Get started

Astra Trident

NetApp
April 03, 2024

Table of Contents

Get started . . . 1
Learn about Astra Trident . . . 1
Quick start for Astra Trident . . . 9
Requirements . . . 10

Get started

Learn about Astra Trident

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 application's persistence demands using industry-standard interfaces, such as the Container Storage Interface (CSI).

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 built on Astra Trident. It also offers 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.

Learn more about Astra or sign up for a free trial.

What is Astra Trident?

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

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

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

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

Take the Astra Trident test drive

To take a test drive, request access to the "Easily Deploy and Clone Persistent Storage for Containerized Workloads" NetApp Test Drive using a ready-to-use lab image. The test drive provides a sandbox environment with a three-node Kubernetes cluster and Astra Trident installed and configured. This is a great way to familiarize yourself with Astra Trident and explore its features.

Another option is the kubeadm Install Guide provided by Kubernetes.

Don’t use a Kubernetes cluster that you build using these instructions in a production environment. Use the production deployment guides provided by your distribution for production-ready clusters.

For more information

- NetApp Astra product family
- Astra Control Service documentation
- Astra Control Center documentation
- Astra API documentation

Astra Trident architecture

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

Understanding controller pods and node pods

Astra Trident deploys as a single Trident Controller Pod and one or more Trident Node Pods on the Kubernetes cluster and uses standard Kubernetes *CSI Sidecar Containers* to simplify the deployment of CSI plugins. Kubernetes CSI Sidecar Containers are maintained by the Kubernetes Storage community.

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

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

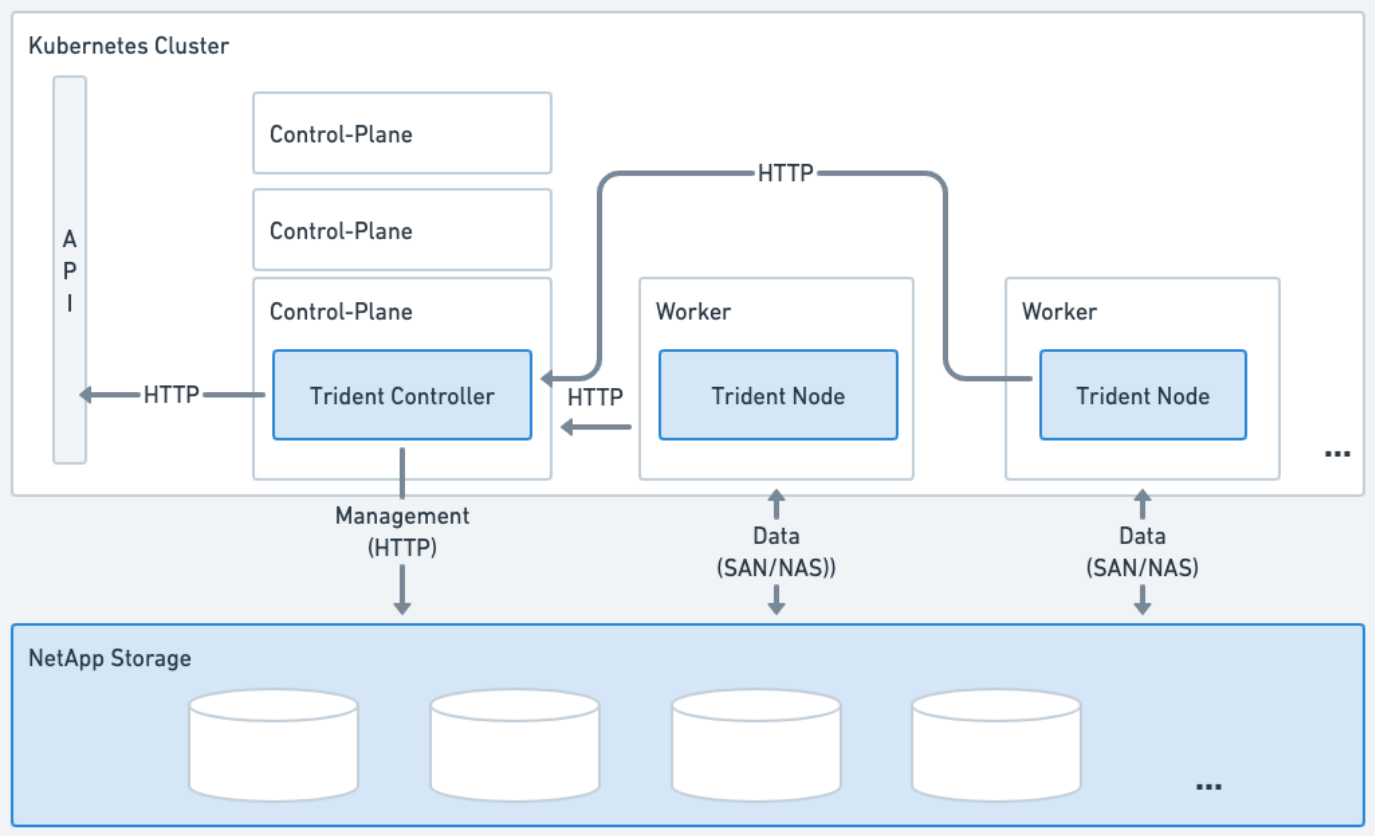


Figure 1. Astra Trident deployed on the Kubernetes cluster

Trident Controller Pod

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

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

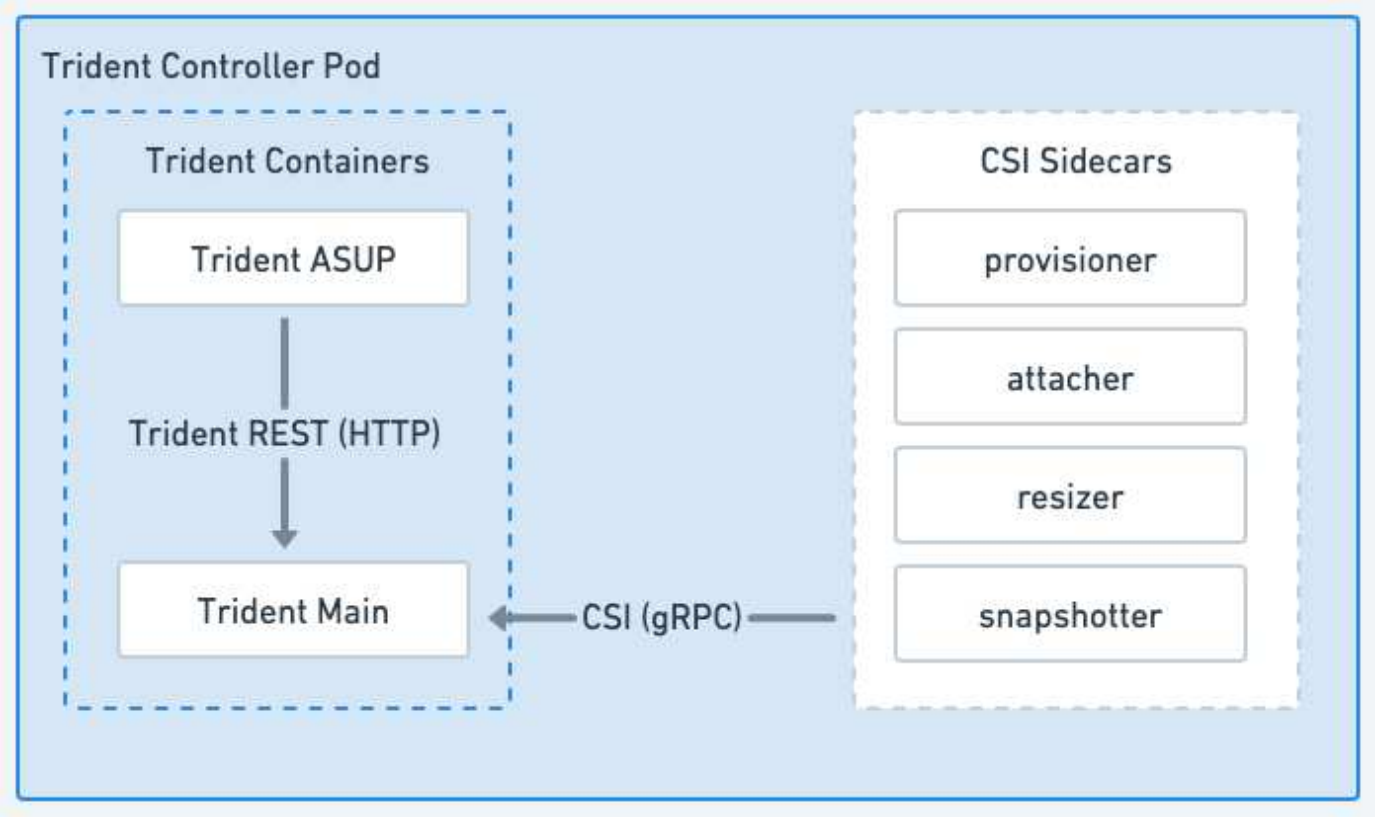


Figure 2. Trident Controller Pod diagram

Trident Node Pods

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

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

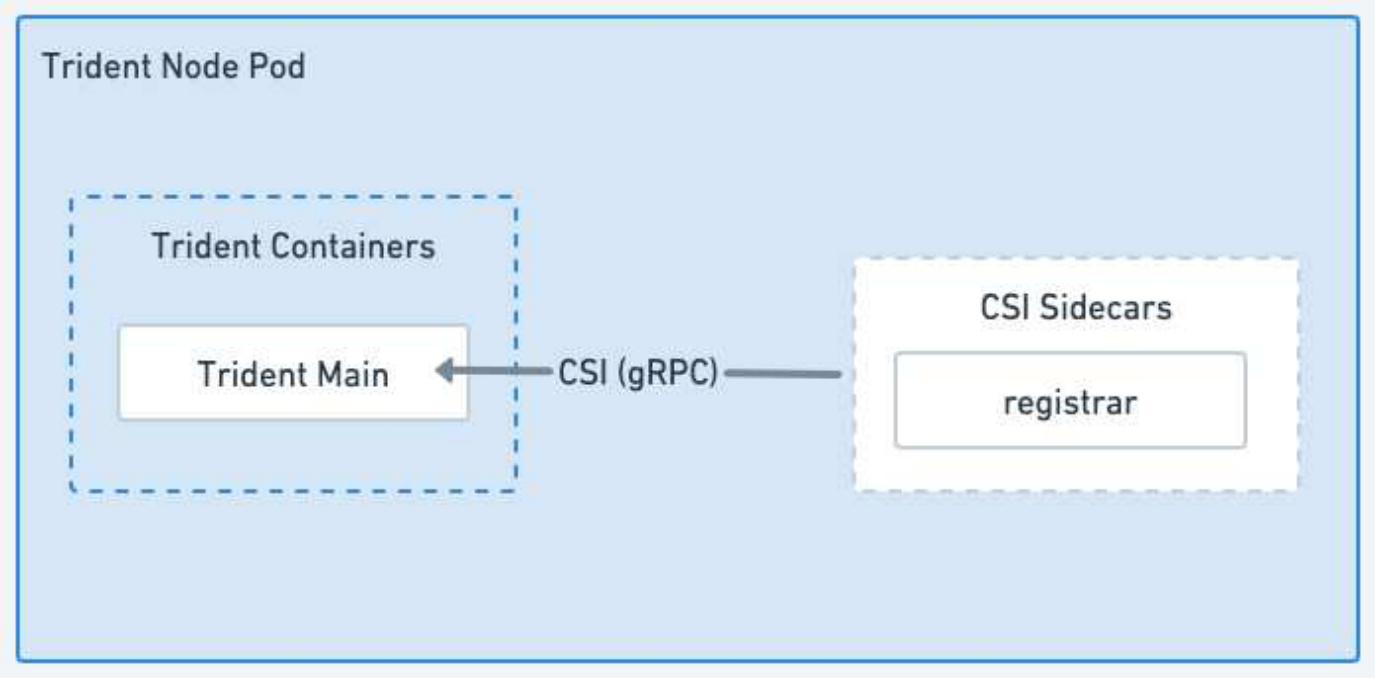


Figure 3. Trident Node Pod diagram

Supported Kubernetes cluster architectures

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

Concepts

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.

Storage class association

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

the storage class. If a storage pool's attributes and name match all of the requested attributes and pool names, 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.

Volume creation

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.

Learn about volume snapshot creation

- 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 snapshot technology delivers more stability, scalability, recoverability, and performance than competing snapshot technologies. These snapshot copies are extremely efficient both in the time needed to create them and in storage space.
- For the `ontap-nas-flexgroup` driver, each Persistent Volume (PV) maps to a FlexGroup. As a result, volume snapshots are created as NetApp FlexGroup snapshots. NetApp snapshot technology delivers more stability, scalability, recoverability, and performance than competing snapshot technologies. These snapshot copies are extremely efficient both in the time needed to create them and in storage space.
- For the `ontap-san-economy` driver, PVs map to LUNs created on shared FlexVols. VolumeSnapshots of PVs are achieved by performing FlexClones of the associated LUN. ONTAP FlexClone technology makes it possible to create copies of even the largest datasets almost instantaneously. Copies share data blocks with their parents, consuming no storage except what is required for metadata.
- For the `solidfire-san` driver, each PV maps to a LUN created on the NetApp Element software/NetApp HCI cluster. VolumeSnapshots are represented by Element snapshots of the underlying LUN. These snapshots are point-in-time copies and only take up a small amount of system resources and space.
- When working with the `ontap-nas` and `ontap-san` drivers, ONTAP snapshots are point-in-time copies of the FlexVol and consume space on the FlexVol itself. This can result in the amount of writable space in the volume to reduce with time as snapshots are created/scheduled. One simple way of addressing this is to grow the volume by resizing through Kubernetes. Another option is to delete snapshots that are no longer required. When a VolumeSnapshot created through Kubernetes is deleted, 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

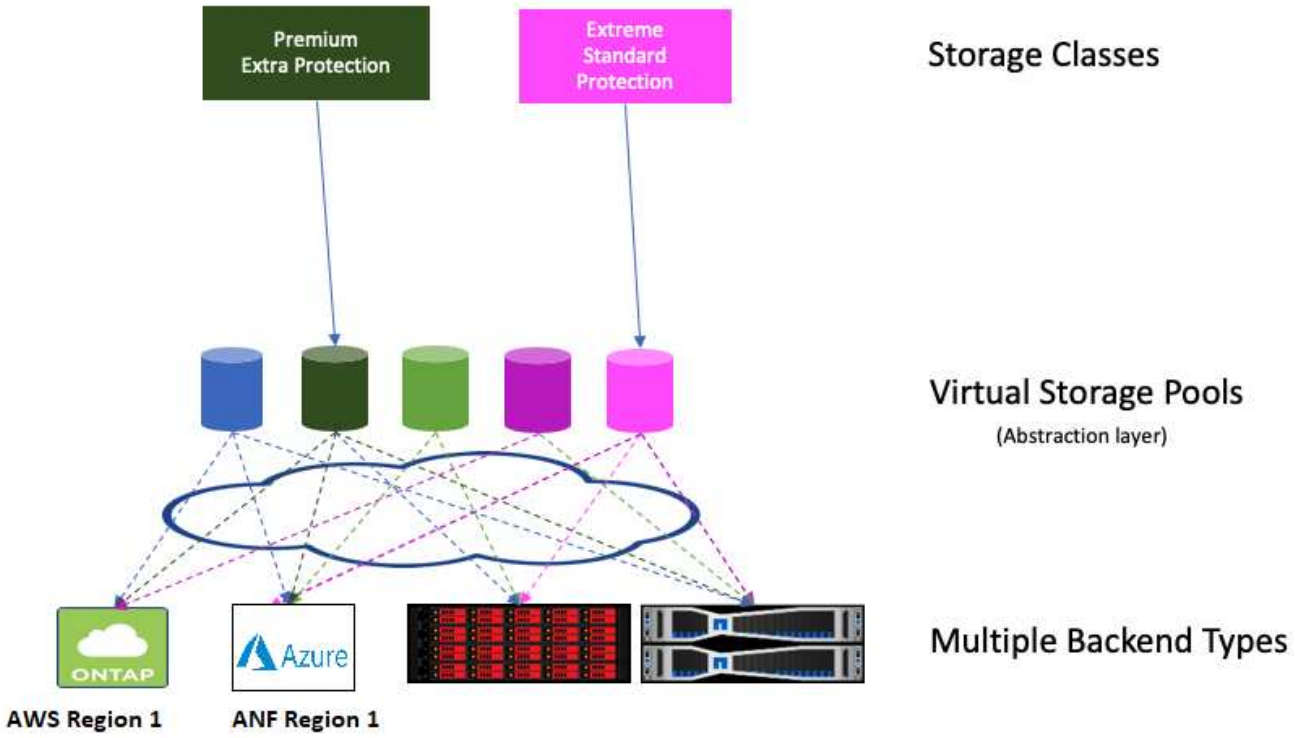
snapshots is performed by using the FlexClone technology for supported ONTAP and CVS backends. When creating a PV from a snapshot, the backing volume is a FlexClone of the snapshot's parent volume. The `solidfire-san` driver uses Element software volume clones to create PVs from snapshots. Here it creates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

Virtual pools

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

Learn about virtual pools

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

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

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

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

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

Volume access groups

Learn more about how 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.

Learn about volume access groups

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 groups, it does not create or otherwise manage access groups themselves. The access groups 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.

Quick start for Astra Trident

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

> For Docker, refer to Astra Trident for Docker.

1 Install Astra Trident

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

Install Astra Trident

2 Prepare the worker node

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

Prepare the worker node

3 Create a backend

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.

Configure a backend for your storage system

4 Create a Kubernetes StorageClass

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

Create a storage class

5 Provision a volume

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

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

Provision a volume

What's next?

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

Requirements

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

Critical information about Astra Trident

You must read the following critical information about Astra Trident.

Critical information about Astra Trident

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

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

Supported frontends (orchestrators)

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

- Anthos On-Prem (VMware) and Anthos on bare metal 1.12
- Kubernetes 1.22 - 1.27
- Mirantis Kubernetes Engine 3.5
- OpenShift 4.10 - 4.13

The Trident operator is supported with these releases:

- Anthos On-Prem (VMware) and Anthos on bare metal 1.12
- Kubernetes 1.22 - 1.27
- OpenShift 4.10 - 4.13

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.

Before upgrading a Kubernetes cluster from 1.24 to 1.25 or later that has Astra Trident installed, see Upgrade a Helm installation.

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
- Cloud Volumes ONTAP
- Cloud Volumes Service for GCP
- FAS/AFF/Select 9.5 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?
Astra Trident	1.22 - 1.27	No
Volume Snapshots	1.22 - 1.27	No
PVC from Volume Snapshots	1.22 - 1.27	No
iSCSI PV resize	1.22 - 1.27	No
ONTAP Bidirectional CHAP	1.22 - 1.27	No
Dynamic Export Policies	1.22 - 1.27	No
Trident Operator	1.22 - 1.27	No
CSI Topology	1.22 - 1.27	No

Tested host operating systems

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

- RedHat CoreOS (RHCOS) versions as supported by OpenShift Container Platform (AMD64 and ARM64)
- RHEL 8+ (AMD64 and ARM64)
- Ubuntu 22.04 or later (AMD64 and ARM64)
- Windows Server 2019 (AMD64)

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.

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

Host configuration

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

Prepare the worker node

Storage system configuration

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

Configure backends

Astra Trident ports

Astra Trident requires access to specific ports for communication.

Astra Trident ports

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
v1.22.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)

Kubernetes version	Container image
v1.23.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)
v1.24.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)
v1.25.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)

Kubernetes version	Container image
v1.26.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)
v1.27.0	• docker.io/netapp/trident:23.07.1 • docker.io/netapp/trident-autosupport:23.07 • registry.k8s.io/sig-storage/csi-provisioner:v3.5.0 • registry.k8s.io/sig-storage/csi-attacher:v4.3.0 • registry.k8s.io/sig-storage/csi-resizer:v1.8.0 • registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.8.0 • docker.io/netapp/trident-operator:23.07.1 (optional)

Copyright information

Trademark information