Reference

Astra Trident

NetApp
November 01, 2022

This PDF was generated from https://docs.netapp.com/us-en/trident/trident-reference/trident-ports.html on November 01, 2022. Always check docs.netapp.com for the latest.

Table of Contents

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
trident.netapp.io/fileSystem	fileSystem	ontap-san, solidfire-san, eseries-iscsi, ontap-san-economy
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas, ontap-san, solidfire-san, azure-netapp-files, gcp-cvs, ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ontap-nas, ontap-san
trident.netapp.io/protocol	protocol	any
trident.netapp.io/exportPolicy	exportPolicy	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	snapshotPolicy	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san
trident.netapp.io/snapshotReserve	snapshotReserve	ontap-nas, ontap-nas-flexgroup, ontap-san, gcp-cvs
trident.netapp.io/snapshotDirectory	snapshotDirectory	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup
trident.netapp.io/blockSize	blockSize	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 lifecyle 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 deamonset 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.

`volume` flags:
* `-h, --help`: Help for volumes.
* `--parentOfSubordinate string`: Limit query to subordinate source volume.
* `--subordinateOf string`: Limit query to subordinates of volume.

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. `iscsiadm` manages iSCSI mounts and uses host networking to communicate with the iSCSI daemon.

Permission	Description
Host IPC	NFS uses interprocess communication (IPC) to communicate with the NFSD.
Host PID	Required to start `rpc-statd` for NFS. Astra Trident queries host processes to determine if `rpc-statd` is running before mounting NFS volumes.

Capabilities	The `SYS_ADMIN` capability is provided as part of the default capabilities for privileged containers. For example, Docker sets these capabilities for privileged containers:
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccomp profile is always "Unconfined" in privileged containers; therefore, it cannot be enabled in Astra Trident.
SELinux	On OpenShift, privileged containers are run in the `spc_t` ("Super Privileged Container") domain, and unprivileged containers are run in the `container_t` domain. On `containerd`, with `container-selinux` installed, all containers are run in the `spc_t` domain, which effectively disables SELinux. Therefore, Astra Trident does not add `seLinuxOptions` to containers.
DAC	Privileged containers must be run as root. Non-privileged containers run as root to access unix sockets required by CSI.

Pod Security Standards (PSS)

Label	Description	Default

`pod-security.kubernetes.io/enforce`
 	
`pod-security.kubernetes.io/enforce-version`
	Allows the Trident Controller and nodes to be admitted into the install namespace.
	
	Do not change the namespace label.
		`enforce: privileged`
		
		`enforce-version: <version of the current cluster or highest version of PSS tested.>`

Changing the namespace labels can result in pods not being scheduled, an "Error creating: …" or, "Warning: trident-csi-…". If this happens, check if the namespace label for `privileged` was changed. If so, reinstall Trident.

Pod Security Policies (PSP)

Field	Description	Default
`allowPrivilegeEscalation`	Privileged containers must allow privilege escalation.	`true`
`allowedCSIDrivers`	Trident does not use inline CSI ephemeral volumes.	Empty
`allowedCapabilities`	Non-privileged Trident containers do not require more capabilities than the default set and privileged containers are granted all possible capabilities.	Empty
`allowedFlexVolumes`	Trident does not make use of a FlexVolume driver, therefore they are not included in the list of allowed volumes.	Empty
`allowedHostPaths`	The Trident node pod mounts the node's root filesystem, therefore there is no benefit to setting this list.	Empty
`allowedProcMountTypes`	Trident does not use any `ProcMountTypes`.	Empty
`allowedUnsageSysctls`	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`

Field	Description	Default
`requiredDropCapabilities`	Trident node pods run a privileged container and cannot drop capabilities.	`none`
`runAsGroup`	Trident containers run as root.	`RunAsAny`
`runAsUser`	Trident containers run as root.	`runAsAny`
`runtimeClass`	Trident does not use `RuntimeClasses`.	Empty
`seLinux`	Trident does not set `seLinuxOptions` because there are currently differences in how container runtimes and Kubernetes distributions handle SELinux.	Empty
`supplementalGroups`	Trident containers run as root.	`RunAsAny`
`volumes`	Trident pods require these volume plugins.	`hostPath, projected, emptyDir`

Security Context Constraints (SCC)

Labels	Description	Default
`allowHostDirVolumePlugin`	Trident node pods mount the node's root filesystem.	`true`
`allowHostIPC`	Mounting NFS volumes requires host IPC to communicate with `nfsd`.	`true`
`allowHostNetwork`	iscsiadm requires the host network to communicate with the iSCSI daemon.	`true`
`allowHostPID`	Host PID is required to check if `rpc-statd` is running on the node.	`true`
`allowHostPorts`	Trident does not use any host ports.	`false`
`allowPrivilegeEscalation`	Privileged containers must allow privilege escalation.	`true`
`allowPrivilegedContainer`	Trident node pods must run a privileged container in order to mount volumes.	`true`
`allowedUnsafeSysctls`	Trident does not require any unsafe `sysctls`.	`none`

Labels	Description	Default
`allowedCapabilities`	Non-privileged Trident containers do not require more capabilities than the default set and privileged containers are granted all possible capabilities.	Empty
`defaultAddCapabilities`	No capabilities are required to be added to privileged containers.	Empty
`fsGroup`	Trident containers run as root.	`RunAsAny`
`groups`	This SCC is specific to Trident and is bound to its user.	Empty
`readOnlyRootFilesystem`	Trident node pods must write to the node filesystem.	`false`
`requiredDropCapabilities`	Trident node pods run a privileged container and cannot drop capabilities.	`none`
`runAsUser`	Trident containers run as root.	`RunAsAny`
`seLinuxContext`	Trident does not set `seLinuxOptions` because there are currently differences in how container runtimes and Kubernetes distributions handle SELinux.	Empty
`seccompProfiles`	Privileged containers always run "Unconfined".	Empty
`supplementalGroups`	Trident containers run as root.	`RunAsAny`
`users`	One entry is provided to bind this SCC to the Trident user in the Trident namespace.	n/a

Labels	Description	Default
`volumes`	Trident pods require these volume plugins.	`hostPath, downwardAPI, projected, emptyDir` :leveloffset: -1 :leveloffset: -1 <<< **Copyright information** Copyright © 2022 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner. Software derived from copyrighted NetApp material is subject to the following license and disclaimer: THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING