Best practices and recommendations

Trident

NetApp

February 02, 2026

Table of Contents

Best practices and recommendations

Deployment

Use the recommendations listed here when you deploy Trident.

Deploy to a dedicated namespace

Namespaces provide administrative separation between different applications and are a barrier for resource sharing. For example, a PVC from one namespace cannot be consumed from another. Trident provides PV resources to all the namespaces in the Kubernetes cluster and consequently leverages a service account which has elevated privileges.

Additionally, access to the Trident pod might enable a user to access storage system credentials and other sensitive information. It is important to ensure that application users and management applications do not have the ability to access the Trident object definitions or the pods themselves.

Use quotas and range limits to control storage consumption

Kubernetes has two features which, when combined, provide a powerful mechanism for limiting the resource consumption by applications. The storage quota mechanism enables the administrator to implement global, and storage class specific, capacity and object count consumption limits on a per-namespace basis. Further, using a range limit ensures that the PVC requests are within both a minimum and maximum value before the request is forwarded to the provisioner.

These values are defined on a per-namespace basis, which means that each namespace should have values defined which fall in line with their resource requirements. See here for information about how to leverage quotas.

Storage configuration

Each storage platform in the NetApp portfolio has unique capabilities that benefit applications, containerized or not.

Platform overview

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

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

ONTAP and Cloud Volumes ONTAP best practices

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

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

Use SVM(s) dedicated to Trident

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

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

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

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

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

Limit the maximum volume count

ONTAP storage systems have a maximum volume count, which varies based on the software version and hardware platform. Refer to NetApp Hardware Universe for your specific platform and ONTAP version to determine the exact limits. When the volume count is exhausted, provisioning operations fail not only for Trident, but for all the storage requests.

Trident's `ontap-nas` and `ontap-san` drivers provision a FlexVolume for each Kubernetes Persistent Volume (PV) that is created. The `ontap-nas-economy` driver creates approximately one FlexVolume for every 200 PVs (configurable between 50 and 300). The `ontap-san-economy` driver creates approximately one FlexVolume for every 100 PVs (configurable between 50 and 200). To prevent Trident from consuming all the available volumes on the storage system, you should set a limit on the SVM. You can do this from the command line:

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

The value for `max-volumes` varies based on several criteria specific to your environment:

- The number of existing volumes in the ONTAP cluster
- The number of volumes you expect to provision outside of Trident for other applications
- The number of persistent volumes expected to be consumed by Kubernetes applications

The `max-volumes` value is the total volumes provisioned across all the nodes in the ONTAP cluster, and not on an individual ONTAP node. As a result, you might encounter some conditions where an ONTAP cluster node might have far more or less Trident provisioned volumes than another node.

For example, a two-node ONTAP cluster has the ability to host a maximum of 2000 FlexVol volumes. Having the maximum volume count set to 1250 appears very reasonable. However, if only aggregates from one node are assigned to the SVM, or the aggregates assigned from one node are unable to be provisioned against (for example, due to capacity), then the other node becomes the target for all Trident provisioned volumes. This means that the volume limit might be reached for that node before the `max-volumes` value is reached, resulting in impacting both Trident and other volume operations that use that node. **You can avoid this situation by ensuring that aggregates from each node in the cluster are assigned to the SVM used by Trident in equal numbers.**

Clone a volume

NetApp Trident supports cloning volumes when using the `ontap-nas`, `ontap-san`, and `solidfire-san` storage drivers. When using the `ontap-nas-flexgroup` or `ontap-nas-economy` drivers, cloning is not supported. Creating a new volume from an existing volume will result in a new snapshot being created.

> Avoid cloning a PVC that is associated with a different StorageClass. Perform cloning operations within the same StorageClass to ensure compatibility and prevent unexpected behavior.

Limit the maximum size of volumes created by Trident

To configure the maximum size for volumes that can be created by Trident, use the `limitVolumeSize` parameter in your `backend.json` definition.

In addition to controlling the volume size at the storage array, you should also leverage Kubernetes capabilities.

Limit the maximum size of FlexVols created by Trident

To configure the maximum size for FlexVols used as pools for ontap-san-economy and ontap-nas-economy drivers, use the `limitVolumePoolSize` parameter in your `backend.json` definition.

Configure Trident to use bidirectional CHAP

You can specify the CHAP initiator and target usernames and passwords in your backend definition and have Trident enable CHAP on the SVM. Using the `useCHAP` parameter in your backend configuration, Trident authenticates iSCSI connections for ONTAP backends with CHAP.

Create and use an SVM QoS policy

Leveraging an ONTAP QoS policy, applied to the SVM, limits the number of IOPS consumable by the Trident provisioned volumes. This helps to prevent a bully or out-of-control container from affecting workloads outside of the Trident SVM.

You can create a QoS policy for the SVM in a few steps. See the documentation for your version of ONTAP for the most accurate information. The example below creates a QoS policy that limits the total IOPS available to the SVM to 5000.

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

Additionally, if your version of ONTAP supports it, you can consider using a QoS minimum to guarantee an amount of throughput to containerized workloads. Adaptive QoS is not compatible with an SVM level policy.

The number of IOPS dedicated to the containerized workloads depends on many aspects. Among other things, these include:

- Other workloads using the storage array. If there are other workloads, not related to the Kubernetes deployment, utilizing the storage resources, care should be taken to ensure that those workloads are not accidentally adversely impacted.
- Expected workloads running in containers. If workloads which have high IOPS requirements will be running in containers, a low QoS policy results in a bad experience.

It’s important to remember that a QoS policy assigned at the SVM level results in all the volumes provisioned to the SVM sharing the same IOPS pool. If one, or a small number, of the containerized applications have a high IOPS requirement, it could become a bully to the other containerized workloads. If this is the case, you might want to consider using external automation to assign per-volume QoS policies.

> You should assign the QoS policy group to the SVM **only** if your ONTAP version is earlier than 9.8.

Create QoS policy groups for Trident

Quality of service (QoS) guarantees that performance of critical workloads is not degraded by competing workloads. ONTAP QoS policy groups provide QoS options for volumes, and enable users to define the throughput ceiling for one or more workloads. For more information about QoS, refer to Guaranteeing throughput with QoS.
You can specify QoS policy groups in the backend or in a storage pool, and they are applied to each volume created in that pool or backend.

ONTAP has two kinds of QoS policy groups: traditional and adaptive. Traditional policy groups provide a flat maximum (or minimum, in later versions) throughput in IOPS. Adaptive QoS automatically scales the throughput to workload size, maintaining the ratio of IOPS to TBs|GBs as the size of the workload changes. This provides a significant advantage when you are managing hundreds or thousands of workloads in a large deployment.

Consider the following when you create QoS policy groups:

- You should set the `qosPolicy` key in the `defaults` block of the backend configuration. See the following backend configuration example:

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
  - labels:
      performance: extreme
    defaults:
      adaptiveQosPolicy: extremely-adaptive-pg
  - labels:
      performance: premium
    defaults:
      qosPolicy: premium-pg
```

- You should apply the policy groups per volume, so that each volume gets the entire throughput as specified by the policy group. Shared policy groups are not supported.

For more information about QoS policy groups, refer to ONTAP command reference.

Limit storage resource access to Kubernetes cluster members

Limiting access to the NFS volumes, iSCSI LUNs, and FC LUNs created by Trident is a critical component of the security posture for your Kubernetes deployment. Doing so prevents hosts that are not a part of the Kubernetes cluster from accessing the volumes and potentially modifying data unexpectedly.

It's important to understand that namespaces are the logical boundary for resources in Kubernetes. The assumption is that resources in the same namespace are able to be shared, however, importantly, there is no cross-namespace capability. This means that even though PVs are global objects, when bound to a PVC they are only accessible by pods which are in the same namespace. **It is critical to ensure that namespaces are used to provide separation when appropriate.**

The primary concern for most organizations with regard to data security in a Kubernetes context is that a process in a container can access storage mounted to the host, but which is not intended for the container. Namespaces are designed to prevent this type of compromise. However, there is one exception: privileged containers.

A privileged container is one that is run with substantially more host-level permissions than normal. These are not denied by default, so ensure that you disable the capability by using pod security policies.

For volumes where access is desired from both Kubernetes and external hosts, the storage should be managed in a traditional manner, with the PV introduced by the administrator and not managed by Trident. This ensures that the storage volume is destroyed only when both the Kubernetes and external hosts have disconnected and are no longer using the volume. Additionally, a custom export policy can be applied, which enables access from the Kubernetes cluster nodes and targeted servers outside of the Kubernetes cluster.

For deployments which have dedicated infrastructure nodes (for example, OpenShift) or other nodes which are unable to schedule user applications, separate export policies should be used to further limit access to storage resources. This includes creating an export policy for services which are deployed to those infrastructure nodes (for example, the OpenShift Metrics and Logging services), and standard applications which are deployed to non-infrastructure nodes.

Use a dedicated export policy

You should ensure that an export policy exists for each backend that only allows access to the nodes present in the Kubernetes cluster. Trident can automatically create and manage export policies. This way, Trident limits access to the volumes it provisions to the nodes in the Kubernetes cluster and simplifies the addition/deletion of nodes.

Alternatively, you can also create an export policy manually and populate it with one or more export rules that process each node access request:

- Use the `vserver export-policy create` ONTAP CLI command to create the export policy.
- Add rules to the export policy by using the `vserver export-policy rule create` ONTAP CLI command.

Running these commands enables you to restrict which Kubernetes nodes have access to the data.

Disable `showmount` for the application SVM

The `showmount` feature enables an NFS client to query the SVM for a list of available NFS exports. A pod deployed to the Kubernetes cluster can issue the `showmount -e` command against the and receive a list of available mounts, including those which it does not have access to. While this, by itself, is not a security compromise, it does provide unnecessary information potentially aiding an unauthorized user with connecting to an NFS export.

You should disable `showmount` by using the SVM-level ONTAP CLI command:

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire best practices

Learn the best practices for configuring SolidFire storage for Trident.

Create Solidfire Account

Each SolidFire account represents a unique volume owner and receives its own set of Challenge-Handshake Authentication Protocol (CHAP) credentials. You can access volumes assigned to an account either by using the account name and the relative CHAP credentials or through a volume access group. An account can have up to two-thousand volumes assigned to it, but a volume can belong to only one account.

Create a QoS policy

Use SolidFire Quality of Service (QoS) policies if you want to create and save a standardized quality of service setting that can be applied to many volumes.

You can set QoS parameters on a per-volume basis. Performance for each volume can be assured by setting three configurable parameters that define the QoS: Min IOPS, Max IOPS, and Burst IOPS.

Here are the possible minimum, maximum, and burst IOPS values for the 4Kb block size.

IOPS parameter	Definition	Min. value	Default value	Max. value(4Kb)
Min IOPS	The guaranteed level of performance for a volume.	50	50	15000
Max IOPS	The performance will not exceed this limit.	50	15000	200,000
Burst IOPS	Maximum IOPS allowed in a short burst scenario.	50	15000	200,000

Although the Max IOPS and Burst IOPS can be set as high as 200,000, the real-world maximum performance of a volume is limited by cluster usage and per-node performance.

Block size and bandwidth have a direct influence on the number of IOPS. As block sizes increase, the system increases bandwidth to a level necessary to process the larger block sizes. As bandwidth increases, the number of IOPS the system is able to attain decreases. Refer to SolidFire Quality of Service for more information about QoS and performance.

SolidFire authentication

Element supports two methods for authentication: CHAP and Volume Access Groups (VAG). CHAP uses the CHAP protocol to authenticate the host to the backend. Volume Access Groups controls access to the volumes it provisions. NetApp recommends using CHAP for authentication as it’s simpler and has no scaling limits.

Trident with the enhanced CSI provisioner supports the use of CHAP authentication. VAGs should only be used in the traditional non-CSI mode of operation.

CHAP authentication (verification that the initiator is the intended volume user) is supported only with account-based access control. If you are using CHAP for authentication, two options are available: unidirectional CHAP and bidirectional CHAP. Unidirectional CHAP authenticates volume access by using the SolidFire account name and initiator secret. The bidirectional CHAP option provides the most secure way of authenticating the volume because the volume authenticates the host through the account name and the initiator secret, and then the host authenticates the volume through the account name and the target secret.

However, if CHAP cannot be enabled and VAGs are required, create the access group and add the host initiators and volumes to the access group. Each IQN that you add to an access group can access each volume in the group with or without CHAP authentication. If the iSCSI initiator is configured to use CHAP authentication, account-based access control is used. If the iSCSI initiator is not configured to use CHAP authentication, then Volume Access Group access control is used.

Where to find more information?

Some of the best practices documentation is listed below. Search the NetApp library for the most current versions.

ONTAP

- NFS Best Practice and Implementation Guide
- SAN Administration (for iSCSI)
- iSCSI Express Configuration for RHEL

Element software

- Configuring SolidFire for Linux

NetApp HCI

- NetApp HCI deployment prerequisites
- Access the NetApp Deployment Engine

Application best practices information

- Best practices for MySQL on ONTAP
- Best practices for MySQL on SolidFire
- NetApp SolidFire and Cassandra
- Oracle best practices on SolidFire
- PostgreSQL best practices on SolidFire

Not all applications have specific guidelines, it’s important to work with your NetApp team and to use the NetApp library to find the most up-to-date documentation.

Integrate Trident

To integrate Trident, the following design and architectural elements require integration: driver selection and deployment, storage class design, virtual pool design, Persistent Volume Claim (PVC) impacts on storage provisioning, volume operations, and OpenShift services deployment using Trident.

Driver selection and deployment

Select and deploy a backend driver for your storage system.

ONTAP backend drivers

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

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

The available ONTAP backend drivers are:

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

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

Note that, in the tables below, not all of the capabilities are exposed through Trident. Some must be applied by the storage administrator after provisioning if that functionality is desired. The superscript footnotes distinguish the functionality per feature and driver.

ONTAP NAS drivers	Snapshots	Clones	Dynamic export policies	Multi-attach	QoS	Resize	Replication
`ontap-nas`	Yes	Yes	Yes [5]	Yes	Yes [1]	Yes	Yes [1]
`ontap-nas-economy`	NO [3]	NO [3]	Yes [5]	Yes	NO [3]	Yes	NO [3]
`ontap-nas-flexgroup`	Yes [1]	NO	Yes [5]	Yes	Yes [1]	Yes	Yes [1]

Trident offers 2 SAN drivers for ONTAP, whose capabilities are shown below.

ONTAP SAN drivers	Snapshots	Clones	Multi-attach	Bi-directional CHAP	QoS	Resize	Replication
`ontap-san`	Yes	Yes	Yes [4]	Yes	Yes [1]	Yes	Yes [1]
`ontap-san-economy`	Yes	Yes	Yes [4]	Yes	NO [3]	Yes	NO [3]

Footnote for the above tables:

Yes [1]: Not managed by Trident

Yes [2]: Managed by Trident, but not PV granular

NO [3]: Not managed by Trident and not PV granular

Yes [4]: Supported for raw-block volumes

Yes [5]: Supported by Trident

The features that are not PV granular are applied to the entire FlexVolume and all of the PVs (that is, qtrees or

LUNs in shared FlexVols) will share a common schedule.

As we can see in the above tables, much of the functionality between the `ontap-nas` and `ontap-nas-economy` is the same. However, because the `ontap-nas-economy` driver limits the ability to control the schedule at per-PV granularity, this can affect your disaster recovery and backup planning in particular. For development teams which desire to leverage PVC clone functionality on ONTAP storage, this is only possible when using the `ontap-nas`, `ontap-san` or `ontap-san-economy` drivers.

The `solidfire-san` driver is also capable of cloning PVCs.

Cloud Volumes ONTAP backend drivers

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

Amazon FSx for ONTAP backend drivers

Amazon FSx for NetApp ONTAP lets you leverage NetApp features, performance, and administrative capabilities you’re familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx for ONTAP supports many ONTAP file system features and administration APIs. The compatible drivers for Cloud Volume ONTAP are `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san` and `ontap-san-economy`.

NetApp HCI/SolidFire backend drivers

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

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

> Footnote:
>
> Yes [1]: Not managed by Trident
>
> Yes [2]: Supported for raw-block volumes

Azure NetApp Files backend drivers

Trident uses the `azure-netapp-files` driver to manage the Azure NetApp Files service.

More information about this driver and how to configure it can be found in Trident backend configuration for Azure NetApp Files.

Azure NetApp Files Driver	Snapshots	Clones	Multi-attach	QoS	Expand	Replication
`azure-netapp-files`	Yes	Yes	Yes	Yes	Yes	Yes [1]

> Footnote:
>
> Yes [1]: Not managed by Trident

Storage class design

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

Specific backend utilization

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

The `storagePools` parameter helps restrict storage to the set of pools that match any specified attributes. The `additionalStoragePools` parameter is used to extend the set of pools that Trident use for provisioning along with the set of pools selected by the attributes and `storagePools` parameters. You can use either parameter alone or both together to make sure that the appropriate set of storage pools are selected.

The `excludeStoragePools` parameter is used to specifically exclude the listed set of pools that match the attributes.

Emulate QoS policies

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

Utilize backend based on specific features

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

Virtual pools

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

Virtual pools allow an administrator to create a level of abstraction over backends which can be referenced through Storage Classes, for greater flexibility and efficient placement of volumes on backends. Different backends can be defined with the same class of service. Moreover, multiple storage pools can be created on the same backend but with different characteristics. When a Storage Class is configured with a selector with the specific labels, Trident chooses a backend which matches all the selector labels to place the volume. If the Storage Class selector labels matches multiple storage pools, Trident will choose one of them to provision the volume from.

Virtual pool design

While creating a backend, you can generally specify a set of parameters. It was impossible for the administrator to create another backend with the same storage credentials and with a different set of parameters. With the introduction of virtual pools, this issue has been alleviated. A virtual pool is a level abstraction introduced between the backend and the Kubernetes Storage Class so that the administrator can define parameters along with labels which can be referenced through Kubernetes Storage Classes as a selector, in a backend-agnostic way. Virtual pools can be defined for all supported NetApp backends with Trident. That list includes SolidFire/NetApp HCI, ONTAP, as well as Azure NetApp Files.

When defining virtual pools, it is recommended to not attempt to rearrange the order of existing virtual pools in a backend definition. It is also advisable to not edit/modify attributes for an existing virtual pool and define a new virtual pool instead.

Emulating different service levels/QoS

It is possible to design virtual pools for emulating service classes. Using the virtual pool implementation for Cloud Volume Service for Azure NetApp Files, let us examine how we can setup up different service classes. Configure the Azure NetApp Files backend with multiple labels, representing different performance levels. Set `servicelevel` aspect to the appropriate performance level and add other required aspects under each labels. Now create different Kubernetes Storage Classes that would map to different virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools may be used to host a volume.

Assigning specific set of aspects

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

PVC characteristics which affect storage provisioning

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

Access mode

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

Trident will attempt to match the storage protocol used with the access method specified according to the following matrix. This is independent of the underlying storage platform.

	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
iSCSI	Yes	Yes	Yes (Raw block)
NFS	Yes	Yes	Yes

A request for a ReadWriteMany PVC submitted to a Trident deployment without an NFS backend configured will result in no volume being provisioned. For this reason, the requestor should use the access mode which is appropriate for their application.

Volume operations

Modify persistent volumes

Persistent volumes are, with two exceptions, immutable objects in Kubernetes. Once created, the reclaim policy and the size can be modified. However, this doesn't prevent some aspects of the volume from being modified outside of Kubernetes. This may be desirable in order to customize the volume for specific applications, to ensure that capacity is not accidentally consumed, or simply to move the volume to a different storage controller for any reason.

Kubernetes in-tree provisioners do not support volume resize operations for NFS, iSCSI, or FC PVs at this time. Trident supports expanding both NFS, iSCSI, and FC volumes.

The connection details of the PV cannot be modified after creation.

Create on-demand volume snapshots

Trident supports on-demand volume snapshot creation and the creation of PVCs from snapshots using the CSI framework. Snapshots provide a convenient method of maintaining point-in-time copies of the data and have a lifecycle independent of the source PV in Kubernetes. These snapshots can be used to clone PVCs.

Create volumes from snapshots

Trident also supports the creation of PersistentVolumes from volume snapshots. To accomplish this, just create a PersistentVolumeClaim and mention the `datasource` as the required snapshot from which the volume needs to be created. Trident will handle this PVC by creating a volume with the data present on the snapshot. With this feature, it is possible to duplicate data across regions, create test environments, replace a damaged or corrupted production volume in its entirety, or retrieve specific files and directories and transfer them to another attached volume.

Move volumes in the cluster

Storage administrators have the ability to move volumes between aggregates and controllers in the ONTAP cluster non-disruptively to the storage consumer. This operation does not affect Trident or the Kubernetes cluster, as long as the destination aggregate is one which the SVM that Trident is using has access to. Importantly, if the aggregate has been newly added to the SVM, the backend will need to be refreshed by re-adding it to Trident. This will trigger Trident to reinventory the SVM so that the new aggregate is recognized.

However, moving volumes across backends is not supported automatically by Trident. This includes between SVMs in the same cluster, between clusters, or onto a different storage platform (even if that storage system is one which is connected to Trident).

If a volume is copied to another location, the volume import feature may be used to import current volumes into Trident.

Expand volumes

Trident supports resizing NFS, iSCSI, and FC PVs. This enables users to resize their volumes directly through the Kubernetes layer. Volume expansion is possible for all major NetApp storage platforms, including ONTAP, and SolidFire/NetApp HCI backends. To allow possible expansion later, set `allowVolumeExpansion` to `true` in your StorageClass associated with the volume. Whenever the Persistent Volume needs to be resized, edit the `spec.resources.requests.storage` annotation in the Persistent Volume Claim to the required volume size. Trident will automatically take care of resizing the volume on the storage cluster.

Import an existing volume into Kubernetes

Volume import provides the ability to import an existing storage volume into a Kubernetes environment. This is currently supported by the `ontap-nas`, `ontap-nas-flexgroup`, `solidfire-san`, and `azure-netapp-files` drivers. This feature is useful when porting an existing application into Kubernetes or during disaster recovery scenarios.

When using the ONTAP and `solidfire-san` drivers, use the command `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml` to import an existing volume into Kubernetes to be managed by Trident. The PVC YAML or JSON file used in the import volume command points to a storage class which identifies Trident as the provisioner. When using a NetApp HCI/SolidFire backend, ensure the volume names are unique. If the volume names are duplicated, clone the volume to a unique name so the volume import feature can distinguish between them.

If the `azure-netapp-files` driver is used, use the command `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` to import the volume into Kubernetes to be managed by Trident. This ensures a unique volume reference.

When the above command is executed, Trident will find the volume on the backend and read its size. It will automatically add (and overwrite if necessary) the configured PVC's volume size. Trident then creates the new PV and Kubernetes binds the PVC to the PV.

If a container was deployed such that it required the specific imported PVC, it would remain in a pending state until the PVC/PV pair are bound via the volume import process. After the PVC/PV pair are bound, the container should come up, provided there are no other issues.

Registry service

Deploying and managing storage for the registry has been documented on netapp.io in the blog.

Logging service

Like other OpenShift services, the logging service is deployed using Ansible with configuration parameters supplied by the inventory file, a.k.a. hosts, provided to the playbook. There are two installation methods which will be covered: deploying logging during initial OpenShift install and deploying logging after OpenShift has been
installed.

> ! As of Red Hat OpenShift version 3.9, the official documentation recommends against NFS for the logging service due to concerns around data corruption. This is based on Red Hat testing of their products. The ONTAP NFS server does not have these issues, and can easily back a logging deployment. Ultimately, the choice of protocol for the logging service is up to you, just know that both will work great when using NetApp platforms and there is no reason to avoid NFS if that is your preference.

If you choose to use NFS with the logging service, you will need to set the Ansible variable `openshift_enable_unsupported_configurations` to `true` to prevent the installer from failing.

Get started

The logging service can, optionally, be deployed for both applications as well as for the core operations of the OpenShift cluster itself. If you choose to deploy operations logging, by specifying the variable `openshift_logging_use_ops` as `true`, two instances of the service will be created. The variables which control the logging instance for operations contain "ops" in them, whereas the instance for applications does not.

Configuring the Ansible variables according to the deployment method is important to ensure that the correct storage is utilized by the underlying services. Let’s look at the options for each of the deployment methods.

The tables below contain only the variables relevant for storage configuration as it relates to the logging service. You can find other options in Red Hat OpenShift logging documentation which should be reviewed, configured, and used according to your deployment.

The variables in the below table will result in the Ansible playbook creating a PV and PVC for the logging service using the details provided. This method is significantly less flexible than using the component installation playbook after OpenShift installation, however, if you have existing volumes available, it is an option.

Variable	Details
`openshift_logging_storage_kind`	Set to `nfs` to have the installer create an NFS PV for the logging service.
`openshift_logging_storage_host`	The hostname or IP address of the NFS host. This should be set to the dataLIF for your virtual machine.
`openshift_logging_storage_nfs_directory`	The mount path for the NFS export. For example, if the volume is junctioned as `/openshift_logging`, you would use that path for this variable.
`openshift_logging_storage_volume_name`	The name, e.g. `pv_ose_logs`, of the PV to create.
`openshift_logging_storage_volume_size`	The size of the NFS export, for example `100Gi`.

If your OpenShift cluster is already running, and therefore Trident has been deployed and configured, the installer can use dynamic provisioning to create the volumes. The following variables will need to be configured.

Variable	Details
`openshift_logging_es_pvc_dynamic`	Set to true to use dynamically provisioned volumes.
`openshift_logging_es_pvc_storage_class_name`	The name of the storage class which will be used in the PVC.
`openshift_logging_es_pvc_size`	The size of the volume requested in the PVC.
`openshift_logging_es_pvc_prefix`	A prefix for the PVCs used by the logging service.
`openshift_logging_es_ops_pvc_dynamic`	Set to `true` to use dynamically provisioned volumes for the ops logging instance.

Variable	Details
`openshift_logging_es_ops_pvc_storage_class_name`	The name of the storage class for the ops logging instance.
`openshift_logging_es_ops_pvc_size`	The size of the volume request for the ops instance.
`openshift_logging_es_ops_pvc_prefix`	A prefix for the ops instance PVCs.

Deploy the logging stack

If you are deploying logging as a part of the initial OpenShift install process, then you only need to follow the standard deployment process. Ansible will configure and deploy the needed services and OpenShift objects so that the service is available as soon as Ansible completes.

However, if you are deploying after the initial installation, the component playbook will need to be used by Ansible. This process may change slightly with different versions of OpenShift, so be sure to read and follow Red Hat OpenShift Container Platform 3.11 documentation for your version.

Metrics service

The metrics service provides valuable information to the administrator regarding the status, resource utilization, and availability of the OpenShift cluster. It is also necessary for pod auto-scale functionality and many organizations use data from the metrics service for their charge back and/or show back applications.

Like with the logging service, and OpenShift as a whole, Ansible is used to deploy the metrics service. Also, like the logging service, the metrics service can be deployed during an initial setup of the cluster or after its operational using the component installation method. The following tables contain the variables which are important when configuring persistent storage for the metrics service.

The tables below only contain the variables which are relevant for storage configuration as it relates to the metrics service. There are many other options found in the documentation which should be reviewed, configured, and used according to your deployment.

Variable	Details
`openshift_metrics_storage_kind`	Set to `nfs` to have the installer create an NFS PV for the logging service.
`openshift_metrics_storage_host`	The hostname or IP address of the NFS host. This should be set to the dataLIF for your SVM.
`openshift_metrics_storage_nfs_directory`	The mount path for the NFS export. For example, if the volume is junctioned as `/openshift_metrics`, you would use that path for this variable.
`openshift_metrics_storage_volume_name`	The name, e.g. `pv_ose_metrics`, of the PV to create.
`openshift_metrics_storage_volume_size`	The size of the NFS export, for example `100Gi`.

If your OpenShift cluster is already running, and therefore Trident has been deployed and configured, the installer can use dynamic provisioning to create the volumes. The following variables will need to be configured.

Variable	Details
`openshift_metrics_cassandra_pvc_prefix`	A prefix to use for the metrics PVCs.
`openshift_metrics_cassandra_pvc_size`	The size of the volumes to request.
`openshift_metrics_cassandra_storage_type`	The type of storage to use for metrics, this must be set to dynamic for Ansible to create PVCs with the appropriate storage class.
`openshift_metrics_cassanda_pvc_storage_class_name`	The name of the storage class to use.

Deploy the metrics service

With the appropriate Ansible variables defined in your hosts/inventory file, deploy the service using Ansible. If you are deploying at OpenShift install time, then the PV will be created and used automatically. If you're deploying using the component playbooks, after OpenShift install, then Ansible creates any PVCs which are needed and, after Trident has provisioned storage for them, deploy the service.

The variables above, and the process for deploying, may change with each version of OpenShift. Ensure you review and follow Red Hat's OpenShift deployment guide for your version so that it is configured for your environment.

Data protection and disaster recovery

Learn about protection and recovery options for Trident and volumes created using Trident. You should have a data protection and recovery strategy for each application with a persistence requirement.

Trident replication and recovery

You can create a backup to restore Trident in the event of a disaster.

Trident replication

Trident uses Kubernetes CRDs to store and manage its own state and the Kubernetes cluster etcd to store its metadata.

Steps

1. Back up the Kubernetes cluster etcd using Kubernetes: Backing up an etcd cluster.
2. Place the backup artifacts on a FlexVol volume

NetApp recommends you protect the SVM where the FlexVol resides with a SnapMirror relationship to another SVM.

Trident recovery

Using Kubernetes CRDs and the Kubernetes cluster etcd snapshot, you can recover Trident.

Steps

1. From the destination SVM, mount the volume which contains the Kubernetes etcd data files and certificates

on to the host which will be set up as a master node.

2. Copy all required certificates pertaining to the Kubernetes cluster under `/etc/kubernetes/pki` and the etcd member files under `/var/lib/etcd`.
3. Restore the Kubernetes cluster from the etcd backup using Kubernetes: Restoring an etcd cluster.
4. Run `kubectl get crd` to verify all Trident custom resources have come up and retrieve the Trident objects to verify all data is available.

SVM replication and recovery

Trident cannot configure replication relationships, however, the storage administrator can use ONTAP SnapMirror to replicate an SVM.

In the event of a disaster, you can activate the SnapMirror destination SVM to start serving data. You can switch back to the primary when systems are restored.

About this task

Consider the following when using the SnapMirror SVM Replication feature:

- You should create a distinct backend for each SVM with SVM-DR enabled.
- Configure the storage classes to select the replicated backends only when needed to avoid having volumes which do not need replication provisioned onto the backends that support SVM-DR.
- Application administrators should understand the additional cost and complexity associated with replication and carefully consider their recovery plan prior to beginning this process.

SVM replication

You can use ONTAP: SnapMirror SVM replication to create the SVM replication relationship.

SnapMirror allows you to set options to control what to replicate. You'll need to know which options you selected when preforming SVM recovery using Trident.

- -identity-preserve true replicates the entire SVM configuration.
- -discard-configs network excludes LIFs and related network settings.
- -identity-preserve false replicates only the volumes and security configuration.

SVM recovery using Trident

Trident does not automatically detect SVM failures. In the event of a disaster, the administrator can manually initiate Trident failover to the new SVM.

Steps

1. Cancel scheduled and ongoing SnapMirror transfers, break the replication relationship, stop the source SVM and then activate the SnapMirror destination SVM.
2. If you specified `-identity-preserve false` or `-discard-config network` when configuring your SVM replication, update the `managementLIF` and `dataLIF` in the Trident backend definition file.
3. Confirm `storagePrefix` is present in the Trident backend definition file. This parameter cannot be changed. Omitting `storagePrefix` will cause the backend update to fail.
4. Update all the required backends to reflect the new destination SVM name using:

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. If you specified `-identity-preserve false` or `discard-config network`, you must bounce all application pods.

   > If you specified `-identity-preserve true`, all volumes provisioned by Trident start serving data when the destination SVM is activated.

Volume replication and recovery

Trident cannot configure SnapMirror replication relationships, however, the storage administrator can use ONTAP SnapMirror replication and recovery to replicate volumes created by Trident.

You can then import the recovered volumes into Trident using tridentctl volume import.

> Import is not supported on `ontap-nas-economy`, `ontap-san-economy`, or `ontap-flexgroup-economy` drivers.

Snapshot data protection

You can protect and restore data using:

- An external snapshot controller and CRDs to create Kubernetes volume snapshots of Persistent Volumes (PVs).

  Volume snapshots

- ONTAP Snapshots to restore the entire contents of a volume or to recover individual files or LUNs.

  ONTAP Snapshots

Automating the failover of stateful applications with Trident

Trident's force-detach feature allows you to automatically detach volumes from unhealthy nodes in a Kubernetes cluster, preventing data corruption and ensuring application availability. This feature is particularly useful in scenarios where nodes become unresponsive or are taken offline for maintenance.

Details about force detach

Force detach is available for `ontap-san`, `ontap-san-economy`, `ontap-nas`, and `ontap-nas-economy` only. Before enabling force detach, non-graceful node shutdown (NGNS) must be enabled on the Kubernetes cluster. NGNS is enabled by default for Kubernetes 1.28 and above. For more information, refer to Kubernetes: Non Graceful node shutdown.

When using the `ontap-nas` or `ontap-nas-economy` driver, you need to set the `autoExportPolicy` parameter in the backend configuration to `true` so that Trident can restrict access from the Kubernetes node with the taint applied using managed export policies.

Because Trident relies on Kubernetes NGNS, do not remove `out-of-service` taints from an unhealthy node until all non-tolerable workloads are rescheduled. Recklessly applying or removing the taint can jeopardize backend data protection.

When the Kubernetes cluster administrator has applied the `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute` taint to the node and `enableForceDetach` is set to `true`, Trident will determine the node status and:

1. Stop backend I/O access for volumes mounted to that node.
2. Mark the Trident node object as `dirty` (not safe for new publications).

The Trident controller will reject new publish volume requests until the node is re-qualified (after having been marked as `dirty`) by the Trident node pod. Any workloads scheduled with a mounted PVC (even after the cluster node is healthy and ready) will be not be accepted until Trident can verify the node `clean` (safe for new publications).

When node health is restored and the taint is removed, Trident will:

1. Identify and clean stale published paths on the node.
2. If the node is in a `cleanable` state (the out-of-service taint has been removed and the node is in `Ready` state) and all stale, published paths are clean, Trident will readmit the node as `clean` and allow new published volumes to the node.

Details about automated failover

You can automate the force-detach process through integration with node health check (NHC) operator. When a node failure occurs, NHC triggers Trident node remediation (TNR) and force-detach automatically by creating a TridentNodeRemediation CR in Trident's namespace defining the failed node. TNR is created only upon node failure, and removed by NHC once the node comes back online or the node is deleted.

Failed node pod removal process

Automated-failover selects the workloads to remove from the failed node. When a TNR is created, the TNR controller marks the node as dirty, preventing any new volume publications and begins removing force-detach supported pods and their volume attachments.

All volumes/PVCs supported by force-detach are supported by automated-failover:

- NAS, and NAS-economy volumes using auto-export policies (SMB is not yet supported).
- SAN, and SAN-economy volumes.

Refer to Details about force detach.

Default behavior:

- Pods using volumes supported by force-detach are removed from the failed node. Kubernetes will reschedule these onto a healthy node.

- Pods using a volume not supported by force-detach, including non-Trident volumes, are not removed from the failed node.
- Stateless pods (not PVCs) are not removed from the failed node, unless the pod annotation `trident.netapp.io/podRemediationPolicy: delete` is set.

Overriding the pod removal behavior:

Pod removal behavior can be customized using a pod annotation: `trident.netapp.io/podRemediationPolicy[retain, delete]`. These annotations are examined and used when a failover occurs.
Apply annotations to the Kubernetes deployment/replicaset pod spec to prevent the annotation from disappearing after a failover:

- `retain` - Pod WILL NOT be removed from the failed node during an automated-failover.
- `delete` - Pod WILL be removed from the failed node during an automated-failover.

These annotations can be applied to any pod.

- I/O operations will be blocked only on failed nodes for volumes that support force-detach.
- For volumes that do not support force-detach, there is a risk of data corruption and multi-attach issues.

TridentNodeRemediation CR

The TridentNodeRemediation (TNR) CR defines a failed node. The name of the TNR is the name of the failed node.

Example TNR:

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediation
metadata:
  name: <K8s-node-name>
spec: {}
```

TNR states:
Use the following commands to view the status of TNRs:
`kubectl get tnr <name> -n <trident-namespace>`

TNRs can be in one of the following states:

- *Remediating*:
  - Cease backend I/O access for volumes supported by force-detach mounted to that node.
  - The Trident node object is marked dirty (not safe for new publications).
  - Remove pods and volume attachments from the node
- *NodeRecoveryPending*:

    - The controller is waiting for the node to come back online.
    - Once the node is online, publish-enforcement will ensure the node is clean and ready for new volume publications.
- If the node is deleted from K8s, the TNR controller will remove the TNR and cease reconciliation.
- *Succeeded*:
    - All remediation and node recovery steps completed successfully. The node is clean and ready for new volume publications.
- *Failed*:
    - Unrecoverable error. Error reasons are set in the status.message field of the CR.

Enabling automated-failover

Prerequisites:

- Ensure that force detach is enabled before enabling automated-failover. For more information, refer to Details about force detach.
- Install node health check (NHC) in the Kubernetes cluster.
    - Install operator-sdk.
    - Install Operator Lifecycle Manager (OLM) in the cluster if not already installed: `operator-sdk olm install`.
    - Install Node Health check Operator: `kubectl create -f https://operatorhub.io/install/node-healthcheck-operator.yaml`.

You can also use alternative ways to detect node failure as specified in the [Integrating Custom Node Health Check Solutions] section below.

See Node Health Check Operator for more information.

Steps

1. Create a NodeHealthCheck (NHC) CR in the Trident namespace to monitor the worker nodes in the cluster. Example:

```yaml
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: <CR name>
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  remediationTemplate:
    apiVersion: trident.netapp.io/v1
    kind: TridentNodeRemediationTemplate
    namespace: <Trident installation namespace>
    name: trident-node-remediation-template
  minHealthy: 0 # Trigger force-detach upon one or more node failures
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 0s
    - type: Ready
      status: Unknown
      duration: 0s
```

2. Apply the node health check CR in the `trident` namespace.

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

The above CR is configured to watch K8s worker nodes for node conditions Ready: false and Unknown. Automated-Failover will be triggered upon a node going into Ready: false, or Ready: Unknown state.

The `unhealthyConditions` in the CR uses a 0 second grace period. This causes automated-failover to trigger immediately upon K8s setting node condition Ready: false, which is set after K8s loses the heartbeat from a node. K8s has a default of 40sec wait after the last heartbeat before setting Ready: false. This grace-period can be customized in K8s deployment options.

For additional configuration options, refer to Node-Healthcheck-Operator documentation.

Additional setup information

When Trident is installed with force-detach enabled, two additional resources are automatically created in the Trident namespace to facilitate integration with NHC: TridentNodeRemediationTemplate (TNRT) and ClusterRole.

TridentNodeRemediationTemplate (TNRT):

The TNRT serves as a template for the NHC controller, which uses TNRT to generate TNR resources as needed.

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

ClusterRole:

A cluster role is also added during the installation when force-detach is enabled. This gives NHC permissions to TNRs in the Trident namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.ext-remediation/aggregate-to-ext-remediation: "true"
  name: tridentnoderemediation-access
rules:
- apiGroups:
  - trident.netapp.io
  resources:
  - tridentnoderemediationtemplates
  - tridentnoderemediations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

K8s cluster upgrades and maintenance

To prevent any failovers, pause automated-failover during K8s maintenance or upgrades, where the nodes are expected to go down or reboot. You can pause the NHC CR (described above) by patching its CR:

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

This pauses the automated-failover. To re-enable automated-failover, remove the pauseRequests from the spec after the maintenance is complete.

Limitations

- I/O operations are prevented only on the failed nodes for volumes supported by force-detach. Only pods using volumes/PVCs supported by force-detach are automatically removed.
- Automatic-failover and force-detach run inside the trident-controller pod. If the node hosting trident-controller fails, automated-failover will be delayed until K8s moves the pod to a healthy node.

Integrating custom node health check solutions

You can replace Node Healthcheck Operator with alternative node failure detection tools to trigger automatic-failover.
To ensure compatibility with the automated failover mechanism, your custom solution should:

- Create a TNR when a node failure is detected, using the failed node’s name as the TNR CR name.
- Delete the TNR when the node has recovered and the TNR is in the Succeeded state.

Security

Security

Use the recommendations listed here to ensure your Trident installation is secure.

Run Trident in its own namespace

It is important to prevent applications, application administrators, users, and management applications from accessing Trident object definitions or the pods to ensure reliable storage and block potential malicious activity.

To separate the other applications and users from Trident, always install Trident in its own Kubernetes namespace (`trident`). Putting Trident in its own namespace assures that only the Kubernetes administrative personnel have access to the Trident pod and the artifacts (such as backend and CHAP secrets if applicable) stored in the namespaced CRD objects.
You should ensure that you allow only administrators access to the Trident namespace and thus access to the `tridentctl` application.

Use CHAP authentication with ONTAP SAN backends

Trident supports CHAP-based authentication for ONTAP SAN workloads (using the `ontap-san` and `ontap-san-economy` drivers). NetApp recommends using bidirectional CHAP with Trident for authentication between a host and the storage backend.

For ONTAP backends that use the SAN storage drivers, Trident can set up bidirectional CHAP and manage CHAP usernames and secrets through `tridentctl`.
Refer to Prepare to configure backend with ONTAP SAN drivers to understand how Trident configures CHAP on ONTAP backends.

Use CHAP authentication with NetApp HCI and SolidFire backends

NetApp recommends deploying bidirectional CHAP to ensure authentication between a host and the NetApp HCI and SolidFire backends. Trident uses a secret object that includes two CHAP passwords per tenant. When Trident is installed, it manages the CHAP secrets and stores them in a `tridentvolume` CR object for the respective PV. When you create a PV, Trident uses the CHAP secrets to initiate an iSCSI session and communicate with the NetApp HCI and SolidFire system over CHAP.

The volumes that are created by Trident are not associated with any Volume Access Group.

Use Trident with NVE and NAE

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

- If NAE is enabled on the backend, any volume provisioned in Trident will be NAE-enabled.
  - You can set the NVE encryption flag to `""` to create NAE-enabled volumes.
- If NAE is not enabled on the backend, any volume provisioned in Trident will be NVE-enabled unless the NVE encryption flag is set to `false` (the default value) in the backend configuration.

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

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

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

For more information on backend configuration options, refer to:

- ONTAP SAN configuration options
- ONTAP NAS configuration options

Linux Unified Key Setup (LUKS)

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

In Trident, LUKS-encrypted volumes use the aes-xts-plain64 cypher and mode, as recommended by NIST.

LUKS encryption is not supported for ASA r2 systems. For information about ASA r2 systems, see Learn about ASA r2 storage systems.

Before you begin

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

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

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

Enable LUKS encryption

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

Steps

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

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

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

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

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

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

Limitations

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

Backend configuration for importing LUKS volumes

To import a LUKS volume, you must set `luksEncryption` to `true` on the backend. The `luksEncryption` option tells Trident if the volume is LUKS-compliant (`true`) or not LUKS-compliant (`false`) as shown in the following example.

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

PVC configuration for importing LUKS volumes

To import LUKS volumes dynamically, set the annotation `trident.netapp.io/luksEncryption` to `true` and include a LUKS-enabled storage class in the PVC as shown in this example.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

Rotate a LUKS passphrase

You can rotate the LUKS passphrase and confirm rotation.

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

About this task

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

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

Steps

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

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

2. Identify existing passphrases on the volume or snapshot.

   Volume

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

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

   Snapshot

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

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

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

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

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

Volume

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

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

Snapshot

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

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

Results

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

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

Enable volume expansion

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

Steps

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

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

Results

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

Kerberos in-flight encryption

Using Kerberos in-flight encryption, you can improve data access security by enabling encryption for the traffic between your managed cluster and the storage backend.

Trident supports Kerberos encryption for ONTAP as a storage backend:

- **On-premise ONTAP** - Trident supports Kerberos encryption over NFSv3 and NFSv4 connections from Red Hat OpenShift and upstream Kubernetes clusters to on-premise ONTAP volumes.

You can create, delete, resize, snapshot, clone, read-only clone, and import volumes that use NFS encryption.

Configure in-flight Kerberos encryption with on-premise ONTAP volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and an on-premise ONTAP storage backend.

Kerberos encryption for NFS traffic with on-premise ONTAP storage backends is only supported using the `ontap-nas` storage driver.

Before you begin

- Ensure that you have access to the `tridentctl` utility.
- Ensure you have administrator access to the ONTAP storage backend.
- Ensure you know the name of the volume or volumes you will be sharing from the ONTAP storage backend.
- Ensure that you have prepared the ONTAP storage VM to support Kerberos encryption for NFS volumes. Refer to Enable Kerberos on a dataLIF for instructions.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Add or modify ONTAP export policies

You need to add rules to existing ONTAP export policies or create new export polices that support Kerberos encryption for the ONTAP storage VM root volume as well as any ONTAP volumes shared with the upstream Kubernetes cluster. The export policy rules you add, or new export policies you create, need to support the following access protocols and access permissions:

Access protocols

Configure the export policy with NFS, NFSv3, and NFSv4 access protocols.

Access details

You can configure one of three different versions of Kerberos encryption, depending on your needs for the volume:

- **Kerberos 5** - (authentication and encryption)
- **Kerberos 5i** - (authentication and encryption with identity protection)
- **Kerberos 5p** - (authentication and encryption with identity and privacy protection)

Configure the ONTAP export policy rule with the appropriate access permissions. For example, if clusters will be mounting the NFS volumes with a mixture of Kerberos 5i and Kerberos 5p encryption, use the following access settings:

Type	Read-only access	Read/Write access	Superuser access
UNIX	Enabled	Enabled	Enabled
Kerberos 5i	Enabled	Enabled	Enabled
Kerberos 5p	Enabled	Enabled	Enabled

Refer to the following documentation for how to create ONTAP export policies and export policy rules:

- Create an export policy
- Add a rule to an export policy

Create a storage backend

You can create a Trident storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can specify one of three different versions of Kerberos encryption using the `spec.nfsMountOptions` parameter:

- `spec.nfsMountOptions: sec=krb5` (authentication and encryption)
- `spec.nfsMountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `spec.nfsMountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used.

Steps

1. On the managed cluster, create a storage backend configuration file using the following example. Replace values in brackets <> with information from your environment:

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. Use the configuration file you created in the previous step to create the backend:

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

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

```
tridentctl logs
```

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

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

About this task

When you create a storage class object, you can specify one of three different versions of Kerberos encryption using the `mountOptions` parameter:

- `mountOptions: sec=krb5` (authentication and encryption)
- `mountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `mountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used. If the level of encryption you specified in the storage backend configuration is different than the level you specify in the storage class object, the storage class object takes precedence.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc ontap-nas-sc
```

You should see output similar to the following:

```
NAME            PROVISIONER               AGE
ontap-nas-sc    csi.trident.netapp.io    15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions, refer to Provision a volume.

Configure in-flight Kerberos encryption with Azure NetApp Files volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and a single Azure NetApp Files storage backend or a virtual pool of Azure NetApp Files storage backends.

Before you begin

- Ensure that you have enabled Trident on the managed Red Hat OpenShift cluster.
- Ensure that you have access to the `tridentctl` utility.
- Ensure that you have prepared the Azure NetApp Files storage backend for Kerberos encryption by noting the requirements and following the instructions in Azure NetApp Files documentation.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Create a storage backend

You can create an Azure NetApp Files storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can define it so that it should be applied at one of two possible levels:

- The **storage backend level** using the `spec.kerberos` field
- The **virtual pool level** using the `spec.storage.kerberos` field

When you define the configuration at the virtual pool level, the pool is selected using the label in the storage class.

At either level, you can specify one of three different versions of Kerberos encryption:

- `kerberos: sec=krb5` (authentication and encryption)
- `kerberos: sec=krb5i` (authentication and encryption with identity protection)
- `kerberos: sec=krb5p` (authentication and encryption with identity and privacy protection)

Steps

1. On the managed cluster, create a storage backend configuration file using one of the following examples, depending on where you need to define the storage backend (storage backend level or virtual pool level). Replace values in brackets <> with information from your environment:

Storage backend level example

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

Virtual pool level example

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

2. Use the configuration file you created in the previous step to create the backend:

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

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

```
tridentctl logs
```

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

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc -sc-nfs
```

You should see output similar to the following:

```
NAME         PROVISIONER              AGE
sc-nfs       csi.trident.netapp.io    15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions, refer to Provision a volume.