045	Outbound	NFS lock manager (nlockmgr)	ONTAP-NAS (NFSv3)	File locking; skip for v4 (pNFS handles); UDP-only.
UDP 4046	Outbound	NFS status monitor (statd)	ONTAP-NAS (NFSv3)	Notifications; may need inbound ephemeral ports (1024-65535) for callbacks.
UDP 4049	Outbound	NFS quota daemon (rquotad)	ONTAP-NAS (NFSv3)	Quotas; skip for v4.
TCP 3260	Outbound	iSCSI target (discovery/data/CHAP)	ONTAP-SAN (iSCSI), E-Series (iSCSI)	Well-known; CHAP auth over this port; enable mutual CHAP for security.

Port/Protocol	Direction	Purpose	Driver/Protocol	Security Notes
TCP 445	Outbound	SMB/CIFS	ONTAP-NAS (SMB), ANF (SMB)	Well-known; use SMB3 with encryption (Trident annotation netapp.io/smb-encryption=true).
TCP/UDP 88 (optional)	Outbound	Kerberos auth	ONTAP (NFS/SMB/iSCSI with Kerb)	If using Kerberos (not default); to AD servers, not storage.
TCP/UDP 389 (optional)	Outbound	LDAP	ONTAP (NFS/SMB with LDAP)	Similar; for name resolution/auth; restrict to AD.

The liveness/readiness probe port can be changed during installation using the `--probe-port` flag. It is important to make sure this port isn't being used by another process on the worker nodes.

Trident REST API

While tridentctl commands and options are the easiest way to interact with the Trident REST API, you can use the REST endpoint directly if you prefer.

When to use the REST API

REST API is useful for advanced installations that use Trident as a standalone binary in non-Kubernetes deployments.

For better security, the Trident `REST API` is restricted to localhost by default when running inside a pod. To change this behavior, you need to set Trident's `-address` argument in its pod configuration.

Using REST API

For examples of how these APIs are called, pass the debug (`-d`) flag. For more information, refer to Manage Trident using tridentctl.

The API works as follows:

GET

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

Lists all objects of that type.

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

Gets the details of the named object.

POST

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

Creates an object of the specified type.

- Requires a JSON configuration for the object to be created. For the specification of each object type, refer to Manage Trident using tridentctl.
- If the object already exists, behavior varies: backends update the existing object, while all other object types will fail the operation.

DELETE

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

Deletes the named resource.

Volumes associated with backends or storage classes will continue to exist; these must be deleted separately. For more information, refer to Manage Trident using tridentctl.

Command-line options

Trident exposes several command-line options for the Trident orchestrator. You can use these options to modify your deployment.

Logging

-debug

Enables debugging output.

-loglevel <level>

Sets the logging level (debug, info, warn, error, fatal). Defaults to info.

Kubernetes

-k8s_pod

Use this option or `-k8s_api_server` to enable Kubernetes support. Setting this causes Trident to use its containing pod's Kubernetes service account credentials to contact the API server. This only works when Trident runs as a pod in a Kubernetes cluster with service accounts enabled.

-k8s_api_server <insecure-address:insecure-port>

Use this option or `-k8s_pod` to enable Kubernetes support. When specified, Trident connects to the Kubernetes API server using the provided insecure address and port. This Enables Trident to be deployed outside of a pod; however, it only supports insecure connections to the API server. To connect securely, deploy Trident in a pod with the `-k8s_pod` option.

Docker

-volume_driver <name>

Driver name used when registering the Docker plugin. Defaults to `netapp`.

-driver_port <port-number>

Listen on this port rather than a UNIX domain socket.

-config <file>

Required; you must specify this path to a backend configuration file.

REST

-address <ip-or-host>

Specifies the address on which Trident's REST server should listen. Defaults to localhost. When listening on localhost and running inside a Kubernetes pod, the REST interface isn't directly accessible from outside the pod. Use `-address ""` to make the REST interface accessible from the pod IP address.

Trident REST interface can be configured to listen and serve at 127.0.0.1 (for IPv4) or [::1] (for IPv6) only.

-port <port-number>

Specifies the port on which Trident's REST server should listen. Defaults to 8000.

-rest

Enables the REST interface. Defaults to true.

Kubernetes and Trident objects

You can interact with Kubernetes and Trident using REST APIs by reading and writing resource objects. There are several resource objects that dictate the relationship between Kubernetes and Trident, Trident and storage, and Kubernetes and storage. Some of these objects are managed through Kubernetes and the others are managed through Trident.

How do the objects interact with one another?

Perhaps the easiest way to understand the objects, what they are for, and how they interact, is to follow a single request for storage from a Kubernetes user:

1. A user creates a `PersistentVolumeClaim` requesting a new `PersistentVolume` of a particular size from a Kubernetes `StorageClass` that was previously configured by the administrator.
2. The Kubernetes `StorageClass` identifies Trident as its provisioner and includes parameters that tell Trident how to provision a volume for the requested class.
3. Trident looks at its own `StorageClass` with the same name that identifies the matching `Backends` and `StoragePools` that it can use to provision volumes for the class.
4. Trident provisions storage on a matching backend and creates two objects: a `PersistentVolume` in Kubernetes that tells Kubernetes how to find, mount, and treat the volume, and a volume in Trident that retains the relationship between the `PersistentVolume` and the actual storage.
5. Kubernetes binds the `PersistentVolumeClaim` to the new `PersistentVolume`. Pods that include the `PersistentVolumeClaim` mount that PersistentVolume on any host that it runs on.

6. A user creates a `VolumeSnapshot` of an existing PVC, using a `VolumeSnapshotClass` that points to Trident.
7. Trident identifies the volume that is associated with the PVC and creates a snapshot of the volume on its backend. It also creates a `VolumeSnapshotContent` that instructs Kubernetes on how to identify the snapshot.
8. A user can create a `PersistentVolumeClaim` using `VolumeSnapshot` as the source.
9. Trident identifies the required snapshot and performs the same set of steps involved in creating a `PersistentVolume` and a `Volume`.

For further reading about Kubernetes objects, we highly recommend that you read the Persistent Volumes section of the Kubernetes documentation.

Kubernetes `PersistentVolumeClaim` objects

A Kubernetes `PersistentVolumeClaim` object is a request for storage made by a Kubernetes cluster user.

In addition to the standard specification, Trident allows users to specify the following volume-specific annotations if they want to override the defaults that you set in the backend configuration:

Annotation	Volume Option	Supported Drivers
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	any
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

Annotation	Volume Option	Supported Drivers
trident.netapp.io/skipRecoveryQueue	skipRecoveryQueue	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy

If the created PV has the `Delete` reclaim policy, Trident deletes both the PV and the backing volume when the PV becomes released (that is, when the user deletes the PVC). Should the delete action fail, Trident marks the PV as such and periodically retries the operation until it succeeds or the PV is manually deleted. If the PV uses the `Retain` policy, Trident ignores it and assumes the administrator will clean it up from Kubernetes and the backend, allowing the volume to be backed up or inspected before its removal. Note that deleting the PV does not cause Trident to delete the backing volume. You should remove it using the REST API (`tridentctl`).

Trident supports the creation of Volume Snapshots using the CSI specification: you can create a Volume Snapshot and use it as a Data Source to clone existing PVCs. This way, point-in-time copies of PVs can be exposed to Kubernetes in the form of snapshots. The snapshots can then be used to create new PVs. Take a look at `On-Demand Volume Snapshots` to see how this would work.

Trident also provides the `cloneFromPVC` and `splitOnClone` annotations for creating clones. You can use these annotations to clone a PVC without having to use the CSI implementation.

Here is an example: If a user already has a PVC called `mysql`, the user can create a new PVC called `mysqlclone` by using the annotation, such as `trident.netapp.io/cloneFromPVC: mysql`. With this annotation set, Trident clones the volume corresponding to the mysql PVC, instead of provisioning a volume from scratch.

Consider the following points:

- NetApp recommends cloning an idle volume.
- A PVC and its clone should be in the same Kubernetes namespace and have the same storage class.
- With the `ontap-nas` and `ontap-san` drivers, it might be desirable to set the PVC annotation `trident.netapp.io/splitOnClone` in conjunction with `trident.netapp.io/cloneFromPVC`. With `trident.netapp.io/splitOnClone` set to `true`, Trident splits the cloned volume from the parent volume and thus, completely decoupling the life cycle of the cloned volume from its parent at the expense of losing some storage efficiency. Not setting `trident.netapp.io/splitOnClone` or setting it to `false` results in reduced space consumption on the backend at the expense of creating dependencies between the parent and clone volumes such that the parent volume cannot be deleted unless the clone is deleted first. A scenario where splitting the clone makes sense is cloning an empty database volume where it's expected for the volume and its clone to greatly diverge and not benefit from storage efficiencies offered by ONTAP.

The `sample-input` directory contains examples of PVC definitions for use with Trident. Refer to Trident `Volume` objects for a full description of the parameters and settings associated with Trident volumes.

Kubernetes `PersistentVolume` objects

A Kubernetes `PersistentVolume` object represents a piece of storage that is made available to the Kubernetes cluster. It has a lifecycle that is independent of the pod that uses it.

Trident creates `PersistentVolume` objects and registers them with the Kubernetes cluster automatically based on the volumes that it provisions. You are not expected to manage them yourself.

When you create a PVC that refers to a Trident-based `StorageClass`, Trident provisions a new volume using the corresponding storage class and registers a new PV for that volume. In configuring the provisioned volume and corresponding PV, Trident follows the following rules:

- Trident generates a PV name for Kubernetes and an internal name that it uses to provision the storage. In both cases, it is assuring that the names are unique in their scope.
- The size of the volume matches the requested size in the PVC as closely as possible, though it might be rounded up to the nearest allocatable quantity, depending on the platform.

Kubernetes `StorageClass` objects

Kubernetes `StorageClass` objects are specified by name in `PersistentVolumeClaims` to provision storage with a set of properties. The storage class itself identifies the provisioner to be used and defines that set of properties in terms the provisioner understands.

It is one of two basic objects that need to be created and managed by the administrator. The other is the Trident backend object.

A Kubernetes `StorageClass` object that uses Trident looks like this:

```
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
```

These parameters are Trident-specific and tell Trident how to provision volumes for the class.

The storage class parameters are:

Attribute	Type	Required	Description
attributes	map[string]string	no	See the attributes section below
storagePools	map[string]StringList	no	Map of backend names to lists of storage pools within
additionalStoragePools	map[string]StringList	no	Map of backend names to lists of storage pools within

Attribute	Type	Required	Description
excludeStoragePools	map[string]StringList	no	Map of backend names to lists of storage pools within

Storage attributes and their possible values can be classified into storage pool selection attributes and Kubernetes attributes.

Storage pool selection attributes

These parameters determine which Trident-managed storage pools should be utilized to provision volumes of a given type.

Attribute	Type	Values	Offer	Request	Supported by
media[1]	string	hdd, hybrid, ssd	Pool contains media of this type; hybrid means both	Media type specified	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san
provisioningType	string	thin, thick	Pool supports this provisioning method	Provisioning method specified	thick: all ontap; thin: all ontap & solidfire-san
backendType	string	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san, azure-netapp-files, ontap-san-economy	Pool belongs to this type of backend	Backend specified	All drivers
snapshots	bool	true, false	Pool supports volumes with snapshots	Volume with snapshots enabled	ontap-nas, ontap-san, solidfire-san
clones	bool	true, false	Pool supports cloning volumes	Volume with clones enabled	ontap-nas, ontap-san, solidfire-san
encryption	bool	true, false	Pool supports encrypted volumes	Volume with encryption enabled	ontap-nas, ontap-nas-economy, ontap-nas-flexgroups, ontap-san
IOPS	int	positive integer	Pool is capable of guaranteeing IOPS in this range	Volume guaranteed these IOPS	solidfire-san

[1]: Not supported by ONTAP Select systems

In most cases, the values requested directly influence provisioning; for instance, requesting thick provisioning results in a thickly provisioned volume. However, an Element storage pool uses its offered IOPS minimum and maximum to set QoS values, rather than the requested value. In this case, the requested value is used only to select the storage pool.

Ideally, you can use `attributes` alone to model the qualities of the storage you need to satisfy the needs of a particular class. Trident automatically discovers and selects storage pools that match *all* of the `attributes` that you specify.

If you find yourself unable to use `attributes` to automatically select the right pools for a class, you can use the `storagePools` and `additionalStoragePools` parameters to further refine the pools or even to select a specific set of pools.

You can use the `storagePools` parameter to further restrict the set of pools that match any specified `attributes`. In other words, Trident uses the intersection of pools identified by the `attributes` and `storagePools` parameters for provisioning. You can use either parameter alone or both together.

You can use the `additionalStoragePools` parameter to extend the set of pools that Trident uses for provisioning, regardless of any pools selected by the `attributes` and `storagePools` parameters.

You can use the `excludeStoragePools` parameter to filter the set of pools that Trident uses for provisioning. Using this parameter removes any pools that match.

In the `storagePools` and `additionalStoragePools` parameters, each entry takes the form `<backend>:<storagePoolList>`, where `<storagePoolList>` is a comma-separated list of storage pools for the specified backend. For example, a value for `additionalStoragePools` might look like `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`.
These lists accept regex values for both the backend and list values. You can use `tridentctl get backend` to get the list of backends and their pools.

Kubernetes attributes

These attributes have no impact on the selection of storage pools/backends by Trident during dynamic provisioning. Instead, these attributes simply supply parameters supported by Kubernetes Persistent Volumes. Worker nodes are responsible for filesystem create operations and might require filesystem utilities, such as xfsprogs.

Attribute	Type	Values	Description	Relevant Drivers	Kubernetes Version
fsType	string	ext4, ext3, xfs	The file system type for block volumes	solidfire-san, ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy	All

Attribute	Type	Values	Description	Relevant Drivers	Kubernetes Version
allowVolumeExpansion	boolean	true, false	Enable or disable support for growing the PVC size	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy, solidfire-san, azure-netapp-files	1.11+
volumeBindingMode	string	Immediate, WaitForFirstConsumer	Choose when volume binding and dynamic provisioning occurs	All	1.19 - 1.26

- The `fsType` parameter is used to control the desired file system type for SAN LUNs. In addition, Kubernetes also uses the presence of `fsType` in a storage class to indicate a filesystem exists. Volume ownership can be controlled using the `fsGroup` security context of a pod only if `fsType` is set. Refer to Kubernetes: Configure a Security Context for a Pod or Container for an overview on setting volume ownership using the `fsGroup` context. Kubernetes will apply the `fsGroup` value only if:
  - `fsType` is set in the storage class.
  - The PVC access mode is RWO.

  For NFS storage drivers, a filesystem already exists as part of the NFS export. In order to use `fsGroup` the storage class still needs to specify a `fsType`. You can set it to `nfs` or any non-null value.
- Refer to Expand volumes for further details on volume expansion.
- The Trident installer bundle provides several example storage class definitions for use with Trident in `sample-input/storage-class-*.yaml`. Deleting a Kubernetes storage class causes the corresponding Trident storage class to be deleted as well.

Kubernetes `VolumeSnapshotClass` objects

Kubernetes `VolumeSnapshotClass` objects are analogous to `StorageClasses`. They help define multiple classes of storage and are referenced by volume snapshots to associate the snapshot with the required snapshot class. Each volume snapshot is associated with a single volume snapshot class.

A `VolumeSnapshotClass` should be defined by an administrator in order to create snapshots. A volume snapshot class is created with the following definition:

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

The `driver` specifies to Kubernetes that requests for volume snapshots of the `csi-snapclass` class are handled by Trident. The `deletionPolicy` specifies the action to be taken when a snapshot must be deleted. When `deletionPolicy` is set to `Delete`, the volume snapshot objects as well as the underlying snapshot on the storage cluster are removed when a snapshot is deleted. Alternatively, setting it to `Retain` means that `VolumeSnapshotContent` and the physical snapshot are retained.

Kubernetes `VolumeSnapshot` objects

A Kubernetes `VolumeSnapshot` object is a request to create a snapshot of a volume. Just as a PVC represents a request made by a user for a volume, a volume snapshot is a request made by a user to create a snapshot of an existing PVC.

When a volume snapshot request comes in, Trident automatically manages the creation of the snapshot for the volume on the backend and exposes the snapshot by creating a unique `VolumeSnapshotContent` object. You can create snapshots from existing PVCs and use the snapshots as a DataSource when creating new PVCs.

> The lifecycle of a VolumeSnapshot is independent of the source PVC: a snapshot persists even after the source PVC is deleted. When deleting a PVC which has associated snapshots, Trident marks the backing volume for this PVC in a **Deleting** state, but does not remove it completely. The volume is removed when all the associated snapshots are deleted.

Kubernetes `VolumeSnapshotContent` objects

A Kubernetes `VolumeSnapshotContent` object represents a snapshot taken from an already provisioned volume. It is analogous to a `PersistentVolume` and signifies a provisioned snapshot on the storage cluster. Similar to `PersistentVolumeClaim` and `PersistentVolume` objects, when a snapshot is created, the `VolumeSnapshotContent` object maintains a one-to-one mapping to the `VolumeSnapshot` object, which had requested the snapshot creation.

The `VolumeSnapshotContent` object contains details that uniquely identify the snapshot, such as the `snapshotHandle`. This `snapshotHandle` is a unique combination of the name of the PV and the name of the `VolumeSnapshotContent` object.

When a snapshot request comes in, Trident creates the snapshot on the backend. After the snapshot is created, Trident configures a `VolumeSnapshotContent` object and thus exposes the snapshot to the Kubernetes API.

> Typically, you do not need to manage the `VolumeSnapshotContent` object. An exception to this is when you want to import a volume snapshot created outside of Trident.

Kubernetes `VolumeGroupSnapshotClass` objects

Kubernetes `VolumeGroupSnapshotClass` objects are analogous to `VolumeSnapshotClass`. They help define multiple classes of storage and are referenced by volume group snapshots to associate the snapshot with the required snapshot class. Each volume group snapshot is associated with a single volume group snapshot class.

A `VolumeGroupSnapshotClass` should be defined by an administrator in order to create group of snapshots. A volume group snapshot class is created with the following definition:

```
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
```

The `driver` specifies to Kubernetes that requests for volume group snapshots of the `csi-group-snap-class` class are handled by Trident. The `deletionPolicy` specifies the action to be taken when a group snapshot must be deleted. When `deletionPolicy` is set to `Delete`, the volume group snapshot objects as well as the underlying snapshot on the storage cluster are removed when a snapshot is deleted. Alternatively, setting it to `Retain` means that `VolumeGroupSnapshotContent` and the physical snapshot are retained.

Kubernetes `VolumeGroupSnapshot` objects

A Kubernetes `VolumeGroupSnapshot` object is a request to create a snapshot of a multiple volumes. Just as a PVC represents a request made by a user for a volume, a volume group snapshot is a request made by a user to create a snapshot of an existing PVC.

When a volume group snapshot request comes in, Trident automatically manages the creation of the group snapshot for the volumes on the backend and exposes the snapshot by creating a unique `VolumeGroupSnapshotContent` object. You can create snapshots from existing PVCs and use the snapshots as a DataSource when creating new PVCs.

> The lifecycle of a VolumeGroupSnapshot is independent of the source PVC: a snapshot persists even after the source PVC is deleted. When deleting a PVC which has associated snapshots, Trident marks the backing volume for this PVC in a **Deleting** state, but does not remove it completely. The volume group snapshot is removed when all the associated snapshots are deleted.

Kubernetes `VolumeGroupSnapshotContent` objects

A Kubernetes `VolumeGroupSnapshotContent` object represents a group snapshot taken from an already provisioned volume. It is analogous to a `PersistentVolume` and signifies a provisioned snapshot on the storage cluster. Similar to `PersistentVolumeClaim` and `PersistentVolume` objects, when a snapshot is created, the `VolumeSnapshotContent` object maintains a one-to-one mapping to the `VolumeSnapshot` object, which had requested the snapshot creation.

The `VolumeGroupSnapshotContent` object contains details that identify the snapshot group, such as the `volumeGroupSnapshotHandle` and and individual volumeSnapshotHandles existing on the storage system.

When a snapshot request comes in, Trident creates the volume group snapshot on the backend. After the volume group snapshot is created, Trident configures a `VolumeGroupSnapshotContent` object and thus exposes the snapshot to the Kubernetes API.

Kubernetes `CustomResourceDefinition` objects

Kubernetes Custom Resources are endpoints in the Kubernetes API that are defined by the administrator and are used to group similar objects. Kubernetes supports the creation of custom resources for storing a collection of objects. You can obtain these resource definitions by running `kubectl get crds`.

Custom Resource Definitions (CRDs) and their associated object metadata are stored by Kubernetes in its metadata store. This eliminates the need for a separate store for Trident.

Trident uses `CustomResourceDefinition` objects to preserve the identity of Trident objects, such as Trident backends, Trident storage classes, and Trident volumes. These objects are managed by Trident. In addition, the CSI volume snapshot framework introduces some CRDs that are required to define volume snapshots.

CRDs are a Kubernetes construct. Objects of the resources defined above are created by Trident. As a simple example, when a backend is created using `tridentctl`, a corresponding `tridentbackends` CRD object is created for consumption by Kubernetes.

Here are a few points to keep in mind about Trident's CRDs:

- When Trident is installed, a set of CRDs are created and can be used like any other resource type.
- When uninstalling Trident by using the `tridentctl uninstall` command, Trident pods are deleted but the created CRDs are not cleaned up. Refer to Uninstall Trident to understand how Trident can be completely removed and reconfigured from scratch.

Trident `StorageClass` objects

Trident creates matching storage classes for Kubernetes `StorageClass` objects that specify `csi.trident.netapp.io` in their provisioner field. The storage class name matches that of the Kubernetes `StorageClass` object it represents.

With Kubernetes, these objects are created automatically when a Kubernetes `StorageClass` that uses Trident as a provisioner is registered.

Storage classes comprise a set of requirements for volumes. Trident matches these requirements with the attributes present in each storage pool; if they match, that storage pool is a valid target for provisioning volumes using that storage class.

You can create storage class configurations to directly define storage classes by using the REST API. However, for Kubernetes deployments, we expect them to be created when registering new Kubernetes `StorageClass` objects.

Trident backend objects

Backends represent the storage providers on top of which Trident provisions volumes; a single Trident instance can manage any number of backends.

This is one of the two object types that you create and manage yourself. The other is the Kubernetes `StorageClass` object.

For more information about how to construct these objects, refer to configuring backends.

Trident `StoragePool` objects

Storage pools represent the distinct locations available for provisioning on each backend. For ONTAP, these correspond to aggregates in SVMs. For NetApp HCI/SolidFire, these correspond to administrator-specified QoS bands. Each storage pool has a set of distinct storage attributes, which define its performance characteristics and data protection characteristics.

Unlike the other objects here, storage pool candidates are always discovered and managed automatically.

Trident `Volume` objects

Volumes are the basic unit of provisioning, comprising backend endpoints, such as NFS shares, and iSCSI and FC LUNs. In Kubernetes, these correspond directly to `PersistentVolumes`. When you create a volume, ensure that it has a storage class, which determines where that volume can be provisioned, along with a size.

- In Kubernetes, these objects are managed automatically. You can view them to see what Trident provisioned.
- When deleting a PV with associated snapshots, the corresponding Trident volume is updated to a **Deleting** state. For the Trident volume to be deleted, you should remove the snapshots of the volume.

A volume configuration defines the properties that a provisioned volume should have.

Attribute	Type	Required	Description
version	string	no	Version of the Trident API ("1")
name	string	yes	Name of volume to create
storageClass	string	yes	Storage class to use when provisioning the volume
size	string	yes	Size of the volume to provision in bytes
protocol	string	no	Protocol type to use; "file" or "block"
internalName	string	no	Name of the object on the storage system; generated by Trident

Attribute	Type	Required	Description
cloneSourceVolume	string	no	ontap (nas, san) & solidfire-*: Name of the volume to clone from
splitOnClone	string	no	ontap (nas, san): Split the clone from its parent
snapshotPolicy	string	no	ontap-*: Snapshot policy to use
snapshotReserve	string	no	ontap-*: Percentage of volume reserved for snapshots
exportPolicy	string	no	ontap-nas*: Export policy to use
snapshotDirectory	bool	no	ontap-nas*: Whether the snapshot directory is visible
unixPermissions	string	no	ontap-nas*: Initial UNIX permissions
blockSize	string	no	solidfire-*: Block/sector size
fileSystem	string	no	File system type
skipRecoveryQueue	string	no	During volume deletion, bypass the recovery queue in storage and delete the volume immediately.

Trident generates `internalName` when creating the volume. This consists of two steps. First, it prepends the storage prefix (either the default `trident` or the prefix in the backend configuration) to the volume name, resulting in a name of the form `<prefix>-<volume-name>`. It then proceeds to sanitize the name, replacing characters not permitted in the backend. For ONTAP backends, it replaces hyphens with underscores (thus, the internal name becomes `<prefix>_<volume-name>`). For Element backends, it replaces underscores with hyphens.

You can use volume configurations to directly provision volumes using the REST API, but in Kubernetes deployments we expect most users to use the standard Kubernetes `PersistentVolumeClaim` method. Trident creates this volume object automatically as part of the provisioning process.

Trident `Snapshot` objects

Snapshots are a point-in-time copy of volumes, which can be used to provision new volumes or restore state. In Kubernetes, these correspond directly to `VolumeSnapshotContent` objects. Each snapshot is associated with a volume, which is the source of the data for the snapshot.

Each `Snapshot` object includes the properties listed below:

Attribute	Type	Required	Description
version	String	Yes	Version of the Trident API ("1")
name	String	Yes	Name of the Trident snapshot object
internalName	String	Yes	Name of the Trident snapshot object on the storage system
volumeName	String	Yes	Name of the Persistent Volume for which the snapshot is created
volumeInternalName	String	Yes	Name of the associated Trident volume object on the storage system

In Kubernetes, these objects are managed automatically. You can view them to see what Trident provisioned.

When a Kubernetes `VolumeSnapshot` object request is created, Trident works by creating a snapshot object on the backing storage system. The `internalName` of this snapshot object is generated by combining the prefix `snapshot-` with the `UID` of the `VolumeSnapshot` object (for example, `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`). `volumeName` and `volumeInternalName` are populated by getting the details of the backing
volume.

Trident `ResourceQuota` object

The Trident deamonset consumes a `system-node-critical` Priority Class—the highest Priority Class available in Kubernetes—to ensure Trident can identify and clean up volumes during graceful node shutdown and allow Trident daemonset pods to preempt workloads with a lower priority in clusters where there is high resource pressure.

To accomplish this, Trident employs a `ResourceQuota` object to ensure a "system-node-critical" Priority Class on the Trident daemonset is satisfied. Prior to deployment and daemonset creation, Trident looks for the `ResourceQuota` object and, if not discovered, applies it.

If you need more control over the default Resource Quota and Priority Class, you can generate a `custom.yaml` or configure the `ResourceQuota` object using Helm chart.

The following is an example of a `ResourceQuota`object prioritizing the Trident daemonset.

```
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
```

For more information on Resource Quotas, refer to Kubernetes: Resource Quotas.

Clean up `ResourceQuota` if installation fails

In the rare case where installation fails after the `ResourceQuota` object is created, first try uninstalling and then reinstall.

If that doesn't work, manually remove the `ResourceQuota` object.

Remove `ResourceQuota`

If you prefer to control your own resource allocation, you can remove the Trident `ResourceQuota` object using the command:

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

Pod Security Standards (PSS) and Security Context Constraints (SCC)

Kubernetes Pod Security Standards (PSS) and Pod Security Policies (PSP) define permission levels and restrict the behavior of pods. OpenShift Security Context Constraints (SCC) similarly define pod restriction specific to the OpenShift Kubernetes Engine. To provide this customization, Trident enables certain permissions during installation. The following sections detail the permissions set by Trident.

PSS replaces Pod Security Policies (PSP). PSP was deprecated in Kubernetes v1.21 and will be removed in v1.25. For more information, Refer to Kubernetes: Security.

Required Kubernetes Security Context and Related Fields

Permission	Description
Privileged	CSI requires mount points to be Bidirectional, which means the Trident node pod must run a privileged container. For more information, refer to Kubernetes: Mount propagation.
Host networking	Required for the iSCSI daemon. `iscsiadm` manages iSCSI mounts and uses host networking to communicate with the iSCSI daemon.
Host IPC	NFS uses interprocess communication (IPC) to communicate with the NFSD.
Host PID	Required to start `rpc-statd` for NFS. Trident queries host processes to determine if `rpc-statd` is running before mounting NFS volumes.

Capabilities	The `SYS_ADMIN` capability is provided as part of the default capabilities for privileged containers. For example, Docker sets these capabilities for privileged containers:
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccomp profile is always "Unconfined" in privileged containers; therefore, it cannot be enabled in Trident.
SELinux	On OpenShift, privileged containers are run in the `spc_t` ("Super Privileged Container") domain, and unprivileged containers are run in the `container_t` domain. On `containerd`, with `container-selinux` installed, all containers are run in the `spc_t` domain, which effectively disables SELinux. Therefore, Trident does not add `seLinuxOptions` to containers.
DAC	Privileged containers must be run as root. Non-privileged containers run as root to access unix sockets required by CSI.

Pod Security Standards (PSS)

Label	Description	Default

`pod-security.kubernetes.io/enforce`
 	
`pod-security.kubernetes.io/enforce-version`
	Allows the Trident Controller and nodes to be admitted into the install namespace.
	
	Do not change the namespace label.
		`enforce: privileged`
		
		`enforce-version: <version of the current cluster or highest version of PSS tested.>`

Changing the namespace labels can result in pods not being scheduled, an "Error creating: …" or, "Warning: trident-csi-…". If this happens, check if the namespace label for `privileged` was changed. If so, reinstall Trident.

Pod Security Policies (PSP)

Field	Description	Default
`allowPrivilegeEscalation`	Privileged containers must allow privilege escalation.	`true`
`allowedCSIDrivers`	Trident does not use inline CSI ephemeral volumes.	Empty
`allowedCapabilities`	Non-privileged Trident containers do not require more capabilities than the default set and privileged containers are granted all possible capabilities.	Empty
`allowedFlexVolumes`	Trident does not make use of a FlexVolume driver, therefore they are not included in the list of allowed volumes.	Empty
`allowedHostPaths`	The Trident node pod mounts the node's root filesystem, therefore there is no benefit to setting this list.	Empty
`allowedProcMountTypes`	Trident does not use any `ProcMountTypes`.	Empty
`allowedUnsafeSysctls`	Trident does not require any unsafe `sysctls`.	Empty
`defaultAddCapabilities`	No capabilities are required to be added to privileged containers.	Empty
`defaultAllowPrivilegeEscalation`	Allowing privilege escalation is handled in each Trident pod.	`false`
`forbiddenSysctls`	No `sysctls` are allowed.	Empty
`fsGroup`	Trident containers run as root.	`RunAsAny`
`hostIPC`	Mounting NFS volumes requires host IPC to communicate with `nfsd`	`true`
`hostNetwork`	iscsiadm requires the host network to communicate with the iSCSI daemon.	`true`
`hostPID`	Host PID is required to check if `rpc-statd` is running on the node.	`true`
`hostPorts`	Trident does not use any host ports.	Empty

Field	Description	Default
`privileged`	Trident node pods must run a privileged container in order to mount volumes.	`true`
`readOnlyRootFilesystem`	Trident node pods must write to the node filesystem.	`false`
`requiredDropCapabilities`	Trident node pods run a privileged container and cannot drop capabilities.	`none`
`runAsGroup`	Trident containers run as root.	`RunAsAny`
`runAsUser`	Trident containers run as root.	`runAsAny`
`runtimeClass`	Trident does not use `RuntimeClasses`.	Empty
`seLinux`	Trident does not set `seLinuxOptions` because there are currently differences in how container runtimes and Kubernetes distributions handle SELinux.	Empty
`supplementalGroups`	Trident containers run as root.	`RunAsAny`
`volumes`	Trident pods require these volume plugins.	`hostPath, projected, emptyDir`

Security Context Constraints (SCC)

Labels	Description	Default
`allowHostDirVolumePlugin`	Trident node pods mount the node's root filesystem.	`true`
`allowHostIPC`	Mounting NFS volumes requires host IPC to communicate with `nfsd`.	`true`
`allowHostNetwork`	iscsiadm requires the host network to communicate with the iSCSI daemon.	`true`
`allowHostPID`	Host PID is required to check if `rpc-statd` is running on the node.	`true`
`allowHostPorts`	Trident does not use any host ports.	`false`
`allowPrivilegeEscalation`	Privileged containers must allow privilege escalation.	`true`
`allowPrivilegedContainer`	Trident node pods must run a privileged container in order to mount volumes.	`true`

Labels	Description	Default
`allowedUnsafeSysctls`	Trident does not require any unsafe `sysctls`.	`none`
`allowedCapabilities`	Non-privileged Trident containers do not require more capabilities than the default set and privileged containers are granted all possible capabilities.	Empty
`defaultAddCapabilities`	No capabilities are required to be added to privileged containers.	Empty
`fsGroup`	Trident containers run as root.	`RunAsAny`
`groups`	This SCC is specific to Trident and is bound to its user.	Empty
`readOnlyRootFilesystem`	Trident node pods must write to the node filesystem.	`false`
`requiredDropCapabilities`	Trident node pods run a privileged container and cannot drop capabilities.	`none`
`runAsUser`	Trident containers run as root.	`RunAsAny`
`seLinuxContext`	Trident does not set `seLinuxOptions` because there are currently differences in how container runtimes and Kubernetes distributions handle SELinux.	Empty
`seccompProfiles`	Privileged containers always run "Unconfined".	Empty
`supplementalGroups`	Trident containers run as root.	`RunAsAny`
`users`	One entry is provided to bind this SCC to the Trident user in the Trident namespace.	n/a
`volumes`	Trident pods require these volume plugins.	`hostPath, downwardAPI, projected, emptyDir`

Legal notices

Legal notices provide access to copyright statements, trademarks, patents, and more.

Copyright

https://www.netapp.com/company/legal/copyright/

Trademarks

NETAPP, the NETAPP logo, and the marks listed on the NetApp Trademarks page are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

https://www.netapp.com/company/legal/trademarks/

Patents

A current list of NetApp owned patents can be found at:

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

Privacy policy

https://www.netapp.com/company/legal/privacy-policy/

Open source

You can review third-party copyright and licenses used in NetApp software for Trident in the notices file for each release at https://github.com/NetApp/trident/.

Copyright information

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

Trademark information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.