



# **Best practices and recommendations**

## **Astra Trident**

NetApp

February 09, 2023

This PDF was generated from <https://docs.netapp.com/us-en/trident/trident-reco/deploy-reco.html> on February 09, 2023. Always check docs.netapp.com for the latest.

# Table of Contents

- Best practices and recommendations ..... 1
  - Deployment ..... 1
  - Storage configuration ..... 1
  - Integrate Astra Trident ..... 8
  - Data protection ..... 18
  - Security ..... 23

# 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 the NetApp portfolio has unique capabilities that benefit applications, containerized or not.

### Platform overview

Trident works with ONTAP and Element. There is not one platform which is better suited for all applications and scenarios than another, however, the needs of the application and the team administering the device should be taken into account when choosing a platform.

You should follow the baseline best practices for the host operating system with the protocol that you are leveraging. Optionally, you might want to consider incorporating application best practices, when available, with backend, storage class, and PVC settings to optimize storage for specific applications.

### ONTAP and Cloud Volumes ONTAP best practices

Learn the best practices for configuring ONTAP and Cloud Volumes ONTAP for Trident.

The following recommendations are guidelines for configuring ONTAP for containerized workloads, which consume volumes that are dynamically provisioned by Trident. Each should be considered and evaluated for appropriateness in your environment.

## Use SVM(s) dedicated to Trident

Storage Virtual Machines (SVMs) provide isolation and administrative separation between tenants on an ONTAP system. Dedicating an SVM to applications enables the delegation of privileges and enables applying best practices for limiting resource consumption.

There are several options available for the management of the SVM:

- Provide the cluster management interface in the backend configuration, along with appropriate credentials, and specify the SVM name.
- Create a dedicated management interface for the SVM by using ONTAP System Manager or the CLI.
- Share the management role with an NFS data interface.

In each case, the interface should be in DNS, and the DNS name should be used when configuring Trident. This helps to facilitate some DR scenarios, for example, SVM-DR without the use of network identity retention.

There is no preference between having a dedicated or shared management LIF for the SVM, however, you should ensure that your network security policies align with the approach you choose. Regardless, the management LIF should be accessible via DNS to facilitate maximum flexibility should [SVM-DR](#) be used in conjunction with Trident.

## Limit the maximum volume count

ONTAP storage systems have a maximum volume count, which varies based on the software version and hardware platform. 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 unable to schedule user applications, separate export policies should be used to further limit access to storage resources. This includes creating an export policy for services which are deployed to those infrastructure nodes (for example, the OpenShift Metrics and Logging services), and standard applications which are deployed to non-infrastructure nodes.

### Use a dedicated export policy

You should ensure that an export policy exists for each backend that only allows access to the nodes present in the Kubernetes cluster. Trident can automatically create and manage export policies 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 pool design, Persistent Volume Claim (PVC) impacts on storage provisioning, volume operations, and OpenShift services deployment using Astra Trident.

### Driver selection and deployment

Select and deploy a backend driver for your storage system.

#### ONTAP backend drivers

ONTAP backend drivers are differentiated by the protocol used and how the volumes are provisioned on the storage system. Therefore, give careful consideration when deciding which driver to deploy.

At a higher level, if your application has components which need shared storage (multiple pods accessing the same PVC), NAS-based drivers would be the default choice, while the block-based iSCSI drivers meet the needs of non-shared storage. Choose the protocol based on the requirements of the application and the comfort level of the storage and infrastructure teams. Generally speaking, there is little difference between them for most applications, so often the decision is based upon whether or not shared storage (where more than one pod will need simultaneous access) is needed.

The available ONTAP backend drivers are:

- `ontap-nas`: Each PV provisioned is a full ONTAP FlexVolume.
- `ontap-nas-economy`: Each PV provisioned is a qtree, with a configurable number of qtrees per FlexVolume (default is 200).
- `ontap-nas-flexgroup`: Each PV provisioned as a full ONTAP FlexGroup, and all aggregates assigned to a SVM are used.
- `ontap-san`: Each PV provisioned is a LUN within its own FlexVolume.
- `ontap-san-economy`: Each PV provisioned is a LUN, with a configurable number of LUNs per FlexVolume (default is 100).

Choosing between the three NAS drivers has some ramifications to the features, which are made available to the application.

Note that, in the tables below, not all of the capabilities are exposed through 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	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.

### Cloud Volumes ONTAP backend drivers

Cloud Volumes ONTAP provides data control along with enterprise-class storage features for various use cases, including file shares and block-level storage serving NAS and SAN protocols (NFS, SMB / CIFS, and iSCSI). The compatible drivers for Cloud Volume ONTAP are `ontap-nas`, `ontap-nas-economy`, `ontap-san` and `ontap-san-economy`. These are applicable for Cloud Volume ONTAP for Azure, Cloud Volume ONTAP for GCP.

### Amazon FSx for ONTAP backend drivers

Amazon FSx for 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`.

### NetApp HCI/SolidFire backend drivers

The `solidfire-san` driver used with the NetApp HCI/SolidFire platforms, helps the admin configure an Element backend for Trident on the basis of QoS limits. If you would like to design your backend to set the specific QoS limits on the volumes provisioned by Trident, use the `type` parameter in the backend file. The admin also can restrict the volume size that could be created on the storage using the `limitVolumeSize` parameter. Currently, Element storage features like volume resize and volume replication are not supported through the `solidfire-san` driver. These operations should be done manually through Element Software web UI.

SolidFire Driver	Snapshots	Clones	Multi-attach	CHAP	QoS	Resize	Replication
solidfire-san	Yes	Yes	Yes [2]	Yes	Yes	Yes	Yes [1]

Footnote:

Yes [1]: Not managed by Astra Trident

Yes [2]: Supported for raw-block volumes

### Azure NetApp Files backend drivers

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

### Cloud Volumes Service on Google Cloud backend driver

Astra Trident uses the `gcp-cvs` driver to link with the Cloud Volumes Service on Google Cloud.

The `gcp-cvs` driver uses virtual pools to abstract the backend and allow Astra Trident to determine volume placement. The administrator defines the virtual pools in the `backend.json` files. Storage classes use selectors to identify virtual pools by label.

- If virtual pools are defined in the backend, Astra Trident will try to create a volume in the Google Cloud storage pools to which those virtual pools are limited.
- If virtual pools are not defined in the backend, Astra Trident will select a Google Cloud storage pool from the available storage pools in the region.

To configure the Google Cloud backend on Astra Trident, you must specify `projectNumber`, `apiRegion`, and `apiKey` in the backend file. You can find the project number in the Google Cloud console. The API key is taken from the service account private key file you created when setting up API access for Cloud Volumes Service on Google Cloud.

For details on Cloud Volumes Service on Google Cloud service types and service levels, see [Learn about Astra Trident support for CVS for GCP](#).

Cloud Volumes Service for Google Cloud driver	Snapshots	Clones	Multi-attach	QoS	Expand	Replication
gcp-cvs	Yes	Yes	Yes	Yes	Yes	Available on CVS-Performance service type only.



#### Replication notes

- Replication is not managed by Astra Trident.
- The clone will be created in the same storage pool as the source volume.

## Storage class design

Individual Storage classes need to be configured and applied to create a Kubernetes Storage Class object. This section discusses how to design a storage class for your application.

### Specific backend utilization

Filtering can be used within a specific storage class object to determine which storage pool or set of pools are to be used with that specific storage class. Three sets of filters can be set in the Storage Class: `storagePools`, `additionalStoragePools`, and/or `excludeStoragePools`.

The `storagePools` parameter helps restrict storage to the set of pools that match any specified attributes. The `additionalStoragePools` parameter is used to extend the set of pools that 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.

### Emulate QoS policies

If you would like to design Storage Classes to emulate Quality of Service policies, create a Storage Class with the `media` attribute as `hdd` or `ssd`. Based on the `media` attribute mentioned in the storage class, Trident will select the appropriate backend that serves `hdd` or `ssd` aggregates to match the `media` attribute and then direct the provisioning of the volumes on to the specific aggregate. Therefore we can create a storage class `PREMIUM` which would have `media` attribute set as `ssd` which could be classified as the `PREMIUM` QoS policy. We can create another storage class `STANDARD` which would have the `media` attribute set as `'hdd'` which could be classified as the `STANDARD` QoS policy. We could also use the `"IOPS"` attribute in the storage class to redirect provisioning to an Element appliance which can be defined as a QoS Policy.

### Utilize backend based on specific features

Storage classes can be designed to direct volume provisioning on a specific backend where features such as thin and thick provisioning, snapshots, clones, and encryption are enabled. To specify which storage to use, create Storage Classes that specify the appropriate backend with the required feature enabled.

## Virtual pools

Virtual pools are available for all Astra Trident backends. You can define virtual pools for any backend, using any driver that Astra Trident provides.

Virtual pools allow an administrator to create a level of abstraction over backends which can be referenced through Storage Classes, for greater flexibility and efficient placement of volumes on backends. Different backends can be defined with the same class of service. Moreover, multiple storage pools can be created on the same backend but with different characteristics. When a Storage Class is configured with a selector with the specific labels, 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 pool design

While creating a backend, you can generally specify a set of parameters. It was impossible for the administrator to create another backend with the same storage credentials and with a different set of parameters. With the introduction of virtual pools, this issue has been alleviated. Virtual 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 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 pools, it is recommended to not attempt to rearrange the order of existing virtual pools in a backend definition. It is also advisable to not edit/modify attributes for an existing virtual pool and define a new virtual pool instead.

## Emulating different service levels/QoS

It is possible to design virtual pools for emulating service classes. Using the virtual pool implementation for Cloud Volume Service for Azure NetApp Files, let us examine how we can setup up different service classes. Configure the 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 pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools may be used to host a volume.

## Assigning specific set of aspects

Multiple virtual pools with a specific set of aspects can be designed from a single storage backend. For doing so, configure the backend with multiple labels and set the required aspects under each label. Now create different Kubernetes Storage Classes using the `parameters.selector` field that would map to different virtual pools. The volumes that get provisioned on the backend will have the aspects defined in the chosen virtual pool.

## PVC characteristics which affect storage provisioning

Some parameters beyond the requested storage class may affect the Astra Trident provisioning decision process when creating a PVC.

## Access mode

When requesting storage via a PVC, one of the mandatory fields is the access mode. The mode desired may affect the backend selected to host the storage request.

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](https://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 auto-scale functionality and many organizations use data from the metrics service for their charge back and/or show back applications.

Like with the logging service, and OpenShift as a whole, Ansible is used to deploy the metrics service. Also, like the logging service, the metrics service can be deployed during an initial setup of the cluster or after 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 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 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**

### **Security**

Use the recommendations listed here to ensure 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.

## Use Astra Trident with NVE and NAE

NetApp ONTAP provides data-at-rest encryption to protect sensitive data in the event a disk is stolen, returned, or repurposed. For details, refer to [Configure NetApp Volume Encryption overview](#).

- If NAE is enabled on the backend, any volume provisioned in Astra Trident will be NAE-enabled.
- If NAE is not enabled on the backend, any volume provisioned in Astra Trident will be NVE-enabled unless you set the NVE encryption flag to `false` in the backend configuration.



Volumes created in Astra Trident on an NAE-enabled backend must be NVE or NAE encrypted.

- You can set the NVE encryption flag to `true` in the Trident backend configuration to override the NAE encryption and use a specific encryption key on a per volume basis.
- Setting the NVE encryption flag to `false` on an NAE-enabled backend will create an NAE-enabled volume. You cannot disable NAE encryption by setting the NVE encryption flag to `false`.

- You can manually create an NVE volume in Astra Trident by explicitly setting the NVE encryption flag to `true`.

For more information on backend configuration options, refer to:

- [ONTAP SAN configuration options](#)

- [ONTAP NAS configuration options](#)

## Linux Unified Key Setup (LUKS)

You can enable Linux Unified Key Setup (LUKS) to encrypt ONTAP SAN and ONTAP SAN ECONOMY volumes on Astra Trident. Astra Trident supports passphrase rotation and volume expansion for LUKS-encrypted volumes.

In Astra Trident, LUKS-encrypted volumes use the aes-xts-plain64 cypher and mode, as recommended by [NIST](#).

### Before you begin

- Worker nodes must have cryptsetup 2.1 or higher (but lower than 3.0) installed. For more information, visit [Gitlab: cryptsetup](#).
- For performance reasons, we recommend that worker nodes support Advanced Encryption Standard New Instructions (AES-NI). To verify AES-NI support, run the following command:

```
grep "aes" /proc/cpuinfo
```

If nothing is returned, your processor does not support AES-NI. For more information on AES-NI, visit: [Intel: Advanced Encryption Standard Instructions \(AES-NI\)](#).

### Enable LUKS encryption

You can enable per-volume, host-side encryption using Linux Unified Key Setup (LUKS) for ONTAP SAN and ONTAP SAN ECONOMY volumes.

### Steps

1. Define LUKS encryption attributes in the backend configuration. For more information on backend configuration options for ONTAP SAN, refer to [ONTAP SAN configuration options](#).

```

"storage": [
  {
    "labels":{"luks": "true"},
    "zone":"us_east_1a",
    "defaults": {
      "luksEncryption": "true"
    }
  },
  {
    "labels":{"luks": "false"},
    "zone":"us_east_1a",
    "defaults": {
      "luksEncryption": "false"
    }
  },
]

```

2. Use `parameters.selector` to define the storage pools using LUKS encryption. For example:

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}

```

3. Create a secret that contains the LUKS passphrase. For example:

```

kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA

```

## Limitations

LUKS-encrypted volumes cannot take advantage of ONTAP deduplication and compression.

## Rotate a LUKS passphrase

You can rotate the LUKS passphrase and confirm rotation.



Do not forget a passphrase until you have verified it is no longer referenced by any volume, snapshot, or secret. If a referenced passphrase is lost, you might be unable to mount the volume and the data will remain encrypted and inaccessible.

### About this task

LUKS passphrase rotation occurs when a pod that mounts the volume is created after a new LUKS passphrase is specified. When a new pod is created, Astra Trident compares the LUKS passphrase on the volume to the active passphrase in the secret.

- If the passphrase on the volume does not match the active passphrase in the secret, rotation occurs.
- If the passphrase on the volume matches the active passphrase in the secret, the `previous-luks-passphrase` parameter is ignored.

### Steps

1. Add the `node-publish-secret-name` and `node-publish-secret-namespace` **StorageClass** parameters. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. Identify existing passphrases on the volume or snapshot.

### Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames: ["A"]
```

## Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames: ["A"]
```

3. Update the LUKS secret for the volume to specify the new and previous passphrases. Ensure previous-luke-passphrase-name and previous-luks-passphrase match the previous passphrase.

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. Create a new pod mounting the volume. This is required to initiate the rotation.
5. Verify the the passphrase was rotated.

## Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames: ["B"]
```

## Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames: ["B"]
```

## Results

The passphrase was rotated when only the new passphrase is returned on the volume and snapshot.



If two passphrases are returned, for example `luksPassphraseNames: ["B", "A"]`, the rotation is incomplete. You can trigger a new pod to attempt to complete the rotation.

## Enable volume expansion

You can enable volume expansion on a LUKS-encrypted volume.

### Steps

1. Enable the `CSINodeExpandSecret` feature gate (beta 1.25+). Refer to [Kubernetes 1.25: Use Secrets for Node-Driven Expansion of CSI Volumes](#) for details.
2. Add the `node-expand-secret-name` and `node-expand-secret-namespace` `StorageClass` parameters. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

### Results

When you initiate online storage expansion, the kubelet passes the appropriate credentials to the driver.

## Copyright information

Copyright © 2023 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

## Trademark information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.