



Astra Trident 22.07 documentation

Astra Trident

NetApp
August 26, 2022

This PDF was generated from <https://docs.netapp.com/us-en/trident/index.html> on August 26, 2022.
Always check docs.netapp.com for the latest.

Table of Contents

Astra Trident 22.07 documentation	1
Release Notes	2
What's new in 22.07	2
Changes in 22.04	3
Changes in 22.01.1	3
Changes in 22.01.0	4
Changes in 21.10.1	4
Changes in 21.10.0	4
Known issues	5
Find more information	6
Concepts	7
Learn about Astra Trident	7
ONTAP drivers	8
Provisioning	8
Volume snapshots	9
Virtual storage pools	9
Volume access groups	11
Get started	12
Try it out	12
Requirements	12
Deployment overview	16
Deploy with Trident operator	19
Deploy with <code>tridentctl</code>	28
What's next?	31
Manage Astra Trident	37
Upgrade Astra Trident	37
Upgrade with the operator	38
Upgrade with <code>tridentctl</code>	46
Uninstall Astra Trident	49
Downgrade Astra Trident	51
Use Astra Trident	55
Configure backends	55
Create backends with <code>kubectl</code>	122
Perform backend management with <code>kubectl</code>	129
Perform backend management with <code>tridentctl</code>	130
Move between backend management options	131
Manage storage classes	138
Perform volume operations	140
Prepare the worker node	166
Monitor Astra Trident	170
Astra Trident for Docker	174
Prerequisites for deployment	174

Deploy Astra Trident	176
Upgrade or uninstall Astra Trident	181
Work with volumes	183
Collect logs	192
Manage multiple Astra Trident instances	193
Storage configuration options	194
Known issues and limitations	209
Frequently asked questions	211
General questions	211
Install and use Astra Trident on a Kubernetes cluster	211
Troubleshooting and support	213
Upgrade Astra Trident	214
Manage backends and volumes	214
Support	219
Troubleshooting	220
General troubleshooting	220
Troubleshooting an unsuccessful Trident deployment using the operator	221
Troubleshooting an unsuccessful Trident deployment using <code>tridentctl</code>	223
Best practices and recommendations	225
Deployment	225
Storage configuration	225
Integrate Astra Trident	232
Data protection	242
Security	247
Reference	249
Astra Trident ports	249
Astra Trident REST API	249
Command-line options	250
NetApp products integrated with Kubernetes	251
Kubernetes and Trident objects	252
<code>tridentctl</code> commands and options	264
Pod Security Standards (PSS) and Security Context Constraints (SCC)	269

Astra Trident 22.07 documentation

Release Notes

Release Notes provide information about new features, enhancements, and bug fixes in the latest version of Astra 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 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 ANF 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.
- Astra Data Store (ADS) driver updated to v1beta1 CRDs, so this version of Trident requires ADS 22.5.0 or later.
- 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 Astra 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 Astra Trident. For previous releases, see [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 ANF 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 Data LIFs 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 Astra Trident performance slows down when > 32 snapshots are created on a SolidFire volume.
- Replaced SHA-1 with SHA-256 in SSL certificate creation.
- Fixed ANF driver to allow duplicate resource names and limit operations to a single location.
- Fixed ANF 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 ANF 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 Astra 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 Astra 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.

- Astra 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 Astra Trident instances, each backend configuration file should

have a different `storagePrefix` value for ONTAP backends or use a different `TenantName` for SolidFire backends. Astra Trident cannot detect volumes that other instances of Astra Trident have created. Attempting to create an existing volume on either ONTAP or SolidFire backends succeeds, because Astra 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 Astra Trident (using `tridentctl` or the Trident Operator) and using `tridentctl` to manage Astra 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/RedHat CoreOS 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 Astra Trident per Kubernetes cluster, Astra 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 Astra Trident per Kubernetes cluster.
- If Astra Trident-based `StorageClass` objects are deleted from Kubernetes while Astra Trident is offline, Astra 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 Astra Trident before deleting the corresponding PVC, Astra 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 Astra 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]`.
- If using the `solidfire-san` driver with OpenShift 4.5, ensure that the underlying worker nodes use MD5 as the CHAP authentication algorithm.

Find more information

- [Astra Trident GitHub](#)
- [Astra Trident blogs](#)

Concepts

Learn about Astra Trident

Astra Trident is a fully supported open source project maintained by NetApp as part of the [Astra product family](#). It has been designed to help you meet your containerized applications' persistence demands using industry-standard interfaces, such as the Container Storage Interface (CSI).

Astra Trident deploys in Kubernetes clusters as pods and provides dynamic storage orchestration services for your Kubernetes workloads. It enables your containerized applications to quickly and easily consume persistent storage from NetApp's broad portfolio that includes ONTAP (AFF/FAS/Select/Cloud/Amazon FSx for NetApp ONTAP), Element software (NetApp HCI/SolidFire), Astra Data Store, as well as the Azure NetApp Files service, and Cloud Volumes Service on Google Cloud.

Astra Trident is also a foundational technology for NetApp's Astra, which addresses your data protection, disaster recovery, portability, and migration use cases for Kubernetes workloads leveraging NetApp's industry-leading data management technology for snapshots, backups, replication, and cloning.

Supported Kubernetes cluster architectures

Astra 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

What is Astra?

Astra makes it easier for enterprises to manage, protect, and move their data-rich containerized workloads running on Kubernetes within and across public clouds and on-premises. Astra provisions and provides persistent container storage using Astra Trident from NetApp's proven and expansive storage portfolio in the public cloud and on-premises. It also offers a rich set of advanced application-aware data management functionality, such as snapshot, backup and restore, activity logs, and active cloning for data protection, disaster/data recovery, data audit, and migration use-cases for Kubernetes workloads.

You can sign up for a free trial on the [Astra page](#).

For more information

- [NetApp Astra product family](#)
- [Astra Control Service documentation](#)
- [Astra Control Center documentation](#)
- [Astra Data Store documentation](#)

ONTAP drivers

Astra Trident provides five unique ONTAP storage drivers for communicating with ONTAP clusters. Learn more about how each driver handles the creation of volumes and access control and their capabilities.

Driver	Protocol	VolumeMode	Access modes supported	File systems supported
ontap-nas	NFS	Filesystem	RWO,RWX,ROX	"", nfs
ontap-nas-economy	NFS	Filesystem	RWO,RWX,ROX	"", nfs
ontap-nas-flexgroup	NFS	Filesystem	RWO,RWX,ROX	"", nfs
ontap-san	iSCSI	Block	RWO,ROX,RWX	No Filesystem. Raw block device
ontap-san	iSCSI	Filesystem	RWO,ROX	xfs, ext3, ext4
ontap-san-economy	iSCSI	Block	RWO,ROX,RWX	No Filesystem. Raw block device
ontap-san-economy	iSCSI	Filesystem	RWO,ROX	xfs, ext3, ext4



ONTAP backends can be authenticated by using login credentials for a security role (username/password) or using the private key and the certificate that is installed on the ONTAP cluster. You can update existing backends to move from one authentication mode to the other with `tridentctl update backend`.

Provisioning

Provisioning in Astra 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.

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, Astra Trident adds that storage pool to the set of suitable storage pools for that storage class. In addition, Astra 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. Astra 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.

Astra Trident then uses the associations between storage classes and storage pools to determine where to provision volumes. When you create a volume, Astra Trident first gets the set of storage pools for that volume's storage class, and, if you specify a protocol for the volume, Astra 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). Astra 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. Astra 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 Astra Trident handles the creation of volume snapshots for its drivers.

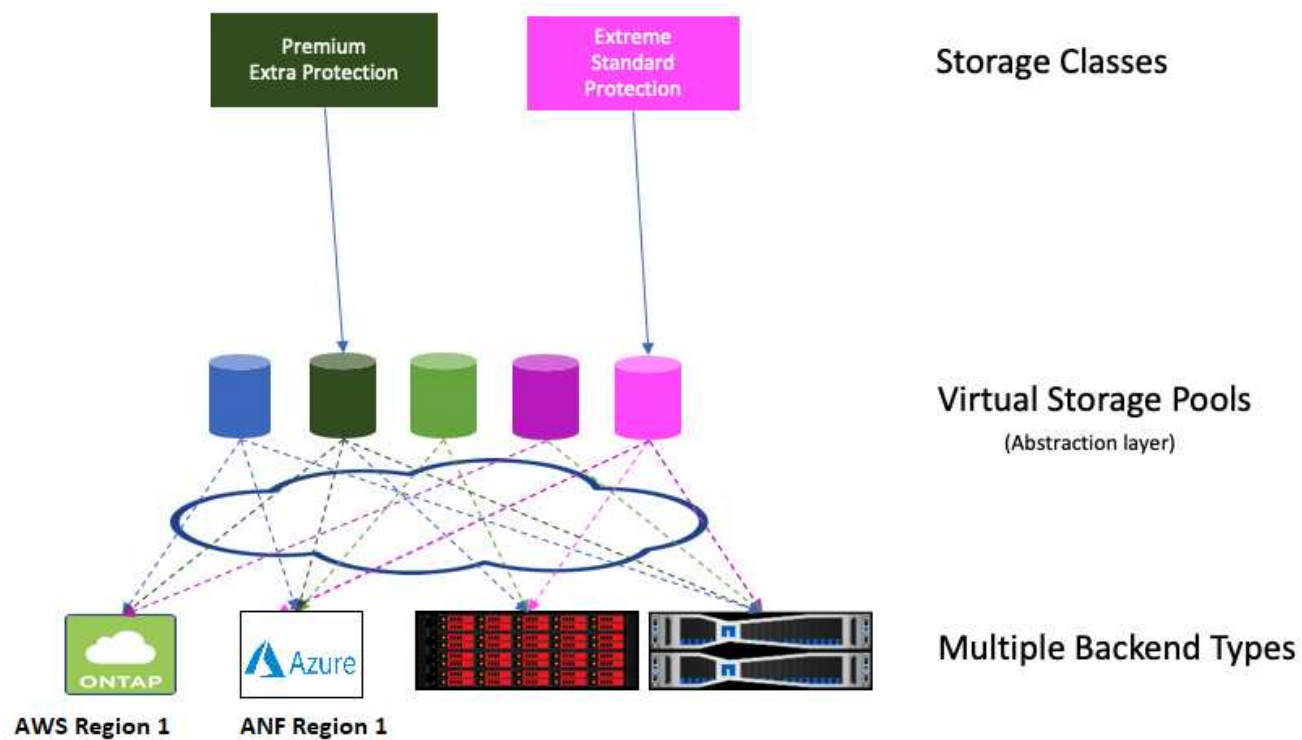
- For the `ontap-nas`, `ontap-san`, `gcp-cvs`, and `azure-netapp-files` drivers, each Persistent Volume (PV) maps to a FlexVol. As a result, volume snapshots are created as NetApp snapshots. NetApp's snapshot technology delivers more stability, scalability, recoverability, and performance than competing 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's 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 FlexVols. VolumeSnapshots of PVs are achieved by performing FlexClones of the associated LUN. ONTAP's 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, Astra Trident will delete the associated ONTAP snapshot. ONTAP snapshots that were not created through Kubernetes can also be deleted.

With Astra 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 and CVS 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 storage pools

Virtual storage pools provide a layer of abstraction between Astra Trident's 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.

The storage administrator can define virtual storage pools on any of the Astra 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 storage pools, do 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.

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.

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

Operator	Example	Description
=	performance=premium	A pool’s label value must match
!=	performance!=extreme	A pool’s label value must not match
in	location in (east, west)	A pool’s label value must be in the set of values

Operator	Example	Description
notin	performance notin (silver, bronze)	A pool's label value must not b

Volume access groups

Learn more about how Astra 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 Astra Trident in CSI mode, you can ignore this section. Astra Trident uses CHAP when installed as an enhanced CSI provisioner.

Astra 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 Astra Trident associates new volumes with the configured access group(s), it does not create or otherwise manage access groups themselves. The access group(s) must exist before the storage backend is added to Astra 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, see [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.

Get started

Try it out

NetApp provides a ready-to-use lab image that you can request through [NetApp Test Drive](#). The Test Drive provides you with a sandbox environment that comes with a three-node Kubernetes cluster and Astra Trident installed and configured. It is a great way to familiarize yourself with Astra Trident and explore its features.

Another option is to see the [kubeadm Install Guide](#) provided by Kubernetes.



You should not use the Kubernetes cluster that you build using these instructions in production. Use the production deployment guides provided by your distribution for creating clusters that are production ready.

If this is the first time you're using Kubernetes, familiarize yourself with the concepts and tools [here](#).

Requirements

Get started by reviewing the supported frontends, backends, and host configuration.



To learn about the ports that Astra Trident uses, see [here](#).

Supported frontends (orchestrators)

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

- Anthos On-Prem (VMware) and Anthos on bare metal 1.9, 1.10, 1.11
- Kubernetes 1.19 - 1.24
- Mirantis Kubernetes Engine 3.5
- OpenShift 4.7 (support ends August 24, 2022), 4.8, 4.9, 4.10

The Trident operator is supported with these releases:

- Anthos On-Prem (VMware) and Anthos on bare metal 1.9, 1.10, 1.11
- Kubernetes 1.19 - 1.24
- OpenShift 4.7 (support ends August 24, 2022), 4.8, 4.9, 4.10

Astra 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), Rancher, and VMWare Tanzu Portfolio.

Supported backends (storage)

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

- Amazon FSx for NetApp ONTAP
- Azure NetApp Files
- Astra Data Store 22.05 or above

- Cloud Volumes ONTAP
- Cloud Volumes Service for GCP
- FAS/AFF/Select 9.3 or later
- NetApp All SAN Array (ASA)
- NetApp HCI/Element software 11 or above

Feature requirements

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

Feature	Kubernetes version	Feature gates required?
CSI Trident	1.19 - 1.24	No
Volume Snapshots	1.19 - 1.24	No
PVC from Volume Snapshots	1.19 - 1.24	No
iSCSI PV resize	1.19 - 1.24	No
ONTAP Bidirectional CHAP	1.19 - 1.24	No
Dynamic Export Policies	1.19 - 1.24	No
Trident Operator	1.19 - 1.24	No
Auto Worker Node Prep (beta)	1.19 - 1.24	No
CSI Topology	1.19 - 1.24	No

Tested host operating systems

By default, Astra 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 Astra Trident provides using the standard NFS client or iSCSI initiator, depending on the backends you are using.

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

- RedHat CoreOS (RHCOS) versions as supported by OpenShift Container Platform
- RHEL or CentOS 7
- Ubuntu 18.04 or later (latest 22.04)

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

Host configuration

Depending on the backend(s) in use, NFS and/or iSCSI utilities should be installed on all of the workers in the cluster. See [here](#) for more information.

Storage system configuration

Astra Trident might require some changes to a storage system before a backend configuration can use it. See [here](#) for details.

Container images and corresponding Kubernetes versions

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

Kubernetes version	Container image
v1.19.0	<ul style="list-style-type: none">• netapp/trident:22.07.0• netapp/trident-autosupport:22.07• k8s.gcr.io/sig-storage/csi-provisioner:v2.2.2• k8s.gcr.io/sig-storage/csi-attacher:v3.5.0• k8s.gcr.io/sig-storage/csi-resizer:v1.5.0• k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3• k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1• netapp/trident-operator:22.07.0 (optional)
v1.20.0	<ul style="list-style-type: none">• netapp/trident:22.07.0• netapp/trident-autosupport:22.07• k8s.gcr.io/sig-storage/csi-provisioner:v3.2.1• k8s.gcr.io/sig-storage/csi-attacher:v3.5.0• k8s.gcr.io/sig-storage/csi-resizer:v1.5.0• k8s.gcr.io/sig-storage/csi-snapshotter:v6.0.1• k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1• netapp/trident-operator:22.07.0 (optional)

Kubernetes version	Container image
v1.21.0	<ul style="list-style-type: none"> • netapp/trident:22.07.0 • netapp/trident-autosupport:22.07 • k8s.gcr.io/sig-storage/csi-provisioner:v3.2.1 • k8s.gcr.io/sig-storage/csi-attacher:v3.5.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.5.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v6.0.1 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1 • netapp/trident-operator:22.07.0 (optional)
v1.22.0	<ul style="list-style-type: none"> • netapp/trident:22.07.0 • netapp/trident-autosupport:22.07 • k8s.gcr.io/sig-storage/csi-provisioner:v3.2.1 • k8s.gcr.io/sig-storage/csi-attacher:v3.5.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.5.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v6.0.1 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1 • netapp/trident-operator:22.07.0 (optional)
v1.23.0	<ul style="list-style-type: none"> • netapp/trident:22.07.0 • netapp/trident-autosupport:22.07 • k8s.gcr.io/sig-storage/csi-provisioner:v3.2.1 • k8s.gcr.io/sig-storage/csi-attacher:v3.5.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.5.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v6.0.1 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1 • netapp/trident-operator:22.07.0 (optional)

Kubernetes version	Container image
v1.24.0	<ul style="list-style-type: none"> • netapp/trident:22.07.0 • netapp/trident-autosupport:22.07 • k8s.gcr.io/sig-storage/csi-provisioner:v3.2.1 • k8s.gcr.io/sig-storage/csi-attacher:v3.5.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.5.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v6.0.1 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.5.1 • netapp/trident-operator:22.07.0 (optional)



On Kubernetes version 1.20 and above, use the validated `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x` image only if the `v1` version is serving the `volumesnapshots.snapshot.storage.k8s.gcr.io` CRD. If the `v1beta1` version is serving the CRD with/without the `v1` version, use the validated `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x` image.

Deployment overview

You can deploy Astra Trident using the Trident operator or with `tridentctl`.



Beginning with the 22.04 release, AES keys will no longer be regenerated every time Astra Trident is installed. With this release, Astra 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.

Choose the deployment method

To determine which deployment method to use, consider the following:

Why should I use the Trident operator?

The [Trident operator](#) is a great way to dynamically manage Astra Trident resources and automate the setup phase. There are some prerequisites that must be satisfied. See [the requirements](#).

The Trident operator provides several benefits as outlined below.

Self-healing capability

You can monitor an Astra Trident installation and actively take measures to address issues, such as when the deployment is deleted or if it is modified accidentally. When the operator is set up as a deployment, a `trident-operator-<generated-id>` pod is created. This pod associates a `TridentOrchestrator` CR with an Astra Trident installation and always ensures there is only one active `TridentOrchestrator`. In other words, the operator ensures that there is only one instance of Astra 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 Astra 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.

Automatically handles Kubernetes upgrades

When the Kubernetes version of the cluster is upgraded to a supported version, the operator updates an existing Astra 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 Astra Trident. If Astra Trident has already been installed with the operator, a warning is displayed to indicate that Astra Trident is installed on an unsupported Kubernetes version.

Manage your Kubernetes clusters using Cloud Manager

With [Astra Trident using Cloud Manager](#), you can upgrade to the latest version of Astra Trident, add and manage storage classes and connect them to Working Environments, and back up persistent volumes using Cloud Backup Service. Cloud Manager supports Astra Trident deployment using the Trident operator, either manually or using Helm.

Why should I use Helm?

If you have other applications that you are managing using Helm, starting with Astra Trident 21.01, you can manage your deployment also using Helm.

When should I use `tridentctl`?

If you have an existing deployment that must be upgraded to or if you are looking to highly customize your deployment, you should take a look at using [tridentctl](#). This is the conventional method of deploying Astra Trident.

Considerations for moving between deployment methods

It is not hard to imagine a scenario where moving between deployment methods is desired. You should consider the following before attempting to move from a `tridentctl` deployment to an operator-based deployment, or vice-versa:

- Always use the same method for uninstalling Astra Trident. If you have deployed with `tridentctl`, you should use the appropriate version of the `tridentctl` binary to uninstall Astra Trident. Similarly, if you are deploying with the operator, you should edit the `TridentOrchestrator` CR and set `spec.uninstall=true` to uninstall Astra Trident.

- If you have an operator-based deployment that you want to remove and use `tridentctl` to deploy Astra Trident, you should first edit `TridentOrchestrator` and set `spec.uninstall=true` to uninstall Astra 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 do 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 use Helm-based deployment without running into issues.

Understand the deployment modes

There are three ways to deploy Astra Trident.

Standard deployment

Deploying Trident on a Kubernetes cluster results in the Astra Trident installer doing two things:

- Fetching the container images over the Internet
- Creating a deployment and/or node daemonset, which spins up Astra Trident pods on all the eligible nodes in the Kubernetes cluster.

A standard deployment such as this can be performed in two different ways:

- Using `tridentctl install`
- Using the Trident operator. You can deploy Trident operator either manually or by using Helm.

This mode of installing is the easiest way to install Astra Trident and works for most environments that do not impose network restrictions.

Offline deployment

To perform an air-gapped deployment, you can use the `--image-registry` flag when invoking `tridentctl install` to point to a private image registry. If deploying with the Trident operator, you can alternatively specify `spec.imageRegistry` in your `TridentOrchestrator`. This registry should contain the [Trident image](#), the [Trident Autosupport image](#), and the CSI sidecar images as required by your Kubernetes version.

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

See these links for more information about customizing your deployment:

- [Customize your operator-based deployment](#)
- [Customize your `tridentctl`-based deployment](#)



If you are using a private image repository, you should add `/sig-storage` to the end of the private registry URL. When using a private registry for `tridentctl` deployment, you should use `--trident-image` and `--autosupport-image` in conjunction with `--image-registry`. If you are deploying Astra Trident by using the Trident operator, ensure that the orchestrator CR includes `tridentImage` and `autosupportImage` in the installation parameters.

Remote deployment

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

- Deploy the appropriate version of `kubectl` on the remote machine from where you want to deploy Astra 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.

Other known configuration options

When installing Astra Trident on VMWare Tanzu Portfolio products:

- The cluster must support privileged workloads.
- 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.

Deploy with Trident operator

You can deploy Astra Trident using the Trident operator. You can deploy the Trident operator in one of two ways:

- Using the Trident [Helm Chart](#): The Helm Chart deploys the Trident operator and installs Trident in one step.
- Manually: Trident provides a [bundle.yaml](#) file that can be used to install the operator and create associated objects.



If you have not already familiarized yourself with the [basic concepts](#), now is a great time to do that.

What you'll need

To deploy Astra Trident, the following prerequisites should be met:

- You have full privileges to a supported Kubernetes cluster running Kubernetes 1.19 - 1.24.
- You have access to a supported NetApp storage system.
- You have the capability to mount volumes from all of the Kubernetes worker nodes.

- You have 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.
- You have set the `KUBECONFIG` environment variable to point to your Kubernetes cluster configuration.
- You have enabled the [feature gates required by Astra Trident](#).
- If you are using Kubernetes with Docker Enterprise, [follow their steps to enable CLI access](#).

Got all that? Great! Let's get started.

Deploy the Trident operator and install Astra Trident using Helm

Perform the steps listed to deploy the Trident operator by using Helm.

What you'll need

In addition to the prerequisites listed above, to deploy Trident operator by using Helm, you need the following:

- Kubernetes 1.19 - 1.24
- Helm version 3

Steps

1. Add Trident's Helm repository:

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

2. Use the `helm install` command and specify a name for your deployment.
See the following example:

```
helm install <release-name> netapp-trident/trident-operator --version 22.4.0 --create-namespace <trident-namespace>
```



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

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

- `--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 `--set` command:

```
$ helm install <name> netapp-trident/trident-operator --version 22.7.0 --set tridentDebug=true
```

The `values.yaml` file, which is part of the Helm chart provides the list of keys and their default values.

`helm list` shows you details about the installation, such as name, namespace, chart, status, app version, revision number, and so on.

Deploy the Trident operator manually

Perform the steps listed to manually deploy the Trident operator.

Step 1: Qualify your Kubernetes cluster

The first thing you need to do is log in to the Linux host and verify that it is managing a *working*, [supported Kubernetes cluster](#) that you have the necessary privileges to.



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`.

To verify your Kubernetes version, run the following command:

```
kubectl version
```

To see if you have Kubernetes cluster administrator privileges, run the following command:

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

To verify if you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network, run the following command:

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

Step 2: Download and set up the operator



Beginning with 21.01, the Trident operator is cluster scoped. Using the Trident operator to install Trident requires creating the `TridentOrchestrator` Custom Resource Definition (CRD) and defining other resources. You should perform these steps to set up the operator before you can install Astra Trident.

1. Download and extract the latest version of the Trident installer bundle from [the Assets section on GitHub](#).

```
wget
https://github.com/NetApp/trident/releases/download/v22.04.0/trident-
installer-22.04.0.tar.gz
tar -xf trident-installer-22.04.0.tar.gz
cd trident-installer
```


2. Use the appropriate CRD manifest to create the `TridentOrchestrator` CRD. You then create a `TridentOrchestrator` Custom Resource later on to instantiate an installation by the operator.

Run the following command:

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

3. After the `TridentOrchestrator` CRD is created, create the following resources required for the operator deployment:

- A `ServiceAccount` for the operator
- A `ClusterRole` and `ClusterRoleBinding` to the `ServiceAccount`
- A dedicated `PodSecurityPolicy`
- The operator itself

The Trident installer contains manifests for defining these resources. By default, the operator is deployed in the `trident` namespace. If the `trident` namespace does not exist, use the following manifest to create one.

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

4. To deploy the operator in a namespace other than the default `trident` namespace, you should update the `serviceaccount.yaml`, `clusterrolebinding.yaml` and `operator.yaml` manifests and generate your `bundle.yaml`.

Run the following command to update the YAML manifests and generate your `bundle.yaml` using the `kustomization.yaml`:

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

Run the following command to create the resources and deploy the operator:

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

5. To verify the status of the operator after you have deployed, do the following:

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

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


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

NAME	READY	STATUS	RESTARTS
trident-operator-54cb664d-lnjxh	1/1	Running	0

3m

The operator deployment successfully creates a pod running on one of the worker nodes in your cluster.



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

Step 3: Create `TridentOrchestrator` and install Trident

You are now ready to install Astra Trident using the operator! This will require creating `TridentOrchestrator`. The Trident installer comes with example definitions for creating `TridentOrchestrator`. This kicks off an installation in the `trident` namespace.

```

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

$ kubectl describe torc trident
Name:          trident
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Spec:
  Debug:      true
  Namespace:  trident
Status:
  Current Installation Params:
    IPv6:          false
    Autosupport Hostname:
    Autosupport Image:      netapp/trident-autosupport:21.04
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:          true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:     30
    Kubelet Dir:    /var/lib/kubelet
    Log Format:     text
    Silence Autosupport:  false
    Trident Image:  netapp/trident:21.04.0
  Message:          Trident installed Namespace:
trident
  Status:           Installed
  Version:          v21.04.0
Events:
  Type Reason Age From Message ---- -
Installing 74s trident-operator.netapp.io Installing Trident Normal
Installed 67s trident-operator.netapp.io Trident installed

```

The Trident operator enables you to customize the manner in which Astra Trident is installed by using the attributes in the `TridentOrchestrator` spec. See [Customize your Trident deployment](#).

The `Status` of `TridentOrchestrator` indicates if the installation was successful and displays the version of Trident installed.

Status	Description
Installing	The operator is installing Astra Trident using this <code>TridentOrchestrator</code> CR.
Installed	Astra Trident has successfully installed.
Uninstalling	The operator is uninstalling Astra Trident, because <code>spec.uninstall=true</code> .
Uninstalled	Astra Trident is uninstalled.
Failed	The operator could not install, patch, update or uninstall Astra 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 <code>TridentOrchestrator</code> is not used. Another one already exists.

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, you should check the logs of the operator. See the [troubleshooting](#) section.

You can confirm if the Astra Trident installation completed by taking a look at the pods that have been created:

```
$ kubectl get pod -n trident
```

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

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

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

+-----+	
SERVER VERSION	CLIENT VERSION
+-----+	
21.04.0	21.04.0
+-----+	

Now you can go ahead and create a backend. See [post-deployment tasks](#).



For troubleshooting issues during deployment, see the [troubleshooting](#) section.

Customize Trident operator deployment

The Trident operator enables you to customize the manner in which Astra Trident is installed by using the attributes in the `TridentOrchestrator` spec.

See the following table for the list of attributes:

Parameter	Description	Default
<code>namespace</code>	Namespace to install Astra Trident in	"default"
<code>debug</code>	Enable debugging for Astra Trident	false
<code>IPv6</code>	Install Astra Trident over IPv6	false
<code>k8sTimeout</code>	Timeout for Kubernetes operations	30sec
<code>silenceAutosupport</code>	Don't send autosupport bundles to NetApp automatically	false
<code>enableNodePrep</code>	Manage worker node dependencies automatically (BETA)	false
<code>autosupportImage</code>	The container image for Autosupport Telemetry	"netapp/trident-autosupport:21.04.0"
<code>autosupportProxy</code>	The address/port of a proxy for sending Autosupport Telemetry	"http://proxy.example.com:8888"
<code>uninstall</code>	A flag used to uninstall Astra Trident	false
<code>logFormat</code>	Astra Trident logging format to be used [text,json]	"text"
<code>tridentImage</code>	Astra Trident image to install	"netapp/trident:21.04"
<code>imageRegistry</code>	Path to internal registry, of the format <registry FQDN>[:port] [/subpath]	"k8s.gcr.io/sig-storage (k8s 1.19+) or quay.io/k8scsi"
<code>kubeletDir</code>	Path to the kubelet directory on the host	"/var/lib/kubelet"
<code>wipeout</code>	A list of resources to delete to perform a complete removal of Astra Trident	
<code>imagePullSecrets</code>	Secrets to pull images from an internal registry	
<code>controllerPluginNodeSelector</code>	Additional node selectors for pods running the Trident Controller CSI Plugin. Follows same format as <code>pod.spec.nodeSelector</code> .	No default; optional

Parameter	Description	Default
controllerPluginTolerations	Overrides tolerations for pods running the Trident Controller CSI Plugin. Follows the same format as pod.spec.Tolerations.	No default; optional
nodePluginNodeSelector	Additional node selectors for pods running the Trident Node CSI Plugin. Follows same format as pod.spec.nodeSelector.	No default; optional
nodePluginTolerations	Overrides tolerations for pods running the Trident Node CSI Plugin. Follows the same format as pod.spec.Tolerations.	No default; optional



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



For more information on formatting pod parameters, see [Assigning Pods to Nodes](#).

You can use the attributes mentioned above when defining `TridentOrchestrator` to customize your installation. Here's an example:

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

Here is another example that shows how Trident can be deployed with node selectors:

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

If you are looking to customize the installation beyond what `TridentOrchestrator` arguments allow, you should consider using `tridentctl` to generate custom YAML manifests that you can modify as needed.

Deploy with `tridentctl`

You can deploy Astra Trident by using `tridentctl`.



If you have not already familiarized yourself with the [basic concepts](#), now is a great time to do that.



To customize your deployment, see [here](#).

What you'll need

To deploy Astra Trident, the following prerequisites should be met:

- You have full privileges to a supported Kubernetes cluster.
- You have access to a supported NetApp storage system.
- You have the capability to mount volumes from all of the Kubernetes worker nodes.
- You have 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.
- You have set the `KUBECONFIG` environment variable to point to your Kubernetes cluster configuration.
- You have enabled the [feature gates required by Astra Trident](#).
- If you are using Kubernetes with Docker Enterprise, [follow their steps to enable CLI access](#).

Got all that? Great! Let's get started.



For information about customizing your deployment, see [here](#).

Step 1: Qualify your Kubernetes cluster

The first thing you need to do is log into the Linux host and verify that it is managing a *working*, [supported Kubernetes cluster](#) that you have the necessary privileges to.



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

To check your Kubernetes version, run the following command:

```
kubectl version
```

To see if you have Kubernetes cluster administrator privileges, run the following command:

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

To verify if you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network, run the following command:

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

Identify your Kubernetes server version. You will use it when you install Astra Trident.

Step 2: Download and extract the installer



The Trident installer creates a Trident pod, configures the CRD objects that are used to maintain its state, and initializes the CSI sidecars that perform actions, such as provisioning and attaching volumes to the cluster hosts.

You can download and extract the latest version of the Trident installer bundle from [the Assets section on GitHub](#).

For example, if the latest version is 21.07.1:

```
wget https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
tar -xf trident-installer-21.07.1.tar.gz
cd trident-installer
```

Step 3: Install Astra Trident

Install Astra Trident in the desired namespace by executing the `tridentctl install` command.


```
$ ./tridentctl install -n trident
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-csi-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                  version=21.07.1
INFO Trident installation succeeded.
....
```

It will look like this when the installer is complete. Depending on the number of nodes in your Kubernetes cluster, you might observe more pods:

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

$ ./tridentctl -n trident version
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 21.07.1        | 21.07.1        |
+-----+-----+
```

If you see output similar to the above example, you've completed this step, but Astra Trident is not yet fully configured. Go ahead and continue to the next step. See [post-deployment tasks](#).

However, if the installer does not complete successfully or you don't see a **Running** trident-csi-
<generated id>, the platform was not installed.



For troubleshooting issues during deployment, see the [troubleshooting](#) section.

Customize tridentctl deployment

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]`.

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
```

What's next?

After you deploy Astra Trident, you can proceed with creating a backend, creating a storage class, provisioning a volume, and mounting the volume in a pod.

Step 1: Create a backend

You can now go ahead and create a backend that will be used by Astra Trident to provision volumes. To do this, create a `backend.json` file that contains the necessary parameters. Sample configuration files for different backend types can be found in the `sample-input` directory.

See [here](#) for more details about how to configure the file for your backend type.

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

```
./tridentctl -n trident create backend -f backend.json
```

NAME	STORAGE DRIVER	UUID
nas-backend	ontap-nas	98e19b74-aec7-4a3d-8dcf-128e5033b214

If the creation fails, something was wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
./tridentctl -n trident logs
```

After you address the problem, simply go back to the beginning of this step and try again. For more troubleshooting tips, see [the troubleshooting](#) section.

Step 2: Create a storage class

Kubernetes users provision volumes by using persistent volume claims (PVCs) that specify a [storage class](#) by name. The details are hidden from the users, but a storage class identifies the provisioner that is used for that class (in this case, Trident), and what that class means to the provisioner.

Create a storage class Kubernetes users will specify when they want a volume. The configuration of the class needs to model the backend that you created in the previous step, so that Astra Trident will use it to provision new volumes.

The simplest storage class to start with is one based on the `sample-input/storage-class-csi.yaml.template` file that comes with the installer, replacing `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

```

This is a Kubernetes object, so you use `kubectl` to create it in Kubernetes.

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

You should now see a **basic-csi** storage class in both Kubernetes and Astra Trident, and Astra 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"
        ]
      }
    }
  ]
}

```

Step 3: Provision your first volume

Now you are ready to dynamically provision your first volume. This is done by creating a Kubernetes [persistent volume claim](#) (PVC) object.

Create a PVC for a volume that uses the storage class that you just created.

See `sample-input/pvc-basic-csi.yaml` for an example. Make sure the storage class name matches the one that you created.

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

```
kubectl get pvc --watch
```

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

Step 4: Mount the volumes in a pod

Now let us mount the volume. We will launch an nginx pod that mounts the PV under /usr/share/nginx/html.

```
cat << EOF > task-pv-pod.yaml
```

```
kind: Pod
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: task-pv-pod
```

```
spec:
```

```
  volumes:
```

```
    - name: task-pv-storage
```

```
      persistentVolumeClaim:
```

```
        claimName: basic
```

```
  containers:
```

```
    - name: task-pv-container
```

```
      image: nginx
```

```
      ports:
```

```
        - containerPort: 80
```

```
          name: "http-server"
```

```
      volumeMounts:
```

```
        - mountPath: "/usr/share/nginx/html"
```

```
          name: task-pv-storage
```

```
EOF
```

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

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

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

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

At this point, the pod (application) no longer exists but the volume is still there. You can use it from another pod if you want to.

To delete the volume, delete the claim:

```
kubectl delete pvc basic
```

You can now do additional tasks, such as the following:

- [Configure additional backends.](#)
- [Create additional storage classes.](#)

Manage Astra Trident

Upgrade Astra Trident

Astra Trident follows a quarterly release cadence, delivering four major releases every calendar year. Each new release builds on top of the previous releases, providing new features and performance enhancements as well as bug fixes and improvements. You are encouraged to upgrade at least once a year to take advantage of the new features in Astra Trident.



Upgrading to a release that is five releases ahead will require you to perform a multistep upgrade.

Determine the version to upgrade to

- You can upgrade to the `YY.MM` release from the `YY-1.MM` release and any in-between releases. For example, you can perform a direct upgrade to 20.07 from 19.07 and later (including dot releases, such as 19.07.1).
- If you have an earlier release, you should perform a multistep upgrade. This requires you to first upgrade to the most recent release that fits your four-release window. For example, if you are running 18.07 and want to upgrade to the 20.07 release, then follow the multistep upgrade process as given below:
 - First upgrade from 18.07 to 19.07. See the documentation of the respective release to obtain specific instructions for upgrading.
 - Then upgrade from 19.07 to 20.07.



All upgrades for versions 19.04 and earlier require the migration of Astra Trident's metadata from its own `etcd` to CRD objects. Ensure that you check the documentation of the release to understand how the upgrade works.



When upgrading, it is important you provide `parameter.fsType` in `StorageClasses` used by Astra 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, see [Known Issues](#).

Which upgrade path should I choose?

You can upgrade by using one of the following paths:

- Using the Trident operator.
- Using `tridentctl`.



CSI Volume Snapshots is now a feature that is GA, beginning with Kubernetes 1.20. When upgrading Astra Trident, all previous alpha snapshot CRs and CRDs (Volume Snapshot Classes, Volume Snapshots and Volume Snapshot Contents) must be removed before the upgrade is performed. Refer to [this blog](#) to understand the steps involved in migrating alpha snapshots to the beta/GA spec.

You can use the Trident operator to upgrade if the following conditions are met:

- You are running CSI Trident (19.07 and later).
- You have a CRD-based Trident release (19.07 and later).
- You are **not** performing a customized install (using custom YAMLs).



Do not use the operator to upgrade Trident if you are using an `etcd`-based Trident release (19.04 or earlier).

If you do not want to use the operator or you have a customized install that cannot be supported by the operator, you can upgrade by using `tridentctl`. This is the preferred method of upgrades for Trident releases 19.04 and earlier.

Changes to the operator

The 21.01 release of Astra Trident introduces some key architectural changes to the operator, namely the following:

- The operator is now **cluster-scoped**. Previous instances of the Trident operator (versions 20.04 through 20.10) were **namespace-scoped**. An operator that is cluster-scoped is advantageous for the following reasons:
 - Resource accountability: The operator now manages resources associated with an Astra Trident installation at the cluster level. As part of installing Astra Trident, the operator creates and maintains several resources by using `ownerReferences`. Maintaining `ownerReferences` on cluster-scoped resources can throw up errors on certain Kubernetes distributors such as OpenShift. This is mitigated with a cluster-scoped operator. For auto-healing and patching Trident resources, this is an essential requirement.
 - Cleaning up during uninstallation: A complete removal of Astra Trident would require all associated resources to be deleted. A namespace-scoped operator might experience issues with the removal of cluster-scoped resources (such as the `clusterRole`, `ClusterRoleBinding` and `PodSecurityPolicy`) and lead to an incomplete clean-up. A cluster-scoped operator eliminates this issue. Users can completely uninstall Astra Trident and install afresh if needed.
- `TridentProvisioner` is now replaced with `TridentOrchestrator` as the Custom Resource used to install and manage Astra Trident. In addition, a new field is introduced to the `TridentOrchestrator` spec. Users can specify that the namespace Trident must be installed/updated from using the `spec.namespace` field. You can take a look at an example [here](#).

Find more information

- [Upgrade by using the Trident operator](#)
- [Upgrade by using `tridentctl`](#)

Upgrade with the operator

You can easily upgrade an existing Astra Trident installation using the operator.

What you'll need

To upgrade by using the operator, the following conditions should be met:

- You should have a CSI-based Astra Trident installation. To check if you are running CSI Trident, examine the pods in your Trident namespace. If they follow the `trident-csi-*` naming pattern, you are running CSI Trident.
- You should have a CRD-based Trident installation. This represents all releases from 19.07 and later. If you have a CSI-based installation, you most likely have a CRD-based installation.
- If you have uninstalled CSI Trident and the metadata from the installation persists, you can upgrade by using the operator.
- Only one Astra Trident installation should exist across all the namespaces in a given Kubernetes cluster.
- You should be using a Kubernetes cluster that runs [version 1.19 -1.24](#).
- If alpha snapshot CRDs are present, you should remove them with `tridentctl oblivate alpha-snapshot-crd`. This deletes the CRDs for the alpha snapshot spec. For existing snapshots that should be deleted/migrated, see [this blog](#).



When upgrading Trident by using the 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, see the [issue details on GitHub](#).

Upgrade a cluster-scoped operator installation

To upgrade from **Trident 21.01 and later**, here is the set of steps to be followed.

Steps

1. Delete the Trident operator that was used to install the current Astra Trident instance. For example, if you are upgrading from 21.01, run the following command:

```
kubectl delete -f 21.01/trident-installer/deploy/bundle.yaml -n trident
```

2. (Optional) If you want to modify the installation parameters, edit the `TridentOrchestrator` object that you created when installing Trident.
This can include changes, such as modifying the custom Trident image, private image registry to pull container images from, enabling debug logs, or specifying image pull secrets.
3. Install Astra Trident by using the `bundle.yaml` file that sets up the Trident operator for the new version. Run the following command:

```
kubectl create -f 21.10.0/trident-installer/deploy/bundle.yaml -n trident
```

As part of this step, the 21.10.0 Trident operator will identify an existing Astra Trident installation and upgrade it to the same version as the operator.

Upgrade a namespace-scoped operator installation

To upgrade from an instance of Astra Trident installed using the namespace-scoped operator (versions 20.07 through 20.10), here is the set of steps to be followed:

Steps

1. Verify the status of the existing Trident installation. To do this, check the **Status** of `TridentProvisioner`. The status should be `Installed`.

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



If status shows `Updating`, ensure you resolve it before proceeding. For a list of possible status values, see [here](#).

2. Create the `TridentOrchestrator` CRD by using the manifest provided with the Trident installer.

```
# Download the release required [21.01]
$ mkdir 21.07.1
$ cd 21.07.1
$ wget
https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
$ tar -xf trident-installer-21.07.1.tar.gz
$ cd trident-installer
$ kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. Delete the namespace-scoped operator by using its manifest. To complete this step, you require the `bundle.yaml` file used to deploy the namespace-scoped operator. You can obtain `bundle.yaml` from the [Trident repository](#). Make sure to use the appropriate branch.



You should make the necessary changes to the Trident install parameters (for example, changing the values for `tridentImage`, `autosupportImage`, private image repository, and providing `imagePullSecrets`) after deleting the namespace-scoped operator and before installing the cluster-scoped operator. For a complete list of parameters that can be updated, see the [list of parameters](#).

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

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

#Confirm the Trident operator was removed
$ kubectl get all -n trident
```

NAME	READY	STATUS	RESTARTS	AGE
pod/trident-csi-68d979fb85-dsrmn	6/6	Running	12	99d
pod/trident-csi-8jfhf	2/2	Running	6	105d
pod/trident-csi-jtnjz	2/2	Running	6	105d
pod/trident-csi-lcxvh	2/2	Running	8	105d

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

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

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

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

At this stage, the trident-operator-xxxxxxxxxx-xxxxx pod is deleted.

4. (Optional) If the install parameters need to be modified, update the `TridentProvisioner` spec. These could be changes such as modifying the private image registry to pull container images from, enabling debug logs, or specifying image pull secrets.

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

5. Install the cluster-scoped operator.



Installing the cluster-scoped operator initiates the migration of `TridentProvisioner` objects to `TridentOrchestrator` objects, deletes `TridentProvisioner` objects and the `tridentprovisioner` CRD, and upgrades Astra Trident to the version of the cluster-scoped operator being used. In the example that follows, Trident is upgraded to 21.07.1.



Upgrading Astra Trident by using the cluster-scoped operator results in the migration of `tridentProvisioner` to a `tridentOrchestrator` object with the same name. This is automatically handled by the operator. The upgrade will also have Astra Trident installed in the same namespace as before.

```
#Ensure you are in the correct directory
$ pwd
$ /root/21.07.1/trident-installer

#Install the cluster-scoped operator in the **same namespace**
$ kubectl create -f deploy/bundle.yaml
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

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

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

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

#Confirm Trident has been updated to the desired version
$ kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v21.07.1
```

Upgrade a Helm-based operator installation

Perform the following steps to upgrade a Helm-based operator installation.

Steps

1. Download the latest Astra Trident release.
2. Use the `helm upgrade` command. See the following example:

```
$ helm upgrade <name> trident-operator-21.07.1.tgz
```

where `trident-operator-21.07.1.tgz` reflects the version that you want to upgrade to.

3. Run `helm list` to verify that the chart and app version have both been upgraded.



To pass configuration data during the upgrade, use `--set`.

For example, to change the default value of `tridentDebug`, run the following command:

```
$ helm upgrade <name> trident-operator-21.07.1-custom.tgz --set  
tridentDebug=true
```

If you run `$ tridentctl logs`, you can see the debug messages.



If you set any non-default options during the initial installation, ensure that the options are included in the upgrade command, or else, the values will be reset to their defaults.

Upgrade from a non-operator installation

If you have a CSI Trident instance that meets the prerequisites listed above, you can upgrade to the latest release of the Trident operator.

Steps

1. Download the latest Astra Trident release.

```
# Download the release required [21.07.1]  
$ mkdir 21.07.1  
$ cd 21.07.1  
$ wget  
https://github.com/NetApp/trident/releases/download/v21.07.1/trident-  
installer-21.07.1.tar.gz  
$ tar -xf trident-installer-21.07.1.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 operator.

```
#Install the cluster-scoped operator in the **same namespace**
$ kubectl create -f deploy/bundle.yaml
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
```

NAME	READY	STATUS	RESTARTS	AGE
trident-csi-79df798bdc-m79dc	6/6	Running	0	150d
trident-csi-xrst8	2/2	Running	0	150d
trident-operator-5574dbbc68-nthjv	1/1	Running	0	1m30s

4. Create a TridentOrchestrator CR for installing Astra Trident.

```
#Create a tridentOrchestrator to initiate a Trident install
$ cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

$ kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
```

NAME	READY	STATUS	RESTARTS	AGE
trident-csi-79df798bdc-m79dc	6/6	Running	0	1m
trident-csi-xrst8	2/2	Running	0	1m
trident-operator-5574dbbc68-nthjv	1/1	Running	0	5m41s

```
#Confirm Trident was upgraded to the desired version
$ kubectl describe torc trident | grep Message -A 3
Message:          Trident installed
Namespace:        trident
Status:           Installed
Version:          v21.07.1
```

The existing backends and PVCs are automatically available.

Upgrade with `tridentctl`

You can easily upgrade an existing Astra Trident installation by using `tridentctl`.

Considerations

When upgrading to the latest release of Astra Trident, consider the following:

- Starting with Trident 20.01, only the beta release of [volume snapshots](#) is supported. Kubernetes administrators should take care to safely back up or convert the alpha snapshot objects to beta to retain the legacy alpha snapshots.
- The beta release of volume snapshots introduces a modified set of CRDs and a snapshot controller, both of which should be set up before installing Astra Trident.



[This blog](#) discusses the steps involved in migrating alpha volume snapshots to the beta format.

About this task

Uninstalling and reinstalling Astra Trident acts as an upgrade. When you uninstall Trident, the Persistent Volume Claim (PVC) and Persistent Volume (PV) used by the Astra Trident deployment are not deleted. PVs that have already been provisioned will remain available while Astra Trident is offline, and Astra Trident will provision volumes for any PVCs that are created in the interim once it is back online.



When upgrading Astra Trident, do not interrupt the upgrade process. Ensure that the installer runs to completion.

Next steps after upgrade

To make use of the rich set of features that are available in newer Trident releases (such as, On-Demand Volume Snapshots), you can upgrade the volumes by using the `tridentctl upgrade` command.

If there are legacy volumes, you should upgrade them from a NFS/iSCSI type to the CSI type to be able to use the complete set of new features in Astra Trident. A legacy PV that has been provisioned by Trident supports the traditional set of features.

Consider the following when deciding to upgrade volumes to the CSI type:

- You might not need to upgrade all the volumes. Previously created volumes will continue to be accessible and function normally.
- A PV can be mounted as part of a deployment/StatefulSet when upgrading. It is not required to bring down the deployment/StatefulSet.
- You **cannot** attach a PV to a standalone pod when upgrading. You should shut down the pod before upgrading the volume.
- You can upgrade only a volume that is bound to a PVC. Volumes that are not bound to PVCs should be removed and imported before upgrading.

Volume upgrade example

Here is an example that shows how a volume upgrade is performed.

1. Run `kubectl get pv` to list the PVs.

```
$ kubectl get pv
```

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

There are currently four PVs that have been created by Trident 20.07, using the `netapp.io/trident` provisioner.

2. Run `kubectl describe pv` to get the details of the PV.

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

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

The PV was created by using the `netapp.io/trident` provisioner and is of the type NFS. To support all the new features provided by Astra Trident, this PV should be upgraded to the CSI type.

3. Run the `tridentctl upgrade volume <name-of-trident-volume>` command to upgrade a legacy

```
$ ./tridentctl get volumes -n trident
```

NAME	SIZE	STORAGE CLASS	PROTOCOL	BACKEND UUID	STATE	MANAGED
default-pvc-2-a8486	1.0 GiB	standard	file	c5a6f6a4-b052-423b-80d4-8fb491a14a22	online	true
default-pvc-3-a849e	1.0 GiB	standard	file	c5a6f6a4-b052-423b-80d4-8fb491a14a22	online	true
default-pvc-1-a8475	1.0 GiB	standard	file	c5a6f6a4-b052-423b-80d4-8fb491a14a22	online	true
default-pvc-4-a84de	1.0 GiB	standard	file	c5a6f6a4-b052-423b-80d4-8fb491a14a22	online	true


```
$ ./tridentctl upgrade volume default-pvc-2-a8486 -n trident
```

NAME	SIZE	STORAGE CLASS	PROTOCOL	BACKEND UUID	STATE	MANAGED
default-pvc-2-a8486	1.0 GiB	standard	file	c5a6f6a4-b052-423b-80d4-8fb491a14a22	online	true

4. Run a `kubectl describe pv` to verify that the volume is a CSI volume.

```

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

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

```

In this manner, you can upgrade volumes of the NFS/iSCSI type that were created by Astra Trident to the CSI type, on a per-volume basis.

Uninstall Astra Trident

Depending on how Astra Trident is installed, there are multiple options to uninstall it.

Uninstall by using Helm

If you installed Astra Trident by using Helm, you can uninstall it by using `helm uninstall`.

```
#List the Helm release corresponding to the Astra Trident install.
$ helm ls -n trident
NAME                NAMESPACE      REVISION      UPDATED
STATUS              CHART           APP VERSION
trident             trident         1             2021-04-20
00:26:42.417764794 +0000 UTC deployed    trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
$ helm uninstall trident -n trident
release "trident" uninstalled
```

Uninstall by using the Trident operator

If you installed Astra Trident by using the operator, you can uninstall it by doing one of the following:

- **Edit `TridentOrchestrator` to set the uninstall flag:** You can edit `TridentOrchestrator` and set `spec.uninstall=true`. Edit the `TridentOrchestrator` CR and set the `uninstall` flag as shown below:

```
$ 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.

- **Delete `TridentOrchestrator`:** By removing the `TridentOrchestrator` CR that was used to deploy Astra Trident, you instruct the operator to uninstall Trident. The operator processes the removal of `TridentOrchestrator` and proceeds to remove the Astra Trident deployment and daemonset, deleting the Trident pods it had created as part of the installation. To completely remove Astra Trident (including the CRDs it creates) and effectively wipe the slate clean, you can edit `TridentOrchestrator` to pass the `wipeout` option. See the following example:

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

This uninstalls Astra Trident completely and clears all metadata related to the backends and volumes it manages. Subsequent installations are treated as fresh installations.



You should only consider wiping out the CRDs when performing a complete uninstallation. This cannot be undone. **Do not wipe out the CRDs unless you are looking to start over and create a fresh Astra Trident installation.**

Uninstall by using `tridentctl`

Run the `uninstall` command in `tridentctl` as follows to removes all of the resources associated with Astra Trident except for the CRDs and related objects, thereby making it easy to run the installer again to update to a more recent version.

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

To perform a complete removal of Astra Trident, you should remove the finalizers for the CRDs created by Astra Trident and delete the CRDs.

Downgrade Astra Trident

Learn about the steps involved in downgrading to an earlier version of Astra Trident.

You might consider downgrading for various reasons, such as the following:

- Contingency planning
- Immediate fix for bugs observed as a result of an upgrade
- Dependency issues, unsuccessful and incomplete upgrades

When to downgrade

You should consider a downgrade when moving to a Astra Trident release that uses CRDs. Because Astra Trident now uses CRDs for maintaining state, all storage entities created (backends, storage classes, PV, and volume snapshots) have associated CRD objects instead of data written into the `trident` PV (used by the earlier installed version of Astra Trident). Newly created PVs, backends, and storage classes are all maintained as CRD objects. If you need to downgrade, this should only be attempted for a version of Astra Trident that runs using CRDs (19.07 and later). This is to ensure that all the operations performed on the current Astra Trident release are visible after the downgrade occurs.

When not to downgrade

You should not downgrade to a release of Trident that uses `etcd` to maintain state (19.04 and earlier). All operations performed with the current Astra Trident release are not reflected after the downgrade. Newly created PVs are not usable when moving back to an earlier version. Changes made to objects such as backends, PVs, storage classes, and volume snapshots (created/updated/deleted) are not visible to Astra Trident when moving back to an earlier version. Going back to an earlier version does not disrupt access for PVs that were already created by using the older release, unless they have been upgraded.

Downgrade process when Astra Trident is installed by using the operator

For installations done using the Trident Operator, the downgrade process is different and does not require the use of `tridentctl`.

For installations done using the Trident operator, Astra Trident can be downgraded to either of the following:

- A version that is installed using the namespace-scoped operator (20.07 - 20.10).
- A version that is installed using the cluster-scoped operator (21.01 and later).

Downgrade to cluster-scoped operator

To downgrade Astra Trident to a release that uses the cluster-scoped operator, follow the steps mentioned below.

Steps

1. **Uninstall Astra Trident.** Do not wipeout the CRDs unless you want to completely remove an existing installation.
2. Delete the cluster-scoped operator. To do this, you will need the manifest used to deploy the operator. You can obtain it from the [Trident GitHub repo](#). Make sure you switch to the required branch.
3. Continue downgrading by installing the desired version of Astra Trident. Follow the documentation for the desired release.

Downgrade to namespace-scoped operator

This section summarizes the steps involved in downgrading to an Astra Trident release that falls in the range 20.07 through 20.10, which will be installed using the namespace-scoped operator.

Steps

1. **Uninstall Astra Trident.** Do not wipeout the CRDs unless you want to completely remove an existing installation.

Make sure the `tridentorchestrator` is deleted.

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

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

2. Delete the cluster-scoped operator. To do this, you will need the manifest used to deploy the operator. You can obtain it here from the [Trident GitHub repo](#). Make sure you switch to the required branch.
3. Delete the `tridentorchestrator` CRD.

```
#Check to see if ``tridentorchestrators.trident.netapp.io`` CRD is
present and delete it.
$ kubectl get crd tridentorchestrators.trident.netapp.io
NAME                                     CREATED AT
tridentorchestrators.trident.netapp.io  2021-01-21T21:11:37Z
$ kubectl delete crd tridentorchestrators.trident.netapp.io
customresourcedefinition.apiextensions.k8s.io
"tridentorchestrators.trident.netapp.io" deleted
```

Astra Trident has been uninstalled.

4. Continue downgrading by installing the desired version. Follow the documentation for the desired release.

Downgrade by using Helm

To downgrade, use the `helm rollback` command. See the following example:

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

Downgrade process when Astra Trident is installed by using `tridentctl`

If you installed Astra Trident by using `tridentctl`, the downgrade process involves the following steps. This sequence walks you through the downgrade process to move from Astra Trident 21.07 to 20.07.



Before beginning the downgrade, you should take a snapshot of your Kubernetes cluster's `etcd`. This enables you to back up the current state of Astra Trident's CRDs.

Steps

1. Make sure that Trident is installed by using `tridentctl`. If you are unsure about how Astra Trident is installed, run this simple test:
 - a. List the pods present in the Trident namespace.
 - b. Identify the version of Astra Trident running in your cluster. You can either use `tridentctl` or take a look at the image used in the Trident pods.
 - c. If you **do not see** a `tridentOrchestrator`, (or) a `tridentprovisioner`, (or) a pod named `trident-operator-xxxxxxxxxx-xxxxx`, Astra Trident **is installed** with `tridentctl`.
2. Uninstall Astra Trident with the existing `tridentctl` binary. In this case, you will uninstall with the 21.07 binary.


```

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

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

```

3. After this is complete, obtain the Trident binary for the desired version (in this example, 20.07), and use it to install Astra Trident. You can generate custom YAMLs for a [customized installation](#) if needed.

```

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

```

The downgrade process is complete.

Use Astra Trident

Configure backends

A backend defines the relationship between Astra Trident and a storage system. It tells Astra Trident how to communicate with that storage system and how Astra Trident should provision volumes from it. Astra Trident will automatically offer up storage pools from backends that together match the requirements defined by a storage class. Learn more about configuring the backend based on the type of storage system you have.

- [Configure an Azure NetApp Files backend](#)
- [Configure a Cloud Volumes Service for Google Cloud Platform 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 Astra Trident with Amazon FSx for NetApp ONTAP](#)

Configure an Azure NetApp Files backend

Learn about how to configure Azure NetApp Files (ANF) as the backend for your Astra Trident installation using the sample configurations provided.



The Azure NetApp Files service does not support volumes less than 100 GB. Astra Trident automatically creates 100-GB volumes if a smaller volume is requested.

What you'll need

To configure and use an [Azure NetApp Files](#) backend, you need the following:

- `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 the `Owner` or `Contributor` role that is predefined by Azure.



To learn more about Azure built-in roles, see the [Azure documentation](#).

- 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.
- If you are using Azure NetApp Files for the first time or in a new location, some initial configuration is required. See the [quickstart guide](#).

About this task

Based on the backend configuration (subnet, virtual network, service level, and location), Trident creates ANF volumes on capacity pools that are available in the requested location and match the requested service level and subnet.



NOTE: Astra Trident does not support Manual QoS capacity pools.

Backend configuration options

See the following table for the backend configuration options:

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

Parameter	Description	Default
debugTraceFlags	Debug flags to use when troubleshooting. Example, <code>\{"api": false, "method": true, "discovery": true\}</code> . Do not use this unless you are troubleshooting and require a detailed log dump.	null



If you encounter a “No capacity pools found” error when attempting to create a PVC, it is likely your app registration doesn’t have the required permissions and resources (subnet, virtual network, capacity pool) associated. Astra Trident will log the Azure resources it discovered when the backend is created when debug is enabled. Be sure to check if an appropriate role is being used.



If you want to mount the volumes by using NFS version 4.1, you can include `nfsvers=4` in the comma-delimited mount options list to choose NFS v4.1. Any mount options set in a storage class override the mount options set in a backend configuration file.



The [Network Features](#) functionality is not generally available in all regions and may have to be enabled in a subscription. Specifying the `networkFeatures` config option when the functionality is not enabled will cause volume provisioning in Trident to fail.

The values for `resourceGroups`, `netappAccounts`, `capacityPools`, `virtualNetwork`, and `subnet` may be specified using short or fully-qualified names. Short names may match multiple resources with the same name, so using fully-qualified names is recommended in most situations. The `resourceGroups`, `netappAccounts`, and `capacityPools` values are filters which restrict the set of discovered resources to those available to this storage backend and may be specified in any combination. The fully-qualified names are of the following 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>

You can control how each volume is provisioned by default by specifying the following options in a special section of the configuration file. See the configuration examples below.

Parameter	Description	Default
exportRule	The export rule(s) for new volumes	"0.0.0.0/0"
snapshotDir	Controls visibility of the <code>.snapshot</code> directory	"false"
size	The default size of new volumes	"100G"

Parameter	Description	Default
unixPermissions	The unix permissions of new volumes (4 octal digits)	"" (preview feature, requires whitelisting in subscription)

The `exportRule` value must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation.



For all the volumes created on an ANF backend, Astra Trident copies all the labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all the volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Example 1: Minimal configuration

This is the absolute minimum backend configuration. With this configuration, Astra Trident discovers all of your NetApp accounts, capacity pools, and subnets delegated to ANF in the configured location, and places new volumes on one of those pools and subnets randomly.

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

```
{
  "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"
}
```

Example 2: Specific service level configuration with capacity pool filters

This backend configuration places volumes in Azure's `eastus` location in an `Ultra` capacity pool. Astra Trident automatically discovers all of the subnets delegated to ANF 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"
  ],
}
```

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

```

Example 4: Virtual storage 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.

```

{
  "version": 1,
  "storageDriverName": "azure-netapp-files",
  "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
  "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
  "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
  "clientSecret": "SECRET",
  "location": "eastus",
  "resourceGroups": ["application-group-1"],
  "networkFeatures": "Basic",
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
  "labels": {
    "cloud": "azure"
  },
  "location": "eastus",

  "storage": [
    {
      "labels": {
        "performance": "gold"
      },
      "serviceLevel": "Ultra",
      "capacityPools": ["ultra-1", "ultra-2"],
      "networkFeatures": "Standard"
    },
    {
      "labels": {
        "performance": "silver"
      },
      "serviceLevel": "Premium",
      "capacityPools": ["premium-1"]
    },
    {
      "labels": {
        "performance": "bronze"
      },
      "serviceLevel": "Standard",
      "capacityPools": ["standard-1", "standard-2"]
    }
  ]
}

```

The following StorageClass definitions refer to the storage pools above. By using the `parameters.selector` field, 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

```

What's next?

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

```
tridentctl create backend -f <backend-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.

Configure a CVS for GCP backend

Learn about how to configure NetApp Cloud Volumes Service (CVS) for Google Cloud Platform (GCP) as the backend for your Astra Trident installation using the sample configurations provided.



NetApp Cloud Volumes Service for Google Cloud does not support CVS-Performance volumes less than 100 GiB in size, or CVS volumes less than 300 GiB in size. Astra Trident automatically creates volumes of the minimum size if a the volume requested is smaller than the minimum size.

What you'll need

To configure and use the [Cloud Volumes Service for Google Cloud](#) backend, you need the following:

- A Google Cloud account configured with NetApp CVS
- Project number of your Google Cloud account
- Google Cloud service account with the `netappcloudvolumes.admin` role
- API key file for your CVS service account

Astra Trident now includes support for smaller volumes with the default [CVS service type on GCP](#). For backends created with `storageClass=software`, volumes will now have a minimum provisioning size of 300 GiB. CVS currently provides this feature under Controlled Availability and does not provide technical support. Users must sign up for access to sub-1TiB volumes [here](#). NetApp recommends customers consume sub-1TiB volumes for **non-production** workloads.



When deploying backends using the default CVS service type (`storageClass=software`), users must obtain access to the sub-1TiB volumes feature on GCP for the Project Number(s) and Project ID(s) in question. This is necessary for Astra Trident to provision sub-1TiB volumes. If not, volume creations will fail for PVCs that are lesser than 600 GiB. Obtain access to sub-1TiB volumes using [this form](#).

Volumes created by Astra Trident for the default CVS service level will be provisioned as follows:

- PVCs that are smaller than 300 GiB will result in Astra Trident creating a 300 GiB CVS volume.
- PVCs that are between 300 GiB to 600 GiB will result in Astra Trident creating a CVS volume of the requested size.
- PVCs that are between 600 GiB and 1 TiB will result in Astra Trident creating a 1TiB CVS volume.
- PVCs that are greater than 1 TiB will result in Astra Trident creating a CVS volume of the requested size.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
<code>version</code>		Always 1
<code>storageDriverName</code>	Name of the storage driver	"gcp-cvs"
<code>backendName</code>	Custom name or the storage backend	Driver name + "_" + part of API key
<code>storageClass</code>	Type of storage. Choose from <code>hardware</code> (performance optimized) or <code>software</code> (CVS service type)	

Parameter	Description	Default
<code>projectNumber</code>	Google Cloud account project number. The value is found on the Google Cloud portal's Home page.	
<code>apiRegion</code>	CVS account region. It is the region where the backend will provision the volumes.	
<code>apiKey</code>	API key for the Google Cloud service account with the <code>netappcloudvolumes.admin</code> role. It includes the JSON-formatted contents of a Google Cloud service account's private key file (copied verbatim into the backend configuration file).	
<code>proxyURL</code>	Proxy URL if proxy server required to connect to CVS Account. The proxy server can either be an HTTP proxy or an HTTPS proxy. For an HTTPS proxy, certificate validation is skipped to allow the usage of self-signed certificates in the proxy server. Proxy servers with authentication enabled are not supported.	
<code>nfsMountOptions</code>	Fine-grained control of NFS mount options.	"nfsvers=3"
<code>limitVolumeSize</code>	Fail provisioning if the requested volume size is above this value	"" (not enforced by default)
<code>serviceLevel</code>	The CVS service level for new volumes. The values are "standard", "premium", and "extreme".	"standard"
<code>network</code>	GCP network used for CVS volumes	"default"
<code>debugTraceFlags</code>	Debug flags to use when troubleshooting. Example, <code>\{"api":false, "method":true\}</code> . Do not use this unless you are troubleshooting and require a detailed log dump.	null

If using a shared VPC network, both `projectNumber` and `hostProjectNumber` must be specified. In that case, `projectNumber` is the service project, and `hostProjectNumber` is the host project.

The `apiRegion` represents the GCP region where Astra Trident creates CVS volumes. When creating cross-region Kubernetes clusters, CVS volumes created in an `apiRegion` can be used in workloads scheduled on nodes across multiple GCP regions. Be aware that cross-region traffic incurs an additional cost.

- To enable cross-region access, your StorageClass definition for `allowedTopologies` must include all regions. For example:

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

- `storageClass` is an optional parameter that you can use to select the desired [CVS service type](#). You can choose from the base CVS service type (`storageClass=software`) or the CVS-Performance service type (`storageClass=hardware`), which Trident uses by default. Make sure you specify an `apiRegion` that provides the respective CVS `storageClass` in your backend definition.

Astra Trident's integration with the base CVS service type on Google Cloud is a **beta feature**, not meant for production workloads. Trident is **fully supported** with the CVS-Performance service type and uses it by default.

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

You can control how each volume is provisioned by default by specifying the following options in a special section of the configuration file. See the configuration examples below.

Parameter	Description	Default
<code>exportRule</code>	The export rule(s) for new volumes	"0.0.0.0/0"
<code>snapshotDir</code>	Access to the <code>.snapshot</code> directory	"false"
<code>snapshotReserve</code>	Percentage of volume reserved for snapshots	"" (accept CVS default of 0)
<code>size</code>	The size of new volumes	"100Gi"

The `exportRule` value must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation.

For all the volumes created on a CVS Google Cloud backend, Trident copies all the labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all the volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Example 1: Minimal configuration

This is the absolute minimum backend configuration.

```
{
  "version": 1,
```

```

"storageDriverName": "gcp-cvs",
"projectNumber": "012345678901",
"apiRegion": "us-west2",
"apiKey": {
  "type": "service_account",
  "project_id": "my-gcp-project",
  "private_key_id": "1234567890123456789012345678901234567890",
  "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisI
sAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSa
PIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZN
chRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzll
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4K
ws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5oj
Y9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznH
czZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrH
isIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbO
guSaPIKeyAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIK
eYAZNchRAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNch
RAGzllzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzll
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK
3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
  "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
  "client_id": "123456789012345678901",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
}
}

```

This example shows a backend definition that uses the base CVS service type, which is meant for general-purpose workloads and provides light/moderate performance, coupled with high zonal availability.

67

```

"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
    }
}

```

Example 3: Single service level configuration

This example shows a backend file that applies the same aspects to all Astra Trident-created storage in the Google Cloud us-west2 region. This example also shows the usage of `proxyURL` in the backend configuration file.

```

{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "apiRegion": "us-west2",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisI
sAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSa
PIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZN
chRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzll
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHi
sIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOgu
SaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyA
ZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz
llZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtr
HisIsAbOguSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbO
guSaPIKeyAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKe
yAZNchRAGzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRA
GzllZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE

```

```

KEY-----\n",
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
  },
  "proxyURL": "http://proxy-server-hostname/",
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
  "limitVolumeSize": "10Ti",
  "serviceLevel": "premium",
  "defaults": {
    "snapshotDir": "true",
    "snapshotReserve": "5",
    "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
    "size": "5Ti"
  }
}

```

Example 4: Virtual storage pool configuration

This example shows the backend definition file configured with virtual storage pools along with StorageClasses that refer back to them.

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the snapshotReserve at 5% and the exportRule to 0.0.0.0/0. The virtual storage pools are defined in the storage section. In this example, each individual storage pool sets its own serviceLevel, and some pools overwrite the default values.

```

{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "apiRegion": "us-west2",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI

```



```
sAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSa
PIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZN
chRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1z
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHi
sIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOgu
SaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyA
ZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz
1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrth
HisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbO
guSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKe
yAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRA
Gz1zZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrthHisIsAbOguSaPIKeyAZNchRAGz1zZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
```

```
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
  },
  "nfsMountOptions": "vers=3,proto=tcp,timeo=600",

  "defaults": {
    "snapshotReserve": "5",
    "exportRule": "0.0.0.0/0"
  },

  "labels": {
    "cloud": "gcp"
  },
  "region": "us-west2",

  "storage": [
```

```

{
  "labels": {
    "performance": "extreme",
    "protection": "extra"
  },
  "serviceLevel": "extreme",
  "defaults": {
    "snapshotDir": "true",
    "snapshotReserve": "10",
    "exportRule": "10.0.0.0/24"
  }
},
{
  "labels": {
    "performance": "extreme",
    "protection": "standard"
  },
  "serviceLevel": "extreme"
},
{
  "labels": {
    "performance": "premium",
    "protection": "extra"
  },
  "serviceLevel": "premium",
  "defaults": {
    "snapshotDir": "true",
    "snapshotReserve": "10"
  }
},
{
  "labels": {
    "performance": "premium",
    "protection": "standard"
  },
  "serviceLevel": "premium"
},
{
  "labels": {
    "performance": "standard"
  },
  "serviceLevel": "standard"
}
]

```

```
}
```

The following StorageClass definitions refer to the storage pools above. By using the `parameters.selector` field, 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.

The first StorageClass (`cvs-extreme-extra-protection`) maps to the first virtual storage pool. This is the only pool offering extreme performance with a snapshot reserve of 10%. The last StorageClass (`cvs-extra-protection`) calls out any storage pool which provides a snapshot reserve of 10%. Astra Trident decides which virtual storage pool is selected and ensures that the snapshot reserve requirement is met.

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

```

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

```

What's next?

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

```
tridentctl create backend -f <backend-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.

Configure a NetApp HCI or SolidFire backend

Learn about how to create and use an Element backend with your Astra Trident installation.

What you'll need

- 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. See [worker node preparation information](#).

What you need to know

The `solidfire-san` storage driver supports both volume modes: `file` and `block`. For the `Filesystem` volumeMode, Astra 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	No Filesystem. Raw block device.
solidfire-san	iSCSI	Block	RWO,ROX,RWX	No Filesystem. Raw block device.
solidfire-san	iSCSI	Filesystem	RWO,ROX	xfs, ext3, ext4
solidfire-san	iSCSI	Filesystem	RWO,ROX	xfs, ext3, ext4



Astra Trident uses CHAP when functioning as an enhanced CSI Provisioner. If you're using CHAP (which is the default for CSI), no further preparation is required. It is recommended to explicitly set the `UseCHAP` option to use CHAP with non-CSI Trident. Otherwise, see [here](#).



Volume access groups are only supported by the conventional, non-CSI framework for Astra Trident. When configured to work in CSI mode, Astra Trident uses CHAP.

If neither `AccessGroups` or `UseCHAP` are set, one of the following rules applies:

- If the default `trident` access group is detected, access groups are used.
- If no access group is detected and Kubernetes version is 1.7 or later, then CHAP is used.

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)	

Parameter	Description	Default
InitiatorIFace	Restrict iSCSI traffic to a specific host interface	"default"
UseCHAP	Use CHAP to authenticate iSCSI	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.



For all volumes created, Astra Trident will copy all labels present on a storage pool to the backing storage LUN at the time it is provisioned. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

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 storage pools

This example shows the backend definition file configured with virtual storage pools along with StorageClasses that refer back to them.

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the `type` at Silver. The virtual storage pools are defined in the `storage` section. In this example, some of the storage pool sets their own type, and some pools overwrite 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 storage 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 storage 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. Astra Trident will decide which virtual storage pool is selected and will ensure 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](#)

Configure a backend with ONTAP or Cloud Volumes ONTAP SAN drivers

Learn about configuring an ONTAP backend with ONTAP and Cloud Volumes ONTAP SAN drivers.

- [Preparation](#)
- [Configuration and examples](#)

User permissions

Astra 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, Astra 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 Astra 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.

Preparation

Learn about how to prepare to configure an ONTAP backend with ONTAP SAN drivers. For all ONTAP backends, Astra Trident requires at least one aggregate assigned to the SVM.

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. See [here](#) for more details.

Authentication

Astra 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:** Astra 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

Astra 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 Astra Trident releases. A custom security login role can be created and used with Astra Trident, but is not recommended.

A sample backend definition will look like this:

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

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/updating 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
```

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",
  "dataLIF": "1.2.3.8",
  "svm": "vserver_test",
  "clientCertificate": "Faaaakkkkeeee...Vaaalllluuuuueeee",
  "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",
  "dataLIF": "1.2.3.8",
  "svm": "vserver_test",
  "username": "vsadmin",
  "password": "secret",
  "storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
$ tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident

+-----+-----+-----+
+-----+-----+
|      NAME      | STORAGE DRIVER |          UUID          |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |          9 |
+-----+-----+-----+
+-----+-----+
```



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 Astra Trident can communicate with the ONTAP backend and handle future volume operations.

Specify igroups

Astra Trident uses igroups to control access to the volumes (LUNs) that it provisions. Administrators have two options when it comes to specifying igroups for backends:

- Astra Trident can automatically create and manage an igroup per backend. If `igroupName` is not included in the backend definition, Astra Trident creates an igroup named `trident-<backend-UUID>` on the SVM. This will ensure each backend has a dedicated igroup and handle the automated addition/deletion of Kubernetes node IQNs.
- Alternatively, pre-created igroups can also be provided in a backend definition. This can be done using the `igroupName` config parameter. Astra Trident will add/delete Kubernetes node IQNs to the pre-existing

igroup.

For backends that have `igroupName` defined, the `igroupName` can be deleted with a `tridentctl backend update` to have Astra Trident auto-handle igroups. This will not disrupt access to volumes that are already attached to workloads. Future connections will be handled using the igroup Astra Trident created.



Dedicating an igroup for each unique instance of Astra Trident is a best practice that is beneficial for the Kubernetes admin as well as the storage admin. CSI Trident automates the addition and removal of cluster node IQNs to the igroup, greatly simplifying its management. When using the same SVM across Kubernetes environments (and Astra Trident installations), using a dedicated igroup ensures that changes made to one Kubernetes cluster don't influence igroups associated with another. In addition, it is also important to ensure each node in the Kubernetes cluster has a unique IQN. As mentioned above, Astra Trident automatically handles the addition and removal of IQNs. Reusing IQNs across hosts can lead to undesirable scenarios where hosts get mistaken for one another and access to LUNs is denied.

If Astra Trident is configured to function as a CSI Provisioner, Kubernetes node IQNs are automatically added to/removed from the igroup. When nodes are added to a Kubernetes cluster, `trident-csi` DaemonSet deploys a pod (`trident-csi-xxxxx`) on the newly added nodes and registers the new nodes it can attach volumes to. Node IQNs are also added to the backend's igroup. A similar set of steps handle the removal of IQNs when node(s) are cordoned, drained, and deleted from Kubernetes.

If Astra Trident does not run as a CSI Provisioner, the igroup must be manually updated to contain the iSCSI IQNs from every worker node in the Kubernetes cluster. IQNs of nodes that join the Kubernetes cluster will need to be added to the igroup. Similarly, IQNs of nodes that are removed from the Kubernetes cluster must be removed from the igroup.

Authenticate connections with bidirectional CHAP

Astra 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`, Astra 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": "FaKePaSsWoRd",
  "igroupName": "trident",
  "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 Astra 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, Astra 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, Astra 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).
- Managing the addition of initiators to the `igroupName` given in the backend. If unspecified, this defaults to `trident`.

After the backend is created, Astra Trident creates a corresponding `tridentbackend` CRD and stores the CHAP secrets and usernames as Kubernetes secrets. All PVs that are created by Astra 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 through the CLI/ONTAP UI as Astra 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": "FaKePaSsWoRd",
  "igroupName": "trident",
  "chapInitiatorSecret": "cl9qxUpDaTeD",
  "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
}

$ ./tridentctl update backend ontap_san_chap -f backend-san.json -n
trident
+-----+-----+-----+
+-----+-----+
| NAME | STORAGE DRIVER | UUID |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| ontap_san_chap | ontap-san | aa458f3b-ad2d-4378-8a33-1a472ffbe5c |
online | 7 |
+-----+-----+-----+
+-----+-----+
```

Existing connections will remain unaffected; they will continue to remain active if the credentials are updated by Astra Trident on the SVM. New connections will use the updated credentials and existing connections continue to remain active. Disconnecting and reconnecting old PVs will result in them using the updated credentials.

Configuration options and examples

Learn about how to create and use ONTAP SAN drivers with your Astra Trident installation. This section provides backend configuration examples and details about how to map backends to StorageClasses.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
version		Always 1

Parameter	Description	Default
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 For seamless MetroCluster switchover, you must specify an SVM management LIF. This feature is tech preview .	“10.0.0.1”, “[2001:1234:abcd::fefe]”
dataLIF	IP address of protocol LIF. Use square brackets for IPv6. Cannot be updated after you set it	Derived by the SVM unless specified
useCHAP	Use CHAP to authenticate iSCSI for ONTAP SAN drivers [Boolean]	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 auth	“”
username	Username to connect to the cluster/SVM. Used for credential-based auth	“”
password	Password to connect to the cluster/SVM. Used for credential-based auth	“”

Parameter	Description	Default
svm	Storage virtual machine to use	Derived if an SVM managementLIF is specified
igroupName	Name of the igroup for SAN volumes to use	"trident-<backend-UUID>"
storagePrefix	Prefix used when provisioning new volumes in the SVM. Cannot be updated after you set it	"trident"
limitAggregateUsage	Fail provisioning if usage is above this percentage. Does not apply to Amazon FSx for ONTAP	"" (not enforced by default)
limitVolumeSize	Fail provisioning if requested volume size is above this value.	"" (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}	null
useREST	Boolean parameter to use ONTAP REST APIs. Tech preview Not supported with MetroCluster.	false

useREST considerations



- useREST is provided as a **tech preview** that is recommended for test environments and not for production workloads. When set to `true`, Astra Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.10 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.
- useREST is not supported with MetroCluster.

To communicate with the ONTAP cluster, you should provide the authentication parameters. This could be the username/password to a security login or an installed certificate.



If you are using an Amazon FSx for NetApp ONTAP backend, do not specify the `limitAggregateUsage` parameter. The `fsxadmin` and `vsadmin` roles provided by Amazon FSx for NetApp ONTAP do not contain the required access permissions to retrieve aggregate usage and limit it through Astra Trident.



Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.

For the `ontap-san` drivers, the default is to use all data LIF IPs from the SVM and to use iSCSI multipath. Specifying an IP address for the `dataLIF` for the `ontap-san` drivers forces them to disable multipath and use only the specified address.



When creating a backend, remember that `dataLIF` and `storagePrefix` cannot be modified after creation. To update these parameters, you will need to create a new backend.

`igroupName` can be set to an igroup that is already created on the ONTAP cluster. If unspecified, Astra Trident automatically creates an igroup named `trident-<backend-UUID>`. If providing a pre-defined `igroupName`, NetApp recommends using an igroup per Kubernetes cluster, if the SVM is to be shared between environments. This is necessary for Astra Trident to maintain IQN additions/deletions automatically.

Backends can also have igroups updated after creation:

- `igroupName` can be updated to point to a new igroup that is created and managed on the SVM outside of Astra Trident.
- `igroupName` can be omitted. In this case, Astra Trident will create and manage a `trident-<backend-UUID>` igroup automatically.

In both cases, volume attachments will continue to be accessible. Future volume attachments will use the updated igroup. This update does not disrupt access to volumes present on the backend.

A fully-qualified domain name (FQDN) can be specified for the `managementLIF` option.

`managementLIF` for all ONTAP drivers can also be set to IPv6 addresses. Make sure to install Trident with the `--use-ipv6` flag. Care must be taken to define `managementLIF` IPv6 address within square brackets.



When using IPv6 addresses, make sure `managementLIF` and `dataLIF` (if included in your backend definition) are defined within square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. If `dataLIF` is not provided, Astra Trident will fetch the IPv6 data LIFs from the SVM.

To enable the `ontap-san` drivers to use CHAP, set the `useCHAP` parameter to `true` in your backend definition. Astra Trident will then configure and use bidirectional CHAP as the default authentication for the SVM given in the backend. See [here](#) to learn about how it works.

For the `ontap-san-economy` driver, the `limitVolumeSize` option will also restrict the maximum size of the volumes it manages for qtrees and LUNs.



Astra Trident sets provisioning labels in the “Comments” field of all volumes created using the `ontap-san` driver. For each volume created, the “Comments” field on the FlexVol will be populated with all labels present on the storage pool it is placed in. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Backend configuration options for provisioning volumes

You can control how each volume is provisioned by default using these options in a special section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
<code>spaceAllocation</code>	Space-allocation for LUNs	“true”

Parameter	Description	Default
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	“”
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 “0”	If snapshotPolicy is “none”, else “”
splitOnClone	Split a clone from its parent upon creation	“false”
splitOnClone	Split a clone from its parent upon creation	“false”
encryption	Enable NetApp volume encryption	“false”
securityStyle	Security style for new volumes	“unix”
tieringPolicy	Tiering policy to use “none”	“snapshot-only” for pre-ONTAP 9.5 SVM-DR configuration



Using QoS policy groups with Astra Trident requires ONTAP 9.8 or later. It is recommended to use a non-shared QoS policy group and ensure the policy group is applied to each constituent individually. A shared QoS policy group will enforce the ceiling for the total throughput of all workloads.

Here’s an example with defaults defined:

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



For all volumes created using the `ontap-san` driver, Astra 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. Astra 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`, Astra Trident calculates the size of volumes as follows:

$$\text{Total volume size} = [(\text{PVC requested size}) / (1 - (\text{snapshotReserve percentage} / 100))] * 1.1$$

The 1.1 is the extra 10 percent Astra Trident adds to the FlexVol to accommodate the LUN metadata. For `snapshotReserve` = 5%, and PVC request = 5GiB, the total volume size is 5.79GiB and the available size is 5.5GiB. The `volume show` command should show results similar to this example:

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e42ec6fe_3baa_4af6_996d_134adbbb8e6d	online	RW	5.79GB	5.50GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

3 entries were displayed.

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 Astra Trident, the recommendation is to specify DNS names for LIFs instead of IP addresses.

ontap-san **driver with certificate-based authentication**

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,
  "storageDriverName": "ontap-san",
  "backendName": "DefaultSANBackend",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

ontap-san **driver with bidirectional CHAP**

This is a minimal backend configuration example. This basic configuration creates an `ontap-san` backend with `useCHAP` set to `true`.


```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "labels": {"k8scluster": "test-cluster-1", "backend": "testcluster1-
sanbackend"},
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "username": "vsadmin",
  "password": "secret"
}
```

ontap-san-economy **driver**

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "svm": "svm_iscsi_eco",
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "username": "vsadmin",
  "password": "secret"
}
```

Examples of backends with virtual storage pools

In the sample backend definition file shown below, specific defaults are set for all storage pools, such as `spaceReserve` at `none`, `spaceAllocation` at `false`, and `encryption` at `false`. The virtual storage pools are defined in the storage section.

In this example, some of the storage pool sets their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools overwrite the default values set above.

```

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

  "defaults": {
    "spaceAllocation": "false",
    "encryption": "false",
    "qosPolicy": "standard"
  },
  "labels":{"store": "san_store", "kubernetes-cluster": "prod-cluster-1"},
  "region": "us_east_1",
  "storage": [
    {
      "labels":{"protection":"gold", "creditpoints":"40000"},
      "zone":"us_east_1a",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "true",
        "adaptiveQosPolicy": "adaptive-extreme"
      }
    },
    {
      "labels":{"protection":"silver", "creditpoints":"20000"},
      "zone":"us_east_1b",
      "defaults": {
        "spaceAllocation": "false",
        "encryption": "true",
        "qosPolicy": "premium"
      }
    },
    {
      "labels":{"protection":"bronze", "creditpoints":"5000"},
      "zone":"us_east_1c",
      "defaults": {

```

```

        "spaceAllocation": "true",
        "encryption": "false"
    }
}
]
}

```

Here is an iSCSI example for the `ontap-san-economy` driver:

```

{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "svm": "svm_iscsi_eco",
  "useCHAP": true,
  "chapInitiatorSecret": "cl9qxIm36DKyawxy",
  "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
  "igroupName": "trident",
  "username": "vsadmin",
  "password": "secret",

  "defaults": {
    "spaceAllocation": "false",
    "encryption": "false"
  },
  "labels": {"store": "san_economy_store"},
  "region": "us_east_1",
  "storage": [
    {
      "labels": {"app": "oracledb", "cost": "30"},
      "zone": "us_east_1a",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "true"
      }
    },
    {
      "labels": {"app": "postgresdb", "cost": "20"},
      "zone": "us_east_1b",
      "defaults": {
        "spaceAllocation": "false",
        "encryption": "true"
      }
    }
  ]
}

```

```

    },
    {
      "labels":{"app":"mysqldb", "cost":"10"},
      "zone":"us_east_1c",
      "defaults": {
        "spaceAllocation": "true",
        "encryption": "false"
      }
    }
  ]
}

```

Map backends to StorageClasses

The following StorageClass definitions refer to the above virtual storage 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 (`protection-gold`) will map to the first, second virtual storage pool in the `ontap-nas-flexgroup` backend and the first virtual storage pool in the `ontap-san` backend. These are the only pool offering gold level protection.
- The second StorageClass (`protection-not-gold`) will map to the third, fourth virtual storage pool in `ontap-nas-flexgroup` backend and the second, third virtual storage pool in `ontap-san` backend. These are the only pools offering protection level other than gold.
- The third StorageClass (`app-mysqldb`) will map to the fourth virtual storage pool in `ontap-nas` backend and the third virtual storage pool in `ontap-san-economy` backend. These are the only pools offering storage pool configuration for `mysqldb` type app.
- The fourth StorageClass (`protection-silver-creditpoints-20k`) will map to the third virtual storage pool in `ontap-nas-flexgroup` backend and the second virtual storage pool in `ontap-san` backend. These are the only pools offering gold-level protection at 20000 creditpoints.
- The fifth StorageClass (`creditpoints-5k`) will map to the second virtual storage pool in `ontap-nas-economy` backend and the third virtual storage pool in `ontap-san` backend. These are the only pool offerings at 5000 creditpoints.

Astra Trident will decide which virtual storage pool is selected and will ensure the storage requirement is met.

```

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

```

Configure a backend with ONTAP NAS drivers

Learn about configuring an ONTAP backend with ONTAP and Cloud Volumes ONTAP NAS drivers.

- [Preparation](#)
- [Configuration and examples](#)

User permissions

Astra 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, Astra 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 Astra 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.

Preparation

Learn about how to prepare to configure an ONTAP backend with ONTAP NAS drivers. For all ONTAP backends, Astra Trident requires at least one aggregate assigned to the SVM.

For all ONTAP backends, Astra Trident requires at least one aggregate assigned to the SVM.

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 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. See [here](#) for more details.

Authentication

Astra 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:** Astra 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

Astra 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 Astra Trident releases. A custom security login role can be created and used with Astra Trident, but is not recommended.

A sample backend definition will look like this:

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

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/updating 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 k8sensv.key --cert ~/k8sensv.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 k8sensv.pem >> cert_base64
base64 -w 0 k8sensv.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...Vaaalllluuuuueeee",
  "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 backend update`.

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

#Update backend with tridentctl
$ tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident

+-----+-----+-----+
+-----+-----+
|      NAME      | STORAGE DRIVER |          UUID          |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |          9 |
+-----+-----+-----+
+-----+-----+
```



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 Astra Trident can communicate with the ONTAP backend and handle future volume operations.

Manage NFS export policies

Astra Trident uses NFS export policies to control access to the volumes that it provisions.

Astra Trident provides two options when working with export policies:

- Astra 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. Astra Trident adds node IPs that fall in these ranges to the export policy automatically. Alternatively, when no CIDRs are specified, any global-scoped unicast IP found on the nodes will be added to the export policy.
- Storage administrators can create an export policy and add rules manually. Astra Trident uses the default

export policy unless a different export policy name is specified in the configuration.

Dynamically manage export policies

The 20.04 release of CSI 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 have IPs in the range specified, supporting a finegrained and automated management.



The dynamic management of export policies is only available for CSI Trident. It is important to ensure that the worker nodes are not being NATed.

Example

There are two configuration options that must be used. Here's an example backend definition:

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap_nas_auto_export",
  "managementLIF": "192.168.0.135",
  "svm": "svm1",
  "username": "vsadmin",
  "password": "FaKePaSsWoRd",
  "autoExportCIDRs": ["192.168.0.0/24"],
  "autoExportPolicy": true
}
```



When using this feature, you must ensure that the root junction in your SVM has a precreated export policy with an export rule that permits the node CIDR block (such as the default export policy). Always follow NetApp's recommended best practice of dedicating a SVM for Astra Trident.

Here is an explanation of how this feature works using the example above:

- `autoExportPolicy` is set to `true`. This indicates that Astra Trident will create an export policy for the `svm1` SVM and handle the addition and deletion of rules using `autoExportCIDRs` address blocks. For example, a backend with UUID `403b5326-8482-40db-96d0-d83fb3f4daec` and `autoExportPolicy` set to `true` creates an export policy named `trident-403b5326-8482-40db-96d0-d83fb3f4daec` on the SVM.
- `autoExportCIDRs` contains a list of address blocks. This field is optional and it defaults to `["0.0.0.0/0", "::/0"]`. If not defined, Astra Trident adds all globally-scoped unicast addresses found on the worker nodes.

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 will be added to the export policy that Astra Trident creates. When Astra 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`. After filtering the IPs, Astra Trident creates export policy rules

for the client IPs it discovers, with one rule for each node it identifies.

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 Astra 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>
      fileType: ext4
```

As nodes are added to a Kubernetes cluster and registered with the Astra Trident controller, export policies of existing backends are updated (provided they fall in the address range specified in `autoExportCIDRs` for the backend).

When a node is removed, Astra Trident checks all backends that are online to remove the access rule for the node. By removing this node IP from the export policies of managed backends, Astra 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` will ensure that Astra Trident manages the export policies automatically. This will create a new export policy named after the backend's UUID and volumes that are present on the backend will use the newly created export policy when they are 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 Astra Trident pod on the node. Astra Trident will then update the export policy for backends it manages to reflect this IP change.

Configuration options and examples

Learn about how to create and use ONTAP NAS drivers with your Astra Trident installation. This section provides backend configuration examples and details about how to map backends to StorageClasses.

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-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 For seamless MetroCluster switchover, you must specify an SVM management LIF. This feature is tech preview .	"10.0.0.1", "[2001:1234:abcd::fefe]"
dataLIF	IP address of protocol LIF. Use square brackets for IPv6. Cannot be updated after you set it	Derived by the SVM unless specified
autoExportPolicy	Enable automatic export policy creation and updating [Boolean]	false
autoExportCIDRs	List of CIDRs to filter Kubernetes' node IPs against when autoExportPolicy is enabled	["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	""
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	""

Parameter	Description	Default
username	Username to connect to the cluster/SVM. Used for credential-based auth	
password	Password to connect to the cluster/SVM. Used for credential-based auth	
svm	Storage virtual machine to use	Derived if an SVM managementLIF is specified
igroupName	Name of the igroup for SAN volumes to use	"trident-<backend-UUID>"
storagePrefix	Prefix used when provisioning new volumes in the SVM. Cannot be updated after you set it	"trident"
limitAggregateUsage	Fail provisioning if usage is above this percentage. Does not apply to Amazon FSx for ONTAP	"" (not enforced by default)
limitVolumeSize	Fail provisioning if requested volume size is above this value.	"" (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}	null
nfsMountOptions	Comma-separated list of NFS mount options	""
qtreesPerFlexvol	Maximum Qtrees per FlexVol, must be in range [50, 300]	"200"
useREST	Boolean parameter to use ONTAP REST APIs. Tech preview Not supported with MetroCluster.	false



useREST considerations

- useREST is provided as a **tech preview** that is recommended for test environments and not for production workloads. When set to `true`, Astra Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.10 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.
- useREST is not supported with MetroCluster.

To communicate with the ONTAP cluster, you should provide the authentication parameters. This could be the username/password to a security login or an installed certificate.



If you are using an Amazon FSx for NetApp ONTAP backend, do not specify the `limitAggregateUsage` parameter. The `fsxadmin` and `vsadmin` roles provided by Amazon FSx for NetApp ONTAP do not contain the required access permissions to retrieve aggregate usage and limit it through Astra Trident.



Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.



When creating a backend, remember that the `dataLIF` and `storagePrefix` cannot be modified after creation. To update these parameters, you will need to create a new backend.

A fully-qualified domain name (FQDN) can be specified for the `managementLIF` option. A FQDN may also be specified for the `dataLIF` option, in which case the FQDN will be used for the NFS mount operations. This way you can create a round-robin DNS to load-balance across multiple data LIFs.

`managementLIF` for all ONTAP drivers can also be set to IPv6 addresses. Make sure to install Astra Trident with the `--use-ipv6` flag. Care must be taken to define the `managementLIF` IPv6 address within square brackets.



When using IPv6 addresses, make sure `managementLIF` and `dataLIF` (if included in your backend definition) are defined within square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. If `dataLIF` is not provided, Astra Trident will fetch the IPv6 data LIFs from the SVM.

Using the `autoExportPolicy` and `autoExportCIDRs` options, CSI Trident can manage export policies automatically. This is supported for all `ontap-nas-*` drivers.

For the `ontap-nas-economy` driver, the `limitVolumeSize` option will also restrict 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.

The `nfsMountOptions` parameter can be used to specify 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, Astra Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in either the storage class or the configuration file, then Astra Trident will not set any mount options on an associated persistent volume.



Astra Trident sets provisioning labels in the "Comments" field of all volumes created using `ontap-nas` and `ontap-nas-flexgroup`. Based on the driver used, the comments are set on the FlexVol (`ontap-nas`) or FlexGroup (`ontap-nas-flexgroup`). Astra Trident will copy all labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Backend configuration options for provisioning volumes

You can control how each volume is provisioned by default using these options in a special 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"
qosPolicy	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""
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", else ""
splitOnClone	Split a clone from its parent upon creation	"false"
encryption	Enable NetApp volume encryption	"false"
securityStyle	Security style for new volumes	"unix"
tieringPolicy	Tiering policy to use "none"	"snapshot-only" for pre-ONTAP 9.5 SVM-DR configuration
unixPermissions	Mode for new volumes	"777"
snapshotDir	Controls visibility of the .snapshot directory	"false"
exportPolicy	Export policy to use	"default"
securityStyle	Security style for new volumes	"unix"



Using QoS policy groups with Astra Trident requires ONTAP 9.8 or later. It is recommended to use a non-shared QoS policy group and ensure the policy group is applied to each constituent individually. A shared QoS policy group will enforce the ceiling for the total throughput of all workloads.

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`, Astra 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, Astra 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 lesser space than what they requested. Before v21.07, when the user requests a PVC (for example, 5GiB), with the `snapshotReserve` to 50 percent, they get only 2.5GiB 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 Astra 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 = 5GiB, the total volume size is $2/.5 = 10\text{GiB}$ and the available size is 5GiB, which is what the user requested in the PVC request. The `volume show` command should show results similar to this example:

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

2 entries were displayed.

Existing backends from previous installs will provision volumes as explained above when upgrading Astra Trident. For volumes that you created before upgrading, you should resize their volumes for the change to be observed. For example, a 2GiB PVC with `snapshotReserve=50` earlier resulted in a volume that provides 1GiB of writable space. Resizing the volume to 3GiB, for example, provides the application with 3GiB 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 **driver with certificate-based authentication**

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_"
}
```

ontap-nas **driver with auto export policy**

This example shows you how you can instruct Astra 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": "secret",
  "nfsMountOptions": "nfsvers=4",
}
```

ontap-nas-flexgroup **driver**

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

ontap-nas **driver with IPv6**

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "nas_ipv6_backend",
  "managementLIF": "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]",
  "labels": {"k8scluster": "test-cluster-east-1a", "backend": "test1-ontap-
ipv6"},
  "svm": "nas_ipv6_svm",
  "username": "vsadmin",
  "password": "netapp123"
}
```

ontap-nas-economy **driver**

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

Examples of backends with virtual storage pools

In the sample backend definition file shown below, specific defaults are set for all storage pools, such as `spaceReserve` at `none`, `spaceAllocation` at `false`, and `encryption` at `false`. The virtual storage pools are defined in the `storage` section.

In this example, some of the storage pool sets their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools overwrite the default values set above.

ontap-nas **driver**

```
{
  {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "admin",
    "password": "secret",
    "nfsMountOptions": "nfsvers=4",

    "defaults": {
      "spaceReserve": "none",
      "encryption": "false",
      "qosPolicy": "standard"
    },
    "labels": {"store": "nas_store", "k8scluster": "prod-cluster-1"},
    "region": "us_east_1",
    "storage": [
      {
        "labels": {"app": "msoffice", "cost": "100"},
        "zone": "us_east_1a",
        "defaults": {
```

```

        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755",
        "adaptiveQosPolicy": "adaptive-premium"
    },
    {
        "labels":{"app":"slack", "cost":"75"},
        "zone":"us_east_1b",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"app":"wordpress", "cost":"50"},
        "zone":"us_east_1c",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0775"
        }
    },
    {
        "labels":{"app":"mysqldb", "cost":"25"},
        "zone":"us_east_1d",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "false",
            "unixPermissions": "0775"
        }
    }
]
}

```

ontap-nas-flexgroup **driver**

```

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

```

```

"password": "secret",

"defaults": {
    "spaceReserve": "none",
    "encryption": "false"
},
"labels":{"store":"flexgroup_store", "k8scluster": "prod-cluster-1"},
"region": "us_east_1",
"storage": [
    {
        "labels":{"protection":"gold", "creditpoints":"50000"},
        "zone":"us_east_1a",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"protection":"gold", "creditpoints":"30000"},
        "zone":"us_east_1b",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0755"
        }
    },
    {
        "labels":{"protection":"silver", "creditpoints":"20000"},
        "zone":"us_east_1c",
        "defaults": {
            "spaceReserve": "none",
            "encryption": "true",
            "unixPermissions": "0775"
        }
    },
    {
        "labels":{"protection":"bronze", "creditpoints":"10000"},
        "zone":"us_east_1d",
        "defaults": {
            "spaceReserve": "volume",
            "encryption": "false",
            "unixPermissions": "0775"
        }
    }
]

```

```
}
```

ontap-nas-economy **driver**

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

  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {"store": "nas_economy_store"},
  "region": "us_east_1",
  "storage": [
    {
      "labels": {"department": "finance", "creditpoints": "6000"},
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {"department": "legal", "creditpoints": "5000"},
      "zone": "us_east_1b",
      "defaults": {
        "spaceReserve": "none",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {"department": "engineering", "creditpoints": "3000"},
      "zone": "us_east_1c",
      "defaults": {
        "spaceReserve": "none",
        "encryption": "true",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

```

    }
  },
  {
    "labels":{"department":"humanresource",
"creditpoints":"2000"},
    "zone":"us_east_1d",
    "defaults": {
      "spaceReserve": "volume",
      "encryption": "false",
      "unixPermissions": "0775"
    }
  }
]
}

```

Map backends to StorageClasses

The following StorageClass definitions refer to the above virtual storage 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 (`protection-gold`) will map to the first, second virtual storage pool in the `ontap-nas-flexgroup` backend and the first virtual storage pool in the `ontap-san` backend. These are the only pool offering gold level protection.
- The second StorageClass (`protection-not-gold`) will map to the third, fourth virtual storage pool in `ontap-nas-flexgroup` backend and the second, third virtual storage pool in `ontap-san` backend. These are the only pools offering protection level other than gold.
- The third StorageClass (`app-mysqldb`) will map to the fourth virtual storage pool in `ontap-nas` backend and the third virtual storage pool in `ontap-san-economy` backend. These are the only pools offering storage pool configuration for `mysqldb` type app.
- The fourth StorageClass (`protection-silver-creditpoints-20k`) will map to the third virtual storage pool in `ontap-nas-flexgroup` backend and the second virtual storage pool in `ontap-san` backend. These are the only pools offering gold-level protection at 20000 creditpoints.
- The fifth StorageClass (`creditpoints-5k`) will map to the second virtual storage pool in `ontap-nas-economy` backend and the third virtual storage pool in `ontap-san` backend. These are the only pool offerings at 5000 creditpoints.

Astra Trident will decide which virtual storage pool is selected and will ensure the storage requirement is met.


```

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

```

Use Astra Trident with Amazon FSx for NetApp ONTAP

[Amazon FSx for NetApp ONTAP](#), is a fully managed AWS service that enables customers to launch and run file systems powered by NetApp's ONTAP storage operating system. Amazon FSx for NetApp ONTAP enables you to leverage NetApp features, performance, and administrative capabilities you are familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx supports many of ONTAP's file system features and administration APIs.

A file system is the primary resource in Amazon FSx, analogous to an ONTAP cluster on premises. Within each SVM you can create one or multiple volumes, which are data containers that store the files and folders in your file system. With Amazon FSx for NetApp ONTAP, Data ONTAP will be provided as a managed file system in the cloud. The new file system type is called **NetApp ONTAP**.

Using Astra Trident with Amazon FSx for NetApp ONTAP, you can ensure Kubernetes clusters running in Amazon Elastic Kubernetes Service (EKS) can provision block and file persistent volumes backed by ONTAP.

Creating your Amazon FSx for ONTAP file system

Volumes created on Amazon FSx filesystems that have automatic backups enabled cannot be deleted by Trident. To delete PVCs, you need to manually delete the PV and the FSx for ONTAP volume.

To prevent this issue:

- Do not use **Quick create** to create the FSx for ONTAP file system. The quick create workflow enables automatic backups and does not provide an opt-out option.
- When using **Standard create**, disable automatic backup. Disabling automatic backups allows Trident to successfully delete a volume without further manual intervention.



Learn about Astra Trident

If you are new to Astra Trident, familiarize yourself by using the links provided below:

- [FAQs](#)
- [Requirements for using Astra Trident](#)
- [Deploy Astra Trident](#)
- [Best practices for configuring ONTAP, Cloud Volumes ONTAP, and Amazon FSx for NetApp ONTAP](#)
- [Integrate Astra Trident](#)
- [ONTAP SAN backend configuration](#)

- [ONTAP NAS backend configuration](#)

Learn more about driver capabilities [here](#).

Amazon FSx for NetApp ONTAP uses [FabricPool](#) to manage storage tiers. It enables you to store data in a tier, based on whether the data is frequently accessed.

Astra 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. It is not recommended to use the `fsxadmin` user, with Trident, as a `vsadmin` SVM user has access to more Astra Trident capabilities.

Drivers

You can integrate Astra Trident with Amazon FSx for NetApp ONTAP by using the following drivers:

- `ontap-san`: Each PV provisioned is a LUN within its own Amazon FSx for NetApp ONTAP volume.
- `ontap-san-economy`: Each PV provisioned is a LUN with a configurable number of LUNs per Amazon FSx for NetApp ONTAP volume.
- `ontap-nas`: Each PV provisioned is a full 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.

Authentication

Astra Trident offers two modes of authentication:

- **Certificate-based**: Astra Trident will communicate with the SVM on your FSx file system using a certificate installed on your SVM.
- **Credential-based**: You can use the `fsxadmin` user for your file system or the `vsadmin` user configured for your SVM.



We strongly recommend using the `vsadmin` user instead of the `fsxadmin` to configure your backend. Astra Trident will communicate with the FSx file system using this username and password.

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.

To learn more about authentication, see these links:

- [ONTAP NAS](#)
- [ONTAP SAN](#)

Deploy and configure Astra Trident on EKS with Amazon FSx for NetApp ONTAP

What you'll 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 and/or iSCSI](#).



Ensure that you follow the node preparation steps required for Amazon Linux and Ubuntu [Amazon Machine Images](#) (AMIs) depending on your EKS AMI type.

For other Astra Trident requirements, see [here](#).

Steps

1. Deploy Astra Trident using one of the [deployment methods](#).
2. Configure Astra Trident as follows:
 - a. Collect your SVM's management LIF DNS name. For example, by using the AWS CLI, find the `DNSName` entry under `Endpoints` → `Management` after running the following command:

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

3. Create and install certificates for authentication. If you are using an `ontap-san` backend, see [here](#). If you are using an `ontap-nas` backend, see [here](#).



You can log in to your file system (for example to install certificates) using SSH from anywhere that can reach your file system. Use the `fsxadmin` user, the password you configured when you created your file system, and the management DNS name from `aws fsx describe-file-systems`.

4. Create a backend file using your certificates and the DNS name of your management LIF, as shown in the sample below:

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

For information about creating backends, see these links:

- [Configure a backend with ONTAP NAS drivers](#)
- [Configure a backend with ONTAP SAN drivers](#)



Do not specify `dataLIF` for the `ontap-san` and `ontap-san-economy` drivers to allow Astra Trident to use multipath.



The `limitAggregateUsage` parameter will not work with the `vsadmin` and `fsxadmin` user accounts. The configuration operation will fail if you specify this parameter.

After deployment, perform the steps to create a [storage class](#), [provision a volume](#), and [mount the volume in a pod](#).

Find more information

- [Amazon FSx for NetApp ONTAP documentation](#)
- [Blog post on Amazon FSx for NetApp ONTAP](#)

Create backends with `kubectl`

A backend defines the relationship between Astra Trident and a storage system. It tells Astra Trident how to communicate with that storage system and how Astra Trident should provision volumes from it. After Astra 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 Astra 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 Astra 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 Astra 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 Astra 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, see 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). Astra 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 Astra 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: t@Ax@7q(>
```

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
Cloud Volumes Service for GCP	private_key_id	ID of the private key. Part of API key for GCP Service Account with CVS admin role

Storage platform Secret Fields description	Secret	Fields description
Cloud Volumes Service for GCP	private_key	Private key. Part of API key for GCP Service Account with CVS admin role
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 <code>ontap-san</code> and <code>ontap-san-economy</code>
ONTAP	chapInitiatorSecret	CHAP initiator secret. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code>
ONTAP	chapTargetUsername	Target username. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code>
ONTAP	chapTargetInitiatorSecret	CHAP target initiator secret. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code>

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
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 Astra 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:** Astra 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 the user's 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 Astra Trident (for example, during Astra 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](#).

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 Astra 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 will ensure the backend is still present and can be managed by using `tridentctl`.

Run the following command:

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

Astra 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. Astra 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.
In this case, `TridentBackendConfig` objects can be updated directly through Kubernetes.

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

Alternatively, make changes to the existing `TridentBackendConfig` CR by running 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 Astra 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 Astra 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 and Trident will continue to manage those volumes and snapshots until they are deleted.

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 Astra Trident. 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 Astra 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 Astra 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's 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 Storage 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 -n trident get tbc tbc-ontap-nas-backend -n trident
NAME                                BACKEND NAME                BACKEND UUID
PHASE    STATUS
tbc-ontap-nas-backend  ontap-nas-backend          52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7    Bound    Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
$ tridentctl get backend -n trident
+-----+-----+
+-----+-----+-----+-----+
|          NAME          | STORAGE DRIVER |          UUID          |
| STATE  | VOLUMES |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ontap-nas-backend      | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |          25 |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

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 will ensure 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, Astra Trident simply removes it without actually deleting the backend itself.

Manage storage classes

Find information about creating a storage class, deleting a storage class, and viewing existing storage classes.

Design a storage class

See [here](#) for more information on what storage classes are and how you configure them.

Create a storage class

After you have a storage class file, run the following command:

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

`<storage-class-file>` should be replaced with your storage class file name.

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 Astra Trident will continue to manage them.



Astra 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.

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 Astra Trident's synchronized storage classes, run the following command:

```
tridentctl get storageclass
```

- To view Astra 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.



You should only have one default storage class in your cluster at any given 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 Astra 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}] '
```

Perform volume operations

Learn about the features Astra Trident provides for managing your volumes.

- [Use CSI Topology](#)
- [Work with snapshots](#)
- [Expand volumes](#)
- [Import volumes](#)

Use CSI Topology

Astra Trident can selectively create and attach volumes to nodes present in a Kubernetes cluster by making use of the [CSI Topology feature](#). 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, Astra Trident uses CSI Topology.



Learn more about the CSI Topology feature [here](#).

Kubernetes provides two unique volume binding modes:

- With `VolumeBindingMode` set to `Immediate`, Astra 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 1.19 -1.24.

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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 Astra Trident is installed for Astra 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","kubernetes.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","kubernetes.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","kubernetes.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

Astra 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 must be 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 what an example backend definition looks like:

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



`supportedTopologies` 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, Astra Trident will create 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": "Netapp123",
"supportedTopologies": [
    {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"},
    {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
]
"storage": [
    {
        "labels": {"workload":"production"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-B",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
        ]
    }
]
}

```

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 will create 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:
  name: pvc-san
spec:
  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
  ----      -
  Normal    WaitForFirstConsumer  6s    persistentvolume-controller
waiting
for first consumer to be created before binding
  Message
  -----

```

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

Beginning with the 20.01 release of Astra Trident, you can create snapshots of PVs at the Kubernetes layer. You can use these snapshots to maintain point-in-time copies of volumes that have been created by Astra Trident and schedule the creation of additional volumes (clones). Volume snapshot is supported by the `ontap-nas`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `gcp-cvs`, and `azure-netapp-files` drivers.



This feature is available from Kubernetes 1.17 (beta) and is GA from 1.20. To understand the changes involved in moving from beta to GA, see [the release blog](#). With the graduation to GA, the `v1` API version is introduced and is backward compatible with `v1beta1` snapshots.

What you'll need

- Creating volume snapshots requires creating an external snapshot controller and Custom Resource Definitions (CRDs). This is the responsibility of the Kubernetes orchestrator being used (for example: Kubeadm, GKE, OpenShift).

If your Kubernetes distribution does not include the snapshot controller and CRDs, you can deploy them as follows.

1. Create volume snapshot CRDs.

For Kubernetes 1.20 and above, use `v1` snapshot CRDs with snapshot components of `v5.0` or above. For Kubernetes 1.19, use `v1beta1` with `v3.0.3` snapshot components.

v5.0 components

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.
yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents
.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

v3.0.3 components

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshots.yaml
```

2. Create the snapshot controller in the desired namespace. Edit the YAML manifests below to modify namespace.

For Kubernetes 1.20 and above use v5.0 or above. For Kubernetes version 1.19 use v3.0.3



Don't create a snapshot controller if setting up on-demand volume snapshots in a GKE environment. GKE uses a built-in, hidden snapshot-controller.

v5.0 controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

v3.0.3 controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```



CSI Snapshotter provides a [validating webhook](#) to help users validate existing v1beta1 snapshots and confirm they are valid resource objects. The validating webhook automatically labels invalid snapshot objects and prevents the creation of future invalid objects. The validating webhook is deployed by the Kubernetes orchestrator. See the instructions to deploy the validating webhook manually [here](#). Find examples of invalid snapshot manifests [here](#).

The example detailed below explains the constructs required for working with snapshots and shows how snapshots can be created and used.

Step 1: Set up a VolumeSnapshotClass

Before creating a volume snapshot, set up a VolumeSnapshotClass.

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.19, use
apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

The driver points to Astra Trident's 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.

Step 2: Create a snapshot of an existing PVC

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.19, use
apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

The snapshot is being created for a PVC named `pvc1`, and the name of the snapshot is set to `pvc1-snap`.

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

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

This created a `VolumeSnapshot` object. A `VolumeSnapshot` is analogous to a PVC and is associated with a `VolumeSnapshotContent` object that represents the actual snapshot.

It is possible to identify the `VolumeSnapshotContent` object for the `pvc1-snap` `VolumeSnapshot` by describing it.

```

$ kubectl describe volumesnapshots pvcl-snap
Name:          pvcl-snap
Namespace:     default
.
.
.
Spec:
  Snapshot Class Name:    pvcl-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-525400f3f660
  Source:
    API Group:
    Kind:      PersistentVolumeClaim
    Name:      pvcl
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
.
.

```

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.

Step 3: Create PVCs from VolumeSnapshots

See the following example for creating a PVC using a snapshot:

```

$ 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: pvcl-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io

```

dataSource shows that the PVC must be created using a VolumeSnapshot named pvcl-snap as the

source of the data. This instructs Astra Trident to create a PVC from the snapshot. After the PVC is created, it can be attached to a pod and used just like any other PVC.



When deleting a Persistent Volume with associated snapshots, the corresponding Trident volume is updated to a “Deleting state”. For the Astra Trident volume to be deleted, the snapshots of the volume should be removed.

Find more information

- [Volume snapshots](#)
- `VolumeSnapshotClass`

Expand volumes

Astra Trident provides Kubernetes users the ability to expand their volumes after they are created. Find information about the configurations required to expand iSCSI and NFS 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.

Overview

Expanding an iSCSI PV includes the following steps:

- Editing the `StorageClass` definition to set the `allowVolumeExpansion` field to `true`.
- Editing the PVC definition and updating the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.
- Attaching the PV must be attached 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, Astra Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
 - When attempting to resize an unattached PV, Astra 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.

The example below shows how expanding iSCSI PVs work.

Step 1: Configure the `StorageClass` to support volume expansion

```
$ 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

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

Astra 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

In this example, a pod is created that uses the `san-pvc`.

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

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

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 Astra 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 |
+-----+-----+-----+-----+
|          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

Astra Trident supports volume expansion for NFS PVs provisioned on `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `gcp-cvs`, 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
```

Astra Trident should create a 20MiB 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 20MiB PV to 1GiB, edit the PVC and set `spec.resources.requests.storage` to 1GB:

```

$ 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 Astra 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 |
+-----+-----+-----+-----+
|          BACKEND UUID   | STATE | MANAGED |
+-----+-----+-----+-----+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas |
| file          | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

Import volumes

You can import existing storage volumes as a Kubernetes PV using `tridentctl import`.

Drivers that support volume import

This table depicts the drivers that support importing volumes and the release they were introduced in.

Driver	Release
ontap-nas	19.04
ontap-nas-flexgroup	19.04
solidfire-san	19.04
azure-netapp-files	19.04

Driver	Release
gcp-cvs	19.04
ontap-san	19.04

Why should I import volumes?

There are several use cases for importing a volume into Trident:

- Containerizing an application and reusing its existing data set
- Using a clone of a data set for an ephemeral application
- Rebuilding a failed Kubernetes cluster
- Migrating application data during disaster recovery

How does the import work?

The Persistent Volume Claim (PVC) file is used by the volume import process to create the PVC. At a minimum, the PVC file should include the name, namespace, accessModes, and storageClassName fields as shown in the following example.

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

The `tridentctl` client is used to import an existing storage volume. Trident imports the volume by persisting volume metadata and creating the PVC and PV.

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

To import a storage volume, specify the name of the Astra Trident backend containing the volume, as well as the name that uniquely identifies the volume on the storage (for example: ONTAP FlexVol, Element Volume, CVS Volume path). The storage volume must allow read/write access and be accessible by the specified Astra Trident backend. The `-f` string argument is required and specifies the path to the YAML or JSON PVC file.

When Astra Trident receives the import volume request, 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.

When a volume is imported with the `--no-manage` argument, Trident does not perform any additional operations on the PVC or PV for the lifecycle of the objects. Because Trident ignores PV and PVC events for `--no-manage` objects, the storage volume is not deleted when the PV is deleted. Other operations such as volume clone and volume resize are also ignored. This option is 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.

Trident 19.07 and later handle the attachment of PVs and mounts the volume as part of importing it. For imports using earlier versions of Astra Trident, there will not be any operations in the data path and the volume import will not verify if the volume can be mounted. If a mistake is made with volume import (for example, the StorageClass is incorrect), you can recover by changing the reclaim policy on the PV to `retain`, deleting the PVC and PV, and retrying the volume import command.

ontap-nas and ontap-nas-flexgroup imports

Each volume created with the `ontap-nas` driver is a FlexVol on the ONTAP cluster. Importing FlexVols with the `ontap-nas` driver works the same. A FlexVol 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.



An ONTAP volume must be of type `rw` to be imported by Trident. If a volume is of type `dp`, it is a SnapMirror destination volume; you should break the mirror relationship before importing the volume into Trident.



The `ontap-nas` driver cannot import and manage qtrees. The `ontap-nas` and `ontap-nas-flexgroup` drivers do not allow duplicate volume names.

For example, to import a volume named `managed_volume` on a backend named `ontap_nas`, use the following command:

```
$ 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	

To import a volume named `unmanaged_volume` (on the `ontap_nas` backend), which Trident will not manage, use the following command:

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

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

When using the `--no-manage` argument, Trident does not rename the volume or validate if the volume was mounted. The volume import operation fails if the volume was not mounted manually.



A previously existing bug with importing volumes with custom UnixPermissions has been fixed. You can specify `unixPermissions` in your PVC definition or backend configuration, and instruct Astra Trident to import the volume accordingly.

ontap-san import

Astra Trident can also import ONTAP SAN FlexVols that contain a single LUN. This is consistent with the `ontap-san` driver, which creates a FlexVol for each PVC and a LUN within the FlexVol. You can use the `tridentctl import` command in the same way as in other cases:

- Include the name of the `ontap-san` backend.
- Provide the name of the FlexVol that needs to be imported. Remember, this FlexVol contains only one LUN that must be imported.
- Provide the path of the PVC definition that must be used with the `-f` flag.
- Choose between having the PVC managed or unmanaged. By default, Trident will manage the PVC and rename the FlexVol and LUN on the backend. To import as an unmanaged volume, pass the `--no-manage` flag.



When importing an unmanaged `ontap-san` volume, you should make sure that the LUN in the FlexVol is named `lun0` and is mapped to an `igroup` with the desired initiators. Astra Trident automatically handles this for a managed import.

Astra Trident will then import the FlexVol and associate it with the PVC definition. Astra Trident also renames the FlexVol to the `pvc-<uuid>` format and the LUN within the FlexVol to `lun0`.



It is recommended to import volumes that do not have existing active connections. If you are looking to import an actively used volume, clone the volume first and then do the import.

Example

To import the `ontap-san-managed` FlexVol that is present on the `ontap_san_default` backend, run the `tridentctl import` command as:

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



An ONTAP volume must be of type `rw` to be imported by Astra Trident. If a volume is of type `dp`, it is a SnapMirror destination volume; you should break the mirror relationship before importing the volume into Astra Trident.

element import

You can import NetApp Element software/NetApp HCI volumes to your Kubernetes cluster with Trident. You need the name of your Astra Trident backend, and the unique name of the volume and the PVC file as the arguments for the `tridentctl import` command.

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



The Element driver supports duplicate volume names. If there are duplicate volume names, Trident's volume import process returns an error. As a workaround, clone the volume and provide a unique volume name. Then import the cloned volume.

gcp-cvs **import**



To import a volume backed by the NetApp Cloud Volumes Service in GCP, identify the volume by its volume path instead of its name.

To import an `gcp-cvs` volume on the backend called `gcpcvs_YEppr` with the volume path of `adroit-jolly-swift`, use the following command:

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

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



The volume path is the portion of the volume's export path after the `:/`. For example, if the export path is `10.0.0.1:/adroit-jolly-swift`, the volume path is `adroit-jolly-swift`.

azure-netapp-files **import**

To import an `azure-netapp-files` volume on the backend called `azurenetafiles_40517` with the volume path `importvol1`, run the following command:


```
$ tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-pvc-file> -n trident
```

PROTOCOL	NAME	BACKEND UUID	SIZE	STORAGE CLASS	STATE	MANAGED
file	pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab	1c01274f-d94b-44a3-98a3-04c953c9a51e	100 GiB	anf-storage	online	true



The volume path for the ANF volume is present in the mount path after the `:/`. For example, if the mount path is `10.0.0.2:/importvol1`, the volume path is `importvol1`.

Prepare the worker node

All of the worker nodes in the Kubernetes cluster need to be able to mount the volumes that you have provisioned for your pods. If you are using the `ontap-nas`, `ontap-nas-economy`, or `ontap-nas-flexgroup` driver for one of your backends, your worker nodes need the NFS tools. Otherwise they require the iSCSI tools.

Recent versions of RedHat CoreOS have both NFS and iSCSI installed by default.



You should always reboot your worker nodes after installing the NFS or iSCSI tools, or else attaching volumes to containers might fail.

Node service discovery

Beginning in 22.07, Astra Trident attempts to automatically detect if the node is capable of running iSCSI or NFS services. Astra Trident creates events for the node to identify the services discovered. You can review these events using the command:

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

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

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



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.

NFS volumes

Protocol	Operating system	Commands
NFS	RHEL/CentOS 7	<code>sudo yum install -y nfs-utils</code>
NFS	Ubuntu	<code>sudo apt-get install -y nfs-common</code>



You should ensure that the NFS service is started up during boot time.


iSCSI volumes

Consider the following when using iSCSI volumes:

- 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, ensure that the CHAP authentication algorithm is set to MD5 in `/etc/iscsi/iscsid.conf`.

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

- When using worker nodes that run RHEL/RedHat CoreOS with iSCSI PVs, make sure to specify the `discard` mountOption in the StorageClass to perform inline space reclamation. See [RedHat's documentation](#).

Protocol	Operating system	Commands
iSCSI	RHEL/CentOS	<ol style="list-style-type: none"> 1. Install the following system packages: <pre>sudo yum install -y lsscsi iscsi-initiator- utils sg3_utils device- mapper-multipath</pre> 2. Check that iscsi-initiator-utils version is 6.2.0.874-2.el7 or later: <pre>rpm -q iscsi-initiator- utils</pre> 3. Set scanning to manual: <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> 4. Enable multipathing: <pre>sudo mpathconf --enable --with_multipathd y --find_multipaths n</pre> <div>  <p>Ensure etc/multipat h.conf contains find_multipa ths no under defaults.</p> </div> 5. Ensure that iscsid and multipathd are running: <pre>sudo systemctl enable --now iscsid multipathd</pre> 6. Enable and start iscsi: <pre>sudo systemctl enable --now iscsi</pre>

Protocol	Operating system	Commands
iSCSI	Ubuntu	<ol style="list-style-type: none"> 1. Install the following system packages: <pre>sudo apt-get install -y open-iscsi lsscsi sg3- utils multipath-tools scsitools</pre> 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): <pre>dpkg -l open-iscsi</pre> 3. Set scanning to manual: <pre>sudo sed -i 's/^\(node.session.scan \).*\/1 = manual/' /etc/iscsi/iscsid.conf</pre> 4. Enable multipathing: <pre>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</pre> <div>  <p>Ensure etc/multipath.conf contains find_multipaths no under defaults.</p> </div> 5. Ensure that open-iscsi and multipath-tools are enabled and running: <pre>sudo systemctl status multipath-tools sudo systemctl enable --now open- iscsi.service</pre>



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.

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

Monitor Astra Trident

Astra Trident provides a set of Prometheus metrics endpoints that you can use to monitor Astra Trident's performance.

The metrics provided by Astra Trident enable you to do the following:

- Keep tabs on Astra 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 Astra 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.

What you'll need

- A Kubernetes cluster with Astra 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 Astra Trident manages, the volumes it creates, and so on. This [blog](#) explains how you can use Prometheus and Grafana with Astra Trident to retrieve metrics. The blog explains how you can run Prometheus as an operator in your Kubernetes cluster and the creation of a ServiceMonitor to obtain Astra Trident's metrics.

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 Astra Trident's metrics.

In addition to metrics available directly from Astra Trident, kubelet exposes many `kubelet_volume_*` metrics via its own metrics endpoint. Kubelet can provide information about the volumes that are attached, and pods and other internal operations it handles. See [here](#).

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 Astra 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 Astra 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 Astra Trident**

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

Get Astra 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 Astra Trident AutoSupport telemetry

By default, Astra Trident sends Prometheus metrics and basic backend information to NetApp on a daily cadence.

- To stop Astra Trident from sending Prometheus metrics and basic backend information to NetApp, pass the `--silence-autosupport` flag during Astra Trident installation.
- Astra Trident can also send container logs to NetApp Support on-demand via `tridentctl send autosupport`. You will need to trigger Astra Trident to upload its logs. Before you submit logs, you should accept NetApp's [privacy policy](#).
- Unless specified, Astra Trident fetches the logs from the past 24 hours.
- You can specify the log retention timeframe 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 Astra 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 Astra 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 Astra 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.

Astra Trident for Docker

Prerequisites for deployment

You have to install and configure the necessary protocol prerequisites on your host before you can deploy Astra Trident.

- 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:

Protocol	Operating system	Commands
NFS	RHEL/CentOS 7	<code>sudo yum install -y nfs-utils</code>
NFS	Ubuntu	<code>sudo apt-get install -y nfs-common</code>

Protocol	Operating system	Commands
iSCSI	RHEL/CentOS 7	<ol style="list-style-type: none"> 1. Install the following system packages: <pre>sudo yum install -y lsscsi iscsi-initiator- utils sg3_utils device- mapper-multipath</pre> 2. Start the multipathing daemon: <pre>sudo mpathconf --enable --with_multipathd y</pre> 3. Ensure that <code>iscsid</code> and <code>multipathd</code> are enabled and running: <pre>sudo systemctl enable iscsid multipathd sudo systemctl start iscsid multipathd</pre> 4. Discover the iSCSI targets: <pre>sudo iscsiadm -m discoverydb -t st -p <DATA_LIF_IP> --discover</pre> 5. Log in to the discovered iSCSI targets: <pre>sudo iscsiadm -m node -p <DATA_LIF_IP> --login</pre> 6. Enable and start <code>iscsi</code>: <pre>sudo systemctl enable iscsi sudo systemctl start iscsi</pre>

Protocol	Operating system	Commands
iSCSI	Ubuntu	<ol style="list-style-type: none"> 1. Install the following system packages: <pre>sudo apt-get install -y open-iscsi lsscsi sg3- utils multipath-tools scsitools</pre> 2. Enable multipathing: <pre>sudo tee /etc/multipath.conf < ←'EOF' defaults { user_friendly_names yes find_multipaths yes } EOF sudo service multipath- tools restart</pre> 3. Ensure that <code>iscsid</code> and <code>multipathd</code> are running: <pre>sudo service open-iscsi start sudo service multipath- tools start</pre> 4. Discover the iSCSI targets: <pre>sudo iscsiadm -m discoverydb -t st -p <DATA_LIF_IP> --discover</pre> 5. Log in to the discovered iSCSI targets: <pre>sudo iscsiadm -m node -p <DATA_LIF_IP> --login</pre>

Deploy Astra Trident

Astra Trident for Docker provides direct integration with the Docker ecosystem for NetApp's 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 Astra Trident can run concurrently on the same host. This allows simultaneous connections to multiple storage systems and storage types, with the ability 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 Astra Trident.

Docker managed plugin method (version 1.13/17.03 and later)



Before you begin

If you have used Astra Trident pre Docker 1.13/17.03 in the traditional daemon method, ensure that you stop the Astra 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": "secret",
  "aggregate": "aggr1"
}
EOF
```

2. Start Astra Trident using the managed plugin system.

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:21.07 config=myConfigFile.json
```

3. Begin using Astra 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/v21.04.0/trident-
installer-21.07.0.tar.gz
tar xzf trident-installer-21.07.0.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": "secret",
    "aggregate": "aggr1"
}
EOF
```

2. After placing the binary and creating the configuration file(s), 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 Astra 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 CentOS 7/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 Astra Trident

You can safely upgrade Astra 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 Astra 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 Astra 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 Astra Trident (pre-20.10) to Astra Trident 20.10 or later, you might run into an error. For more information, see [Known Issues](#). If you run into the error, you should first disable the plugin, then remove the plugin, and then install the required Astra 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 Astra 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 Astra 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 Astra 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 20GiB 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 Astra Trident for Docker.

Clone a volume

When using the `ontap-nas`, `ontap-san`, `solidfire-san`, and `gcp-cvs` storage drivers, Astra 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

```
[me@host ~]$ 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"
        }
      ]
    }
  }
]

[me@host ~]$ docker volume create -d ontap-nas --name clonedVolume -o
from=firstVolume
clonedVolume

[me@host ~]$ docker volume rm clonedVolume
[me@host ~]$ docker volume create -d ontap-nas --name volFromSnap -o
from=firstVolume -o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

[me@host ~]$ 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 Astra Trident (for example: an `ext4`-formatted `/dev/sdc1` will not be accessible via Astra 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 and iSCSI 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 <code>none</code> (thin provisioned) and <code>volume</code> (thick provisioned).
snapshotPolicy	This will set the snapshot policy to the desired value. The default is <code>none</code> , 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 <code>.snapshot</code> 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 <code>ontap-nas-economy</code> .
splitOnClone	When cloning a volume, this will cause ONTAP to immediately split the clone from its parent. The default is <code>false</code> . 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.
encryption	<div>This will enable NetApp Volume Encryption (NVE) on the new volume, defaults to <code>false</code>. NVE must be licensed and enabled on the cluster to use this option.</div> <div> NetApp Aggregate Encryption (NAE) is not currently supported in Trident.</div>

Option	Description
<code>tieringPolicy</code>	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
<code>unixPermissions</code>	This controls the permission set for the volume itself. By default the permissions will be set to <code>---rwxr-xr-x</code> , or in numerical notation <code>0755</code> , and <code>root</code> will be the owner. Either the text or numerical format will work.
<code>snapshotDir</code>	Setting this to <code>true</code> will make the <code>.snapshot</code> directory visible to clients accessing the volume. The default value is <code>false</code> , meaning that visibility of the <code>.snapshot</code> directory is disabled by default. Some images, for example the official MySQL image, don't function as expected when the <code>.snapshot</code> directory is visible.
<code>exportPolicy</code>	Sets the export policy to be used for the volume. The default is <code>default</code> .
<code>securityStyle</code>	Sets the security style to be used for access to the volume. The default is <code>unix</code> . Valid values are <code>unix</code> and <code>mixed</code> .

The following additional options are for iSCSI **only**:

Option	Description
<code>fileSystemType</code>	Sets the file system used to format iSCSI volumes. The default is <code>ext4</code> . Valid values are <code>ext3</code> , <code>ext4</code> , and <code>xfs</code> .
<code>spaceAllocation</code>	Setting this to <code>false</code> will turn off the LUN's space-allocation feature. The default value is <code>true</code> , 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 10GiB volume:

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

- Create a 100GiB 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 20MiB.

If the snapshot reserve is not specified and the snapshot policy is `none`, Trident will 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 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 1GiB 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 10GiB Gold volume:

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- Create a 100GiB Bronze volume:

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

CVS on GCP volume options

Volume create options for the CVS on GCP driver include the following:

Option	Description
size	The size of the volume, defaults to 100 GiB for CVS-Performance volumes or 300 GiB for CVS volumes.
serviceLevel	The CVS service level of the volume, defaults to standard. Valid values are standard, premium, and extreme.
snapshotReserve	This will set the snapshot reserve to the desired percentage. The default is no value, meaning CVS will select the snapshot reserve (usually 0%).

Examples

- Create a 2TiB volume:

```
docker volume create -d netapp --name demo -o size=2T
```

- Create a 5TiB premium volume:

```
docker volume create -d netapp --name demo -o size=5T -o  
serviceLevel=premium
```

The minimum volume size is 100 GiB for CVS-Performance volumes, or 300 GiB for CVS volumes.

Azure NetApp Files volume options

Volume create options for the Azure NetApp Files driver include the following:

Option	Description
size	The size of the volume, defaults to 100 GB.

Examples

- Create a 200GiB volume:

```
docker volume create -d netapp --name demo -o size=200G
```

The minimum volume size is 100 GB.

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 Astra 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 `systemctl status rpcbind` or its equivalent.

Manage multiple Astra 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 Astra Trident configurations.

Global configuration options

These configuration options apply to all Astra 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, azure-netapp-files, or gcp-cvs
storagePrefix	Optional prefix for volume names. Default: "netappdvp_".	staging_
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. We recommend 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
<code>size</code>	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
<code>managementLIF</code>	IP address of ONTAP management LIF. You can specify a fully-qualified domain name (FQDN).	10.0.0.1
<code>dataLIF</code>	IP address of protocol LIF; will be derived if not specified. For the <code>ontap-nas</code> drivers only , you can specify an FQDN, in which case the FQDN will be used for the NFS mount operations. For the <code>ontap-san</code> drivers, the default is to use all data LIF IPs from the SVM and to use iSCSI multipath. Specifying an IP address for <code>dataLIF</code> for the <code>ontap-san</code> drivers forces the driver to disable multipath and use only the specified address.	10.0.0.2
<code>svm</code>	Storage virtual machine to use (required, if management LIF is a cluster LIF)	<code>svm_nfs</code>
<code>username</code>	Username to connect to the storage device	<code>vsadmin</code>
<code>password</code>	Password to connect to the storage device	<code>secret</code>

Option	Description	Example
aggregate	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the <code>ontap-nas-flexgroup</code> 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 <code>ontap-nas</code> and <code>ontap-nas-economy</code> drivers. See NFS host configuration information here.	-o nfsvers=4
igroupName	The igroup used by the plugin; defaults to “netappdvp”. Available only for the <code>ontap-san</code> driver.	myigroup
limitVolumeSize	Maximum requestable volume size and qtree parent volume size. For the <code>ontap-nas-economy</code> driver, this option additionally limits the size of the FlexVols that it creates.	300g
qtreesPerFlexvol	Maximum qtrees per FlexVol, must be in range [50, 300], default is 200. For the <code>ontap-nas-economy</code> driver, this option allows customizing the maximum number of qtrees per FlexVol.	300

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

Option	Description	Example
snapshotReserve	Snapshot reserve percentage, default is "" to accept ONTAP's default	10
splitOnClone	Split a clone from its parent upon creation, defaults to "false"	false
encryption	Enable NetApp Volume Encryption, defaults to "false"	true
unixPermissions	NAS option for provisioned NFS volumes, defaults to "777"	777
snapshotDir	NAS option for access to the .snapshot directory, defaults to "false"	true
exportPolicy	NAS option for the NFS export policy to use, defaults to "default"	default
securityStyle	NAS option for access to the provisioned NFS volume, defaults to "unix"	mixed
fileSystemType	SAN option to select the file system type, defaults to "ext4"	xf
tieringPolicy	Tiering policy to use, default is "none"; "snapshot-only" for pre-ONTAP 9.5 SVM-DR configuration	none

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 FlexVols. 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 FlexVols. 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 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 FlexVols. 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 size is 100GB.
- Cloning is not supported for FlexGroup Volumes.

For information about FlexGroups and workloads that are appropriate for FlexGroups see the [NetApp FlexGroup Volume Best Practices and Implementation Guide](#).

To get advanced features and huge 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`.

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": "secret",
  "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": "secret",
  "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": "secret",
  "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": "secret",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}
```

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": "secret",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}
```

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	<code>https://&lt;login&gt;:&lt;password&gt;@&lt;mvip&gt;/json-rpc/&lt;element-version&gt;</code>	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"
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
      }
    }
  ]
}
```

Cloud Volumes Service (CVS) on GCP configuration

Trident now includes support for smaller volumes with the default CVS service type on [GCP](#). For backends created with `storageClass=software`, volumes will now have a minimum provisioning size of 300 GiB.

NetApp recommends customers consume sub-1TiB volumes for non-production workloads. CVS currently provides this feature under Controlled Availability and does not provide technical support.



Sign up for access to sub-1TiB volumes [here](#).



When deploying backends using the default CVS service type `storageClass=software`, you should obtain access to the sub-1TiB volumes feature on GCP for the Project Number(s) and Project ID(s) in question. This is necessary for Trident to provision sub-1TiB volumes. If not, volume creations **will fail** for PVCs that are <600 GiB. Obtain access to sub-1TiB volumes using [this form](#).

Volumes created by Trident for the default CVS service level will be provisioned as follows:

- PVCs that are smaller than 300 GiB will result in Trident creating a 300 GiB CVS volume.
- PVCs that are between 300 GiB to 600 GiB will result in Trident creating a CVS volume of the requested size.
- PVCs that are between 600 GiB and 1 TiB will result in Trident creating a 1TiB CVS volume.
- PVCs that are greater than 1 TiB will result in Trident creating a CVS volume of the requested size.

In addition to the global configuration values, when using CVS on GCP, these options are available.

Option	Description	Example
<code>apiRegion</code>	CVS account region (required). Is the GCP region where this backend will provision volumes.	"us-west2"
<code>projectNumber</code>	GCP project number (required). Can be found in the GCP web portal's Home screen.	"123456789012"
<code>hostProjectNumber</code>	GCP shared VPC host project number (required if using a shared VPC)	"098765432109"
<code>apiKey</code>	API key for GCP service account with CVS admin role (required). Is the JSON-formatted contents of a GCP service account's private key file (copied verbatim into the backend config file). The service account must have the <code>netappcloudvolumes.admin</code> role.	(contents of the private key file)
<code>secretKey</code>	CVS account secret key (required). Can be found in the CVS web portal in Account settings > API access.	"default"

Option	Description	Example
proxyURL	Proxy URL if proxy server required to connect to the CVS account. The proxy server can either be an HTTP proxy or an HTTPS proxy. In case of an HTTPS proxy, certificate validation is skipped to allow the usage of self-signed certificates in the proxy server. Proxy servers with authentication enabled are not supported.	"http://proxy-server-hostname/"
nfsMountOptions	NFS mount options; defaults to "-o nfsvers=3"	"nfsvers=3,proto=tcp,timeo=600"
serviceLevel	Performance level (standard, premium, extreme), defaults to "standard"	"premium"
network	GCP network used for CVS volumes, defaults to "default"	"default"



If using a shared VPC network, you should specify both `projectNumber` and `hostProjectNumber`. In that case, `projectNumber` is the service project and `hostProjectNumber` is the host project.



The NetApp Cloud Volumes Service for GCP does not support CVS-Performance volumes less than 100 GiB in size, or CVS volumes less than 300 GiB in size. To make it easier to deploy applications, Trident automatically creates volumes of the minimum size if a too-small volume is requested.

When using CVS on GCP, these default volume option settings are available.

Option	Description	Example
exportRule	NFS access list (addresses and/or CIDR subnets), defaults to "0.0.0.0/0"	"10.0.1.0/24,10.0.2.100"
snapshotDir	Controls visibility of the <code>.snapshot</code> directory	"false"
snapshotReserve	Snapshot reserve percentage, default is "" to accept the CVS default of 0	"10"
size	Volume size, defaults to "100GiB"	"10T"

Example CVS on GCP configuration file

```
{
  "version": 1,
  "storageDriverName": "gcp-cvs",
  "projectNumber": "012345678901",
  "apiRegion": "us-west2",
  "apiKey": {
    "type": "service_account",
    "project_id": "my-gcp-project",
    "private_key_id": "1234567890123456789012345678901234567890",
    "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZ
srtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisI
sAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSa
PIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZN
chRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllz
ZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl
/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kw
s8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY
9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHc
zZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHi
sIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOgu
SaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyA
ZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz
llzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3
bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4
Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5o
jY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nzn
HczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtr
HisIsAbOguSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbO
guSaPIKeyAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKe
yAZNchRAGzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRA
GzllzZE4jK3bl/qP8B4Kws8zX5ojY9m\nznHczZsrtrHisIsAbOguSaPIKeyAZNchRAGzllzZE4j
K3bl/qP8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgWgGa==\n-----END PRIVATE
KEY-----\n",
    "client_email": "cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com",
    "client_id": "123456789012345678901",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com"
```

```

    },
    "proxyURL": "http://proxy-server-hostname/"
  }

```

Azure NetApp Files configuration

To configure and use an [Azure NetApp Files](#) backend, you will need the following:

- 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
- Azure location that contains at least one [delegated subnet](#)



If you're using Azure NetApp Files for the first time or in a new location, some initial configuration is required that the [quickstart guide](#) will walk you through.



Astra Trident 21.04.0 and earlier do not support Manual QoS capacity pools.

Option	Description	Default
version	Always 1	
storageDriverName	"azure-netapp-files"	
backendName	Custom name for the storage backend	Driver name + "_" + random characters
subscriptionID	The subscription ID from your Azure subscription	
tenantID	The tenant ID from an App Registration	
clientID	The client ID from an App Registration	
clientSecret	The client secret from an App Registration	
serviceLevel	One of "Standard", "Premium" or "Ultra"	"" (random)
location	Name of the Azure location new volumes will be created in	"" (random)

Option	Description	Default
virtualNetwork	Name of a virtual network with a delegated subnet	"" (random)
subnet	Name of a subnet delegated to Microsoft.Netapp/volumes	"" (random)
nfsMountOptions	Fine-grained control of NFS mount options	"-o nfsvers=3"
limitVolumeSize	Fail provisioning if requested volume size is above this value	"" (not enforced by default)



The Azure NetApp Files service does not support volumes less than 100 GB in size. To make it easier to deploy applications, Trident automatically creates 100 GB volumes if a smaller volume is requested.

You can control how each volume is provisioned by default using these options in a special section of the configuration.

Option	Description	Default
exportRule	The export rule(s) for new volumes. Must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation.	"0.0.0.0/0"
snapshotDir	Controls visibility of the .snapshot directory	"false"
size	The default size of new volumes	"100G"

Example Azure NetApp Files configurations

Example 1: Minimal backend configuration for azure-netapp-files

This is the absolute minimum backend configuration. With this configuration, Trident will discover all of your NetApp accounts, capacity pools, and subnets delegated to ANF in every location worldwide, and place new volumes on one of them randomly.

This configuration is useful when you're just getting started with ANF and trying things out, but in practice you're going to want to provide additional scoping for the volumes you provision to make sure that they have the characteristics you want and end up on a network that's close to the compute that's using it. See the subsequent examples for more details.

```
{
  "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"
}
```

Example 2: Single location and specific service level for azure-netapp-files

This backend configuration places volumes in Azure's "eastus" location in a "Premium" capacity pool. Trident automatically discovers all of the subnets delegated to ANF in that location and will place 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": "Premium"
}
```

Example 3: Advanced configuration for azure-netapp-files

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": "Premium",
  "virtualNetwork": "my-virtual-network",
  "subnet": "my-subnet",
  "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
  "limitVolumeSize": "500Gi",
  "defaults": {
    "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
    "size": "200Gi"
  }
}
```

Example 4: Virtual storage pools with azure-netapp-files

This backend configuration defines multiple [pools of storage](#) 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.

This is just scratching the surface of the power of virtual storage pools and their labels.

```

{
  "version": 1,
  "storageDriverName": "azure-netapp-files",
  "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
  "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
  "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
  "clientSecret": "SECRET",
  "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
  "labels": {
    "cloud": "azure"
  },
  "location": "eastus",

  "storage": [
    {
      "labels": {
        "performance": "gold"
      },
      "serviceLevel": "Ultra"
    },
    {
      "labels": {
        "performance": "silver"
      },
      "serviceLevel": "Premium"
    },
    {
      "labels": {
        "performance": "bronze"
      },
      "serviceLevel": "Standard",
    }
  ]
}

```

Known issues and limitations

Find information about known issues and limitations when using Astra 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 Astra 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 Astra 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.

Frequently asked questions

Find answers to the frequently asked questions about installing, configuring, upgrading, and troubleshooting Astra Trident.

General questions

How frequently is Astra Trident released?

Astra Trident is released every three months: January, April, July, and October. This is one month after a Kubernetes release.

Does Astra Trident support all the features that are released in a particular version of Kubernetes?

Astra 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 Astra Trident have any dependencies on other NetApp products for its functioning?

Astra 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 Astra Trident configuration details?

Use the `tridentctl get` command to obtain more information about your Astra Trident configuration.

Can I obtain metrics on how storage is provisioned by Astra Trident?

Yes. Trident 20.01 introduces Prometheus endpoints that can be used to gather information about Astra Trident's 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 Astra 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 Astra Trident is recommended if you want to use all the new features provided by current and future releases.

Install and use Astra Trident on a Kubernetes cluster

What are the supported versions of `etcd`?

Astra Trident no longer needs an `etcd`. It uses CRDs to maintain state.

Does Astra Trident support an offline install from a private registry?

Yes, Astra Trident can be installed offline. See [here](#).

Can I install Astra Trident be remotely?

Yes. Astra 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 Astra Trident?

Astra 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 Astra Trident is installed is lost or the pod is otherwise inaccessible, Kubernetes automatically re-deploys the pod to a healthy node in your cluster. Astra Trident is control-plane only, so currently mounted pods are not affected if Astra Trident is re-deployed.

Does Astra Trident need access to the kube-system namespace?

Astra 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 Astra 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. See [here](#).

Can I locally generate the exact manifest files Astra Trident uses for installation?

You can locally generate and modify the exact manifest files Astra Trident uses for installation, if needed. See [here](#).

Can I share the same ONTAP backend SVM for two separate Astra Trident instances for two separate Kubernetes clusters?

Although it is not advised, you can use the same backend SVM for two Astra 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 is not used for both instances.

Is it possible to install Astra Trident under ContainerLinux (formerly CoreOS)?

Astra Trident is simply a Kubernetes pod and can be installed wherever Kubernetes is running.

Can I use Astra Trident with NetApp Cloud Volumes ONTAP?

Yes, Astra Trident is supported on AWS, Google Cloud, and Azure.

Does Astra Trident work with Cloud Volumes Services?

Yes, Astra Trident supports the Azure NetApp Files service in Azure as well as the Cloud Volumes Service in GCP.

Troubleshooting and support

Does NetApp support Astra Trident?

Although Astra 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 [Astra Trident Github](#) and mention **RFE** in the subject and description of the issue.

Where do I raise a defect?

Create an issue on [Astra Trident Github](#). Make sure to include all the necessary information and logs pertaining to the issue.

What happens if I have quick question on Astra 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 Astra [Discord channel](#) or GitHub.

My storage system's password has changed and Astra 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.

Astra Trident cannot find my Kubernetes node. How do I fix this?

There are two likely scenarios why Astra 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 Astra 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's metadata is stored in CRD objects. All PVs that have been provisioned by Trident will function normally.

Upgrade Astra Trident

Can I upgrade from a older version directly to a newer version (skipping a few versions)?

NetApp supports upgrading Astra 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?

There are a number of factors to be evaluated if you want to downgrade. See [the section on downgrading](#).

Manage backends and volumes

Do I need to define both Management and Data LIFs in an ONTAP backend definition file?

NetApp recommends having both in the backend definition file. However, the Management LIF is the only one that is mandatory.

Can Astra Trident configure CHAP for ONTAP backends?

Yes. Beginning with 20.04, Astra Trident supports bidirectional CHAP for ONTAP backends. This requires setting `useCHAP=true` in your backend configuration.

How do I manage export policies with Astra Trident?

Astra 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, Astra Trident automatically manages the addition and deletion of rules for nodes with IPs within the given CIDRs. This feature requires CSI Trident.

Can we specify a port in the DataLIF?

Astra Trident 19.01 and later support specifying a port in the DataLIF. Configure it in the `backend.json` file as `"managementLIF": <ip address>:<port>`. For example, if the IP address of your management LIF is 192.0.2.1, and the port is 1000, configure `"managementLIF": "192.0.2.1:1000"`.

Can IPv6 addresses be used for the Management and Data LIFs?

Yes. Astra Trident 20.01 supports defining IPv6 addresses for the `managementLIF` and `dataLIF` parameters for ONTAP backends. You should ensure that the address follows IPv6 semantics and the `managementLIF` is defined within square brackets, (for example, `[ec0d:6504:a9c1:ae67:53d1:4bdf:ab32:e233]`). You should also ensure that Astra Trident is installed using the `--use-ipv6` flag 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 Data LIF on the backend?

No, it is not possible to update the Data LIF on the backend.

Can I create multiple backends in Astra Trident for Kubernetes?

Astra Trident can support many backends simultaneously, either with the same driver or different drivers.

How does Astra Trident store backend credentials?

Astra Trident stores the backend credentials as Kubernetes Secrets.

How does Astra 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 Astra Trident will not provision from a specific backend?

The `excludeStoragePools` parameter is used to filter the set of pools that Astra Trident will use for provisioning and will remove any pools that match.

If there are multiple backends of the same kind, how does Astra Trident select which backend to use?

If there are multiple configured backends of the same type, Astra Trident selects the appropriate backend based on the parameters present in `StorageClass` and `PersistentVolumeClaim`. For example, if there are multiple `ontap-nas` driver backends, Astra 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, Astra Trident selects from one of them at random.

Does Astra Trident support bi-directional CHAP with Element/SolidFire?

Yes.

How does Astra 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 (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 already exists that can service the new Qtree. If the FlexVol does not exist that can service the Qtree, a new FlexVol is created.

How can I set Unix permissions for volumes provisioned on ONTAP NAS?

You can set Unix permissions on the volume provisioned by Astra 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, Astra 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 Astra Trident with ONTAP?

You can set encryption on the volume provisioned by Trident by using the encryption parameter in the backend definition file.

What is the best way to implement QoS for ONTAP through Astra Trident?

Use `StorageClasses` to implement QoS for ONTAP.

How do I specify thin or thick provisioning through Astra 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 Astra Trident?

Yes. You can expand a PVC that has been created by Astra Trident. Note that volume autogrow is an ONTAP feature that is not applicable to Trident.

If I have a volume that was created outside Astra Trident can I import it into Astra Trident?

Starting in 19.04, you can use the volume import feature to bring volumes into Kubernetes.

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.
```

Can I expand iSCSI PVCs that were created by Astra Trident?

Trident 19.10 supports expanding iSCSI PVs using the CSI Provisioner.

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, Astra Trident tries to provision but errors out.

Can I create Volume Snapshots using Astra Trident?

Yes. Creating on-demand volume snapshots and Persistent Volumes from Snapshots are supported by Astra Trident. To create PVs from snapshots, ensure that the `VolumeSnapshotDataSource` feature gate has been enabled.

What are the drivers that support Astra 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`, `gcp-cvs`, and `azure-netapp-files` backend drivers.

How do I take a snapshot backup of a volume provisioned by Astra 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 level granularity and not on the qtrees level granularity. To enable the ability to snapshot volumes provisioned by Astra 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 Astra Trident.

Can I set a snapshot reserve percentage for a volume provisioned through Astra Trident?

Yes, you can reserve a specific percentage of disk space for storing the snapshot copies through Astra 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 Astra 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.

Support

Astra Trident is an officially supported NetApp project. You can reach out to NetApp using any of the standard mechanisms and get the enterprise grade support that you need.

There is also a vibrant public community of container users (including Astra Trident developers) on our [Astra Discord channel](#). This is a great place to ask general questions about the project and discuss related topics with like-minded peers.

Troubleshooting

Use the pointers provided here for troubleshooting issues you might encounter while installing and using Astra Trident.



For help with Astra Trident, create a support bundle using `tridentctl logs -a -n trident` and send it to NetApp Support <Getting Help>.



For a comprehensive list of troubleshooting articles, see the [NetApp Knowledgebase \(login required\)](#). You can also find information about troubleshooting issues related to Astra [here](#).

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 Astra 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 RedHat CoreOS, 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, see [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 are upgrading your Kubernetes cluster and/or Trident to use beta Volume Snapshots, ensure that all the existing alpha snapshot CRs are completely removed. You can then use the `tridentctl oblivate alpha-snapshot-crd` command to delete alpha snapshot CRDs. See [this blog](#) to understand the steps involved in migrating alpha snapshots.
- 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 are looking 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. Only consider a downgrade if there are no new PVs created and if no changes have been made to already existing PVs/backends/ storage classes. Since Trident now uses CRDs for maintaining state, all storage entities created (backends, storage classes, PVs and Volume Snapshots) have associated CRD objects `<Kubernetes CustomResourceDefinition Objects>` instead of data written into the PV that was used by the earlier installed version of Trident. **Newly created PVs will not be usable when moving back to an earlier version. Changes made to objects, such as backends, PVs, storage classes, and volume snapshots (created/updated/deleted) will not be visible to Trident when downgraded.** The PV that was used by the earlier version of Trident installed will still be visible to Trident. Going back to an earlier version will not disrupt access for PVs that were already created using the older release, unless they have been upgraded.
- To completely remove Trident, run the `tridentctl oblivate crd` command. This will remove all CRD objects and undefine the CRDs. Trident will no longer manage any PVs it had already provisioned.



Trident will need to be reconfigured from scratch after this.

- 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.

Troubleshooting an 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.

Troubleshooting an 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.
```

Best practices and recommendations

Deployment

Use the recommendations listed here when you deploy Astra 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. Astra 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 NetApp's portfolio has unique capabilities that benefit applications, containerized or not. 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. See [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 FlexVolumes. 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.**

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.

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. Bidirectional CHAP support is available with Trident 20.04 and above.

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, see [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, see [ONTAP 9.8 QoS commands](#).

Limit storage resource access to Kubernetes cluster members

Limiting access to the NFS volumes and iSCSI 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 not schedulable for 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 starting from the 20.04 release. 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 data LIF 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. See [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 Guide](#) (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 Astra Trident

To integrate Astra Trident, the following design and architectural elements require integration: driver selection and deployment, storage class design, virtual storage pool design, Persistent Volume Claim (PVC) impacts on storage provisioning, volume operations, and OpenShift services deployment using Astra Trident.

Driver selection and deployment

Choose a backend driver for ONTAP

Four different backend drivers are available for ONTAP systems. These drivers are differentiated by the protocol being used and how the volumes are provisioned on the storage system. Therefore, give careful consideration regarding 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 five drivers for ONTAP backends are listed below:

- `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 Astra 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
<code>ontap-nas</code>	Yes	Yes	Yes [5]	Yes	Yes [1]	Yes	Yes [1]
<code>ontap-nas-economy</code>	Yes [3]	Yes [3]	Yes [5]	Yes	Yes [3]	Yes	Yes [3]
<code>ontap-nas-flexgroup</code>	Yes [1]	No	Yes [5]	Yes	Yes [1]	Yes	Yes [1]

Astra 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
<code>ontap-san</code>	Yes	Yes	Yes [4]	Yes	Yes [1]	Yes	Yes [1]
<code>ontap-san-economy</code>	Yes	Yes	Yes [4]	Yes	Yes [3]	Yes [1]	Yes [3]

Footnote for the above tables:

Yes [1]: Not managed by Astra Trident

Yes [2]: Managed by Astra Trident, but not PV granular

Yes [3]: Not managed by Astra Trident and not PV granular

Yes [4]: Supported for raw-block volumes

Yes [5]: Supported by CSI 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.

Choose a backend driver for Cloud Volumes ONTAP

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.

Choose a backend driver for Amazon FSx for ONTAP

Amazon FSx for ONTAP enables customers to leverage NetApp features, performance, and administrative capabilities they're familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx for ONTAP supports many of ONTAP's 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`.

Choose a backend driver for NetApp HCI/SolidFire

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 Astra Trident

Yes [2]: Supported for raw-block volumes

Choose a backend driver for Azure NetApp Files

Astra 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 [Astra 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 Astra Trident

Choose a backend driver for Cloud Volumes Service with GCP

Astra Trident uses the `gcp-cvs` driver to link with the Cloud Volumes Service on the GCP backend. To configure the GCP backend on Trident, you are required specify `projectNumber`, `apiRegion`, and `apiKey` in the backend file. The project number may be found in the GCP web portal, while the API key must be taken from the service account private key file that you created while setting up API access for Cloud Volumes on GCP. Astra Trident can create CVS volumes in one of two [service types](#):

1. **CVS:** The base CVS service type, which provides high zonal availability with limited/moderate performance levels.
2. **CVS-Performance:** Performance-optimized service type best suited for production workloads that value performance. Choose from three unique service levels [`standard`, `premium`, and `extreme`]. Currently, 100 GiB is the minimum CVS-Performance volume size that will be provisioned, while CVS volumes must be at least 300 GiB. Future releases of CVS may remove this restriction.



When deploying backends using the default CVS service type [`storageClass=software`], users **must obtain access** to the sub-1TiB volumes feature on GCP for the Project Number(s) and Project ID(s) in question. This is necessary for Trident to provision sub-1TiB volumes. If not, volume creations **will fail** for PVCs that are <600 GiB. Use [this form](#) to obtain access to sub-1TiB volumes.

CVS for GCP Driver	Snapshots	Clones	Multi-attach	QoS	Expand	Replication
gcp-cvs	Yes	Yes	Yes	Yes	Yes	Yes [1]

Footnote:

Yes [1]: Not managed by Astra Trident

The `gcp-cvs` driver uses virtual storage pools. Virtual storage pools abstract the backend, letting Astra Trident decide volume placement. The administrator defines the virtual storage pools in the `backend.json` file(s). Storage classes identify the virtual storage pools with the use of labels.

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.

Storage class design for 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 Astra Trident will 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.

Storage class design to 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.

Storage class design to 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.

Storage class design for Virtual Storage Pools

Virtual Storage Pools are available for all Astra Trident backends. You can define Virtual Storage Pools for any

backend, using any driver that Astra Trident provides.

Virtual Storage 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, Astra Trident chooses a backend which matches all the selector labels to place the volume. If the Storage Class selector labels matches multiple Storage Pools, Astra Trident will choose one of them to provision the volume from.

Virtual Storage 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 Storage Pools, this issue has been alleviated. Virtual Storage Pools 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 Storage Pools can be defined for all supported NetApp backends with Astra Trident. That list includes SolidFire/NetApp HCI, ONTAP, Cloud Volumes Service on GCP, as well as Azure NetApp Files.



When defining Virtual Storage 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.

Design Virtual Storage Pools for emulating different service levels/QoS

It is possible to design Virtual Storage 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 ANF 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 Storage Pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pool(s) may be used to host a volume.

Design Virtual Pools for assigning specific set of aspects

Multiple Virtual Storage 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 Storage Pools. The volumes that get provisioned on the backend will have the aspects defined in the chosen Virtual Storage Pool.

PVC characteristics which affect storage provisioning

Some parameters beyond the requested storage class may affect Astra Trident's 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.

Astra 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 or iSCSI PVs at this time. Astra Trident supports expanding both NFS and iSCSI volumes.

The connection details of the PV cannot be modified after creation.

Create on-demand volume snapshots

Astra 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

Astra 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. Astra 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 Astra Trident or the Kubernetes cluster, as long as the destination aggregate is one which the SVM that Astra 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 Astra Trident. This will trigger Astra Trident to reinventory the SVM so that the new aggregate is recognized.

However, moving volumes across backends is not supported automatically by Astra 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 Astra Trident).

If a volume is copied to another location, the volume import feature may be used to import current volumes into Astra Trident.

Expand volumes

Astra Trident supports resizing NFS and iSCSI 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, SolidFire/NetApp HCI and Cloud Volumes Service 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`, `azure-netapp-files`, and `gcp-cvs` 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 Astra Trident. The PVC YAML or JSON file used in the import volume command points to a storage class which identifies Astra 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` or `gcp-cvs` 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 Astra Trident. This ensures a unique volume reference.

When the above command is executed, Astra 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. Astra 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.

Deploy OpenShift services

The OpenShift value-add cluster services provide important functionality to cluster administrators and the applications being hosted. The storage which these services use can be provisioned using the node-local resources, however, this often limits the capacity, performance, recoverability, and sustainability of the service. Leveraging an enterprise storage array to provide the capacity to these services can enable dramatically improved service, however, as with all applications, the OpenShift and storage administrators should work closely together to determine the best options for each. The Red Hat documentation should be leveraged heavily to determine the requirements and ensure that sizing and performance needs are met.

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. ONTAP's 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 in order 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 only contain the variables which are relevant for storage configuration as it relates to the logging service. You can find other options in [RedHat 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
<code>openshift_logging_storage_kind</code>	Set to <code>nfs</code> to have the installer create an NFS PV for the logging service.
<code>openshift_logging_storage_host</code>	The hostname or IP address of the NFS host. This should be set to the data LIF for your virtual machine.
<code>openshift_logging_storage_nfs_directory</code>	The mount path for the NFS export. For example, if the volume is junctioned as <code>/openshift_logging</code> , you would use that path for this variable.
<code>openshift_logging_storage_volume_name</code>	The name, e.g. <code>pv_ose_logs</code> , of the PV to create.
<code>openshift_logging_storage_volume_size</code>	The size of the NFS export, for example <code>100Gi</code> .

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
<code>openshift_logging_es_pvc_dynamic</code>	Set to <code>true</code> to use dynamically provisioned volumes.
<code>openshift_logging_es_pvc_storage_class_name</code>	The name of the storage class which will be used in the PVC.
<code>openshift_logging_es_pvc_size</code>	The size of the volume requested in the PVC.
<code>openshift_logging_es_pvc_prefix</code>	A prefix for the PVCs used by the logging service.
<code>openshift_logging_es_ops_pvc_dynamic</code>	Set to <code>true</code> to use dynamically provisioned volumes for the ops logging instance.
<code>openshift_logging_es_ops_pvc_storage_class_name</code>	The name of the storage class for the ops logging instance.
<code>openshift_logging_es_ops_pvc_size</code>	The size of the volume request for the ops instance.
<code>openshift_logging_es_ops_pvc_prefix</code>	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 [RedHat 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 autoscale 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 it's 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
<code>openshift_metrics_storage_kind</code>	Set to <code>nfs</code> to have the installer create an NFS PV for the logging service.
<code>openshift_metrics_storage_host</code>	The hostname or IP address of the NFS host. This should be set to the data LIF for your SVM.

Variable	Details
<code>openshift_metrics_storage_nfs_directory</code>	The mount path for the NFS export. For example, if the volume is junctioned as <code>/openshift_metrics</code> , you would use that path for this variable.
<code>openshift_metrics_storage_volume_name</code>	The name, e.g. <code>pv_ose_metrics</code> , of the PV to create.
<code>openshift_metrics_storage_volume_size</code>	The size of the NFS export, for example <code>100Gi</code> .

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
<code>openshift_metrics_cassandra_pvc_prefix</code>	A prefix to use for the metrics PVCs.
<code>openshift_metrics_cassandra_pvc_size</code>	The size of the volumes to request.
<code>openshift_metrics_cassandra_storage_type</code>	The type of storage to use for metrics, this must be set to <code>dynamic</code> for Ansible to create PVCs with the appropriate storage class.
<code>openshift_metrics_cassandra_pvc_storage_class_name</code>	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 will create any PVCs which are needed and, after Astra 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 [RedHat's OpenShift deployment guide](#) for your version so that it is configured for your environment.

Data protection

Learn about data protection and recoverability options that NetApp's storage platforms provide. Astra Trident can provision volumes that can take advantage of some of these features. You should have a data protection and recovery strategy for each application with a persistence requirement.

Back up the etcd cluster data

Astra Trident stores its metadata in the Kubernetes cluster's `etcd` database. Periodically backing up the `etcd` cluster data is important to recover Kubernetes clusters under disaster scenarios.

Steps

1. The `etcdctl snapshot save` command enables you to take a point-in-time snapshot of the `etcd` cluster:

```
sudo docker run --rm -v /backup:/backup \
  --network host \
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \
  --env ETCDCTL_API=3 \
  registry.k8s.io/etcd-amd64:3.2.18 \
  etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  snapshot save /backup/etcd-snapshot.db
```

This command creates an etcd snapshot by spinning up an etcd container and saves it in the `/backup` directory.

2. In the event of a disaster, you can spin up a Kubernetes cluster by using the etcd snapshots. Use the `etcdctl snapshot restore` command to restore a specific snapshot taken to the `/var/lib/etcd` folder. After restoring, confirm if the `/var/lib/etcd` folder has been populated with the `member` folder. The following is an example of `etcdctl snapshot restore` command:

```
# etcdctl snapshot restore '/backup/etcd-snapshot-latest.db' ; mv
/default.etcd/member/ /var/lib/etcd/
```

3. Before you initialize the Kubernetes cluster, copy all the necessary certificates.
4. Create the cluster with the `--ignore-preflight-errors=DirAvailable--var-lib-etcd` flag.
5. After the cluster comes up ensure that the kube-system pods have started.
6. Use the `kubectl get crd` command to verify if the custom resources created by Trident are present and retrieve Trident objects to make sure that all the data is available.

Recover data by using ONTAP snapshots

Snapshots play an important role by providing point-in-time recovery options for application data. However, snapshots are not backups by themselves, they do not protect against storage system failure or other catastrophes. But, they are a convenient, quick, and easy way to recover data in most scenarios. Learn about how you can use ONTAP snapshot technology to take backups of the volume and how to restore them.

- If the snapshot policy has not been defined in the backend, it defaults to using the `none` policy. This results in ONTAP taking no automatic snapshots. However, the storage administrator can take manual snapshots or change the snapshot policy via the ONTAP management interface. This does not affect Trident operation.
- The snapshot directory is hidden by default. This helps facilitate maximum compatibility of volumes provisioned using the `ontap-nas` and `ontap-nas-economy` drivers. Enable the `.snapshot` directory when using the `ontap-nas` and `ontap-nas-economy` drivers to allow applications to recover data from snapshots directly.
- Restore a volume to a state recorded in a prior snapshot by using the `volume snapshot restore` ONTAP CLI command. When you restore a snapshot copy, the restore operation overwrites the existing volume configuration. Any changes made to the data in the volume after the Snapshot copy was created

are lost.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot  
vol3_snap_archive
```

Replicate data by using ONTAP

Replicating data can play an important role in protecting against data loss due to storage array failure.



To learn more about ONTAP replication technologies, see the [ONTAP documentation](#).

SnapMirror Storage Virtual Machines (SVM) replication

You can use [SnapMirror](#) to replicate a complete SVM, which includes its configuration settings and its volumes. 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 the systems are restored.

Astra Trident cannot configure replication relationships itself, so the storage administrator can use ONTAP's SnapMirror SVM Replication feature to automatically replicate volumes to a Disaster Recovery (DR) destination.

Consider the following if you are planning to use the SnapMirror SVM Replication feature or are currently using the feature:

- You should create a distinct backend for each SVM, which has SVM-DR enabled.
- You should configure the storage classes so as to not select the replicated backends except when desired. This is important to avoid having volumes which do not need the protection of a replication relationship to be provisioned onto the backend(s) that support SVM-DR.
- Application administrators should understand the additional cost and complexity associated with replicating the data and a plan for recovery should be determined before they leverage data replication.
- Before activating the SnapMirror destination SVM, stop all the scheduled SnapMirror transfers, abort all ongoing SnapMirror transfers, break the replication relationship, stop the source SVM, and then start the SnapMirror destination SVM.
- Astra Trident does not automatically detect SVM failures. Therefore, upon a failure, the administrator should run the `tridentctl backend update` command to trigger Trident's failover to the new backend.

Here is an overview of the SVM setup steps:

- Set up peering between the source and destination cluster and SVM.
- Create the destination SVM by using the `-subtype dp-destination` option.
- Create a replication job schedule to ensure that replication happens at the required intervals.
- Create a SnapMirror replication from the destination SVM to the source SVM by using the `-identity -preserve true` option to ensure that the source SVM configurations and source SVM interfaces are copied to the destination. From the destination SVM, initialize the SnapMirror SVM replication relationship.



Disaster recovery workflow for Trident

Astra Trident 19.07 and later use Kubernetes CRDs to store and manage its own state. It uses the Kubernetes cluster's `etcd` to store its metadata. Here we assume that the Kubernetes `etcd` data files and the certificates are stored on NetApp FlexVolume. This FlexVolume resides in a SVM, which has a SnapMirror SVM-DR relationship with a destination SVM at the secondary site.

The following steps describe how to recover a single master Kubernetes cluster with Astra Trident in the event of a disaster:

1. If the source SVM fails, activate the SnapMirror destination SVM. To do this, you should stop scheduled SnapMirror transfers, abort ongoing SnapMirror transfers, break the replication relationship, stop the source SVM, and start the destination SVM.
2. From the destination SVM, mount the volume which contains the Kubernetes `etcd` data files and certificates on to the host which will be setup as a master node.
3. Copy all the required certificates pertaining to the Kubernetes cluster under `/etc/kubernetes/pki` and the `etcd` member files under `/var/lib/etcd`.
4. Create a Kubernetes cluster by using the `kubeadm init` command with the `--ignore-preflight-errors=DirAvailable-var-lib-etcd` flag. The hostnames used for the Kubernetes nodes should be the same as the source Kubernetes cluster.
5. Run the `kubectl get crd` command to verify if all the Trident custom resources have come up and retrieve the Trident objects to verify that all the data is available.
6. Update all the required backends to reflect the new destination SVM name by running the `./tridentctl update backend <backend-name> -f <backend-json-file> -n <namespace>` command.



For application persistent volumes, when the destination SVM is activated, all the volumes provisioned by Trident start serving data. After the Kubernetes cluster is set up on the destination side by using the steps outlined above, all the deployments and pods are started and the containerized applications should run without any issues.

SnapMirror volume replication

ONTAP SnapMirror volume replication is a disaster recovery feature, which enables failover to destination storage from primary storage on a volume level. SnapMirror creates a volume replica or mirror of the primary storage on the secondary storage by syncing snapshots.

Here is an overview of the ONTAP SnapMirror volume replication setup steps:

- Set up peering between the clusters in which the volumes reside and the SVMs that serve data from the volumes.
- Create a SnapMirror policy, which controls the behavior of the relationship and specifies the configuration attributes for that relationship.
- Create a SnapMirror relationship between the destination volume and the source volume by using the `snapmirror create` [command](#) and assign the appropriate SnapMirror policy.
- After the SnapMirror relationship is created, initialize the relationship so that a baseline transfer from the source volume to the destination volume is completed.



SnapMirror volume disaster recovery workflow for Trident

The following steps describe how to recover a single master Kubernetes cluster with Astra Trident.

1. In the event of a disaster, stop all the scheduled SnapMirror transfers and abort all ongoing SnapMirror transfers. Break the replication relationship between the destination and source volumes so that the destination volume becomes read/write.
2. From the destination SVM, mount the volume that contains the Kubernetes `etcd` data files and certificates on to the host, which will be set up as a master node.
3. Copy all the required certificates pertaining to the Kubernetes cluster under `/etc/kubernetes/pki` and the `etcd` member files under `/var/lib/etcd`.
4. Create a Kubernetes cluster by running the `kubeadm init` command with the `--ignore-preflight-errors=DirAvailable--var-lib-etcd` flag. The hostnames should be the same as the source Kubernetes cluster.
5. Run the `kubectl get crd` command to verify if all the Trident custom resources have come up and

retrieve Trident objects to make sure that all the data is available.

6. Clean up the previous backends and create new backends on Trident. Specify the new management and data LIF, new SVM name, and password of the destination SVM.

Disaster recovery workflow for application persistent volumes

The following steps describe how SnapMirror destination volumes can be made available for containerized workloads in the event of a disaster:

1. Stop all the scheduled SnapMirror transfers and abort all ongoing SnapMirror transfers. Break the replication relationship between the destination and source volume so that the destination volume becomes read/write. Clean up the deployments which were consuming PVC bound to volumes on the source SVM.
2. After the Kubernetes cluster is set up on the destination side by using the steps outlined above, clean up the deployments, PVCs and PV, from the Kubernetes cluster.
3. Create new backends on Trident by specifying the new management and data LIF, new SVM name and password of the destination SVM.
4. Import the required volumes as a PV bound to a new PVC by using the Trident import feature.
5. Redeploy the application deployments with the newly created PVCs.

Recover data by using Element snapshots

Back up data on an Element volume by setting a snapshot schedule for the volume and ensuring that the snapshots are taken at the required intervals. You should set the snapshot schedule by using the Element UI or APIs. Currently, it is not possible to set a snapshot schedule to a volume through the `solidfire-san` driver.

In the event of data corruption, you can choose a particular snapshot and roll back the volume to the snapshot manually by using the Element UI or APIs. This reverts any changes made to the volume since the snapshot was created.

Security

Use the recommendations listed here to ensure that your Astra Trident installation is secure.

Run Astra Trident in its own namespace

It is important to prevent applications, application administrators, users, and management applications from accessing Astra Trident object definitions or the pods to ensure reliable storage and block potential malicious activity.

To separate the other applications and users from Astra Trident, always install Astra Trident in its own Kubernetes namespace (`trident`). Putting Astra Trident in its own namespace assures that only the Kubernetes administrative personnel have access to the Astra 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 Astra Trident namespace and thus access to the `tridentctl` application.

Use CHAP authentication with ONTAP SAN backends

Astra Trident supports CHAP-based authentication for ONTAP SAN workloads (using the `ontap-san` and

ontap-san-economy drivers). NetApp recommends using bidirectional CHAP with Astra Trident for authentication between a host and the storage backend.

For ONTAP backends that use the SAN storage drivers, Astra Trident can set up bidirectional CHAP and manage CHAP usernames and secrets through `tridentctl`.

See [here](#) to understand how Astra Trident configures CHAP on ONTAP backends.



CHAP support for ONTAP backends is available with Trident 20.04 and later.

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. Astra Trident uses a secret object that includes two CHAP passwords per tenant. When Trident is installed as a CSI provisioner, it manages the CHAP secrets and stores them in a `tridentvolume` CR object for the respective PV. When you create a PV, CSI Astra 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 CSI Trident are not associated with any Volume Access Group.

In the non-CSI frontend, the attachment of volumes as devices on the worker nodes is handled by Kubernetes. After volume creation, Astra Trident makes an API call to the NetApp HCI/SolidFire system to retrieve the secrets if the secret for that tenant does not already exist. Astra Trident then passes the secrets on to Kubernetes. The kubelet located on each node accesses the secrets via the Kubernetes API and uses them to run/enable CHAP between each node accessing the volume and the NetApp HCI/SolidFire system where the volumes are located.

Reference

Astra Trident ports

Learn more about the ports that Astra Trident communicates over.

Astra Trident communicates over the following ports:

Port	Purpose
8443	Backchannel HTTPS
8001	Prometheus metrics endpoint
8000	Trident REST server
17546	Liveness/readiness probe port used by Trident daemonset pods



The liveness/readiness probe port can be changed during installation time using the `--probe-port` flag. It is important to make sure this port isn't being used by another process on the worker nodes.

Astra Trident REST API

While [tridentctl commands and options](#) is the easiest way to interact with Astra Trident's REST API, you can use the REST endpoint directly if you prefer.

This is useful for advanced installations that use Astra Trident as a standalone binary in non-Kubernetes deployments.

For better security, Astra Trident's REST API is restricted to localhost by default when running inside a pod. To change this behavior, you need to set Astra Trident's `-address` argument in its pod configuration.

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, see [tridentctl commands and options](#).
- 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, see `tridentctl` [commands and options](#).

For examples of how these APIs are called, pass the debug (`-d`) flag. For more information, see `tridentctl` [commands and options](#).

Command-line options

Astra 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 allows 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.
- `-k8s_config_path <file>`: Required; you must specify this path to a KubeConfig file.

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.

NetApp products integrated with Kubernetes

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

Astra

[Astra](#) makes it easier for enterprises to manage, protect, and move their data-rich containerized workloads running on Kubernetes within and across public clouds and on-premises. Astra provisions and provides persistent container storage using Trident from NetApp's proven and expansive storage portfolio in the public cloud and on-premises. It also offers a rich set of advanced application-aware data management functionality, such as snapshot, backup and restore, activity logs, and active cloning for data protection, disaster/data recovery, data audit, and migration use cases for Kubernetes workloads.

ONTAP

ONTAP is NetApp's 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, including engineered hardware (FAS and AFF), white-box (ONTAP Select), and cloud-only (Cloud Volumes ONTAP).



Trident supports all the above mentioned ONTAP deployment models.

Cloud Volumes ONTAP

[Cloud Volumes ONTAP](#) is a software-only storage appliance that runs the ONTAP data management software in the cloud. You can use Cloud Volumes ONTAP for production workloads, disaster recovery, DevOps, file shares, and database management. It extends enterprise storage to the cloud by offering storage efficiencies, high availability, data replication, data tiering and application consistency.

Amazon FSx for NetApp ONTAP

[Amazon FSx for NetApp ONTAP](#) is a fully managed AWS service that enables customers to launch and run file systems powered by NetApp's ONTAP storage operating system. FSx for ONTAP enables customers to leverage NetApp features, performance, and administrative capabilities they're familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx for ONTAP supports many of ONTAP's file system features and administration APIs.

Element software

[Element](#) enables the storage administrator to consolidate workloads by guaranteeing performance and enabling a simplified and streamlined storage footprint. Coupled with an API to enable automation of all aspects of storage management, Element enables storage administrators to do more with less effort.

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.

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

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 Service for Google Cloud

[NetApp Cloud Volumes Service for Google Cloud](#) is a cloud native file service that provides NAS volumes over NFS and SMB with all-flash performance. This service enables any workload, including legacy applications, to run in the GCP cloud. It provides a fully managed service which offers consistent high performance, instant cloning, data protection and secure access to Google Compute Engine (GCE) instances.

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
<code>trident.netapp.io/fileSystem</code>	<code>fileSystem</code>	ontap-san, solidfire-san, eseries-iscsi, ontap-san-economy
<code>trident.netapp.io/cloneFromPVC</code>	<code>cloneSourceVolume</code>	ontap-nas, ontap-san, solidfire-san, azure-netapp-files, gcp-cvs, ontap-san-economy
<code>trident.netapp.io/splitOnClone</code>	<code>splitOnClone</code>	ontap-nas, ontap-san
<code>trident.netapp.io/protocol</code>	<code>protocol</code>	any
<code>trident.netapp.io/exportPolicy</code>	<code>exportPolicy</code>	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
<code>trident.netapp.io/snapshotPolicy</code>	<code>snapshotPolicy</code>	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san
<code>trident.netapp.io/snapshotReserve</code>	<code>snapshotReserve</code>	ontap-nas, ontap-nas-flexgroup, ontap-san, gcp-cvs
<code>trident.netapp.io/snapshotDirectory</code>	<code>snapshotDirectory</code>	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
<code>trident.netapp.io/unixPermissions</code>	<code>unixPermissions</code>	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
<code>trident.netapp.io/blockSize</code>	<code>blockSize</code>	solidfire-san

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 (on Kubernetes 1.13 and earlier) or if your Kubernetes release does not support beta Volume Snapshots (Kubernetes 1.16 and earlier). Keep in mind that Trident 19.10 supports the CSI workflow for cloning from a PVC.



You can use the `cloneFromPVC` and `splitOnClone` annotations with CSI Trident as well as the traditional non-CSI frontend.

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:

- We recommend 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. See [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/v1beta1
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
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 & eseries-iscsi; thin: all ontap & solidfire-san
backendType	string	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san, eseries-iscsi, gcp-cvs, 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, gcp-cvs
clones	bool	true, false	Pool supports cloning volumes	Volume with clones enabled	ontap-nas, ontap-san, solidfire-san, gcp-cvs
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, etc.	The file system type for block volumes	solidfire-san, ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy, eseries-iscsi	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, gcp-cvs, azure-netapp-files	1.11+
volumeBindingMode	string	Immediate, WaitForFirstConsumer	Choose when volume binding and dynamic provisioning occurs	All	1.19 - 1.24

- 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. See [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.

- See [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/v1beta1
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.



Trident creates `VolumeSnapshotContent` objects and registers them with the Kubernetes cluster automatically based on the volumes that it provisions. You are not expected to manage them yourself.

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.

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.

Beginning with the 19.07 release, Trident uses a number of 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 upgrading from a previous version of Trident (one that used `etcd` to maintain state), the Trident installer migrates data from the `etcd` key-value data store and creates corresponding CRD objects.
- When uninstalling Trident by using the `tridentctl uninstall` command, Trident pods are deleted but the created CRDs are not cleaned up. See [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/netapp.io/trident` 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, see [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. For Cloud Volumes Service, these correspond to cloud provider regions. 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 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
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

Attribute	Type	Required	Description
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

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

Attribute	Type	Required	Description
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.

Astra Trident `ResourceQuota` object

The Trident daemonset consumes a `system-node-critical` Priority Class—the highest Priority Class available in Kubernetes—to ensure Astra 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, Astra 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, Astra 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, see [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 Astra Trident `ResourceQuota` object using the command:

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

tridentctl commands and options

The [Trident installer bundle](#) includes a command-line utility, `tridentctl`, that provides simple access to Astra Trident. Kubernetes users with sufficient privileges can use it to install Astra Trident as well as to interact with it directly to manage the namespace that contains the Astra Trident pod.

For usage information, run `tridentctl --help`.

The available commands and global options are:

```
Usage:
  tridentctl [command]
```

Available commands:

- `create`: Add a resource to Astra Trident.
- `delete`: Remove one or more resources from Astra Trident.
- `get`: Get one or more resources from Astra Trident.
- `help`: Help about any command.
- `images`: Print a table of the container images Astra Trident needs.
- `import`: Import an existing resource to Astra Trident.
- `install`: Install Astra Trident.
- `logs`: Print the logs from Astra Trident.
- `send`: Send a resource from Astra Trident.
- `uninstall`: Uninstall Astra Trident.
- `update`: Modify a resource in Astra Trident.

- **upgrade:** Upgrade a resource in Astra Trident.
- **version:** Print the version of Astra Trident.

Flags:

- **-d, --debug:** Debug output.
- **-h, --help:** Help for `tridentctl`.
- **-n, --namespace string:** Namespace of Astra Trident deployment.
- **-o, --output string:** Output format. One of `json|yaml|name|wide|ps` (default).
- **-s, --server string:** Address/port of Astra 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.



Trident REST interface can be configured to listen and serve at 127.0.0.1 (for IPv4) or `:::1` (for IPv6) only.

create

You can use run the `create` command to add a resource to Astra Trident.

Usage:

```
tridentctl create [option]
```

Available option:

backend: Add a backend to Astra Trident.

delete

You can run the `delete` command to remove one or more resources from Astra Trident.

Usage:

```
tridentctl delete [option]
```

Available options:

- **backend:** Delete one or more storage backends from Astra Trident.
- **snapshot:** Delete one or more volume snapshots from Astra Trident.
- **storageclass:** Delete one or more storage classes from Astra Trident.
- **volume:** Delete one or more storage volumes from Astra Trident.

get

You can run the `get` command to get one or more resources from Astra Trident.

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

Available options:

- `backend`: Get one or more storage backends from Astra Trident.
- `snapshot`: Get one or more snapshots from Astra Trident.
- `storageclass`: Get one or more storage classes from Astra Trident.
- `volume`: Get one or more volumes from Astra Trident.

images

You can run the `images` flag to print a table of the container images Astra Trident needs.

```
Usage:
  tridentctl images [flags]
```

Flags:

- * `-h, --help``: Help for images.
- * `-v, --k8s-version string``: Semantic version of Kubernetes cluster.

import volume

You can run the `import volume` command to import an existing volume to Astra Trident.

```
Usage:
  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

You can run the `install` flags to install Astra Trident.

Usage:

```
tridentctl install [flags]
```

Flags:

- `--autosupport-image string`: The container image for Autosupport Telemetry (default "netapp/trident autosupport:20.07.0").
- `--autosupport-proxy string`: The address/port of a proxy for sending Autosupport Telemetry.
- `--csi`: Install CSI Trident (override for Kubernetes 1.13 only, requires feature gates).
- `--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 Astra Trident logging format (text, json) (default "text").
- `--pv string`: The name of the legacy PV used by Astra Trident, makes sure this doesn't exist (default "trident").
- `--pvc string`: The name of the legacy PVC used by Astra 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 Astra Trident image to install.
- `--use-custom-yaml`: Use any existing YAML files that exist in setup directory.
- `--use-ipv6`: Use IPv6 for Astra Trident's communication.

logs

You can run the `logs` flags to print the logs from Astra Trident.

Usage:

```
tridentctl logs [flags]
```

Flags:

- ``-a, --archive`: Create a support archive with all logs unless otherwise specified.
- ``-h, --help`: Help for logs.
- ``-l, --log string`: Astra 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

You can run the `send` command to send a resource from Astra Trident.

```
Usage:
  tridentctl send [option]
```

Available option:

`autosupport`: Send an Autosupport archive to NetApp.

uninstall

You can run the `uninstall` flags to uninstall Astra Trident.

```
Usage:
  tridentctl uninstall [flags]
```

Flags:

- * `-h, --help`: Help for uninstall.
- * `--silent`: Disable most output during uninstallation.

update

You can run the `update` commands to modify a resource in Astra Trident.

```
Usage:
  tridentctl update [option]
```

Available options:

`backend`: Update a backend in Astra Trident.

upgrade

You can run the `upgrade` commands to upgrade a resource in Astra Trident.

```
Usage:
tridentctl upgrade [option]
```

Available option:

`volume`: Upgrade one or more persistent volumes from NFS/iSCSI to CSI.

version

You can run the `version` flags to print the version of `tridentctl` and the running Trident service.

```
Usage:
tridentctl version [flags]
```

Flags:

* `--client`: Client version only (no server required).

* `-h`, `--help`: Help for version.

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, Astra Trident enables certain permissions during installation. The following sections detail the permissions set by Astra Trident.



PSS replaces Pod Security Policies (PSP). PSP was deprecated in Kubernetes v1.21 and will be removed in v1.25. For more information, see [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, see Kubernetes: Mount propagation .
Host networking	Required for the iSCSI daemon. <code>iscsiadm</code> 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.

Permission	Description
Host PID	Required to start <code>rpc-statd</code> for NFS. Astra Trident queries host processes to determine if <code>rpc-statd</code> is running before mounting NFS volumes.
Capabilities	The <code>SYS_ADMIN</code> capability is provided as part of the default capabilities for privileged containers. For example, Docker sets these capabilities for privileged containers: <code>CapPrm: 0000003fffffffff</code> <code>CapEff: 0000003fffffffff</code>
Seccomp	Seccomp profile is always "Unconfined" in privileged containers; therefore, it cannot be enabled in Astra Trident.
SELinux	On OpenShift, privileged containers are run in the <code>spc_t</code> ("Super Privileged Container") domain, and unprivileged containers are run in the <code>container_t</code> domain. On <code>containerd</code> , with <code>container-selinux</code> installed, all containers are run in the <code>spc_t</code> domain, which effectively disables SELinux. Therefore, Astra Trident does not add <code>seLinuxOptions</code> 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
<code>pod-security.kubernetes.io/enforce</code>	Allows the Trident Controller and nodes to be admitted into the install namespace.	<code>enforce: privileged</code>
<code>pod-security.kubernetes.io/enforce-version</code>	Do not change the namespace label.	<code>enforce-version: <version of the current cluster or highest version of PSS tested.></code>



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
<code>allowPrivilegeEscalation</code>	Privileged containers must allow privilege escalation.	<code>true</code>

Field	Description	Default
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
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

Field	Description	Default
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 <code>seLinuxOptions</code> 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 <code>nfsd</code> .	true
allowHostNetwork	<code>iscsiadm</code> requires the host network to communicate with the iSCSI daemon.	true
allowHostPID	Host PID is required to check if <code>rpc-statd</code> 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
allowedUnsafeSysctls	Trident does not require any unsafe <code>sysctls</code> .	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

Labels	Description	Default
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 <code>seLinuxOptions</code> 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

Labels	Description	Default
volumes	Trident pods require these volume plugins.	<p>hostPath, downwardAPI, projected, emptyDir</p> <p>:leveloffset: -1</p> <p>= Earlier versions of documentation</p> <p>:hardbreaks:</p> <p>:icons: font</p> <p>:relative_path: ./</p> <p>:imagesdir: /tmp/d20220826-5327-1yxj6wq/source/./trident-reference/./media/</p> <p>[.lead]</p> <p>If you aren't running Astra Trident 22.07, the documentation for previous releases is available.</p> <p>NOTE: For releases prior to 21.07, you will be redirected to the legacy documentation site.</p> <ul style="list-style-type: none"> * Astra Trident 22.04 * Astra Trident 22.01 * Astra Trident 21.10 * Astra Trident 21.07 * Astra Trident 21.04 * Astra Trident 21.01 * Astra Trident 20.10 * Astra Trident 20.07 * Astra Trident 20.04 * Astra Trident 20.01 * Astra Trident 19.10 * Astra Trident 19.07 * Astra Trident 19.04 * Astra Trident 19.01 <p>= Legal notices</p> <p>:icons: font</p> <p>:relative_path: ./</p> <p>:imagesdir: /tmp/d20220826-5327-1yxj6wq/source/./trident-reference/./media/</p> <p>[.lead]</p> <p>Legal notices provide access to copyright statements, trademarks,</p>