Use Trident

Trident

NetApp
June 30, 2026

Table of Contents

Use Trident

Prepare the worker node

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

Selecting the right tools

If you are using a combination of drivers, you should install all required tools for your drivers. Recent versions of Red Hat Enterprise Linux CoreOS (RHCOS) have the tools installed by default.

NFS tools

Install the NFS tools if you are using: `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, or `azure-netapp-files`.

iSCSI tools

Install the iSCSI tools if you are using: `ontap-san`, `ontap-san-economy`, `solidfire-san`.

NVMe tools

Install the NVMe tools if you are using `ontap-san` for nonvolatile memory express (NVMe) over TCP (NVMe/TCP) protocol.

> NetApp recommends ONTAP 9.12 or later for NVMe/TCP.

SCSI over FC tools

Refer to Ways to configure FC & FC-NVMe SAN hosts for more information about configuring your FC and FC-NVMe SAN hosts.

Install the FC tools if you are using `ontap-san` with sanType `fcp` (SCSI over FC).

Points to consider:
* SCSI over FC is supported on OpenShift and KubeVirt environments.
* SCSI over FC is not supported on Docker.
* iSCSI self healing is not applicable to SCSI over FC.

SMB tools

Prepare to provision SMB volumes if you are using: `ontap-nas` to provision SMB volumes.

Node service discovery

Trident attempts to automatically detect if the node can run iSCSI or NFS services.

> Node service discovery identifies discovered services but does not guarantee services are properly configured. Conversely, the absence of a discovered service does not guarantee the volume mount will fail.

Review events

Trident creates events for the node to identify the discovered services. To review these events, run:

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

Review discovered services

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

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

NFS volumes

Install the NFS tools using the commands for your operating system. Ensure the NFS service is started up during boot time.

RHEL 8+

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

Reboot your worker nodes after installing the NFS tools to prevent failure when attaching volumes to containers.

iSCSI volumes

Trident can automatically establish an iSCSI session, scan LUNs, and discover multipath devices, format them, and mount them to a pod.

iSCSI self-healing capabilities

For ONTAP systems, Trident runs iSCSI self-healing every five minutes to:

1. **Identify** the desired iSCSI session state and the current iSCSI session state.
2. **Compare** the desired state to the current state to identify needed repairs. Trident determines repair priorities and when to preempt repairs.
3. **Perform repairs** required to return the current iSCSI session state to the desired iSCSI session state.

Logs of self-healing activity are located in the `trident-main` container on the respective Daemonset pod. To view logs, you must have set `debug` to "true" during Trident installation.

Trident iSCSI self-healing capabilities can help prevent:

- Stale or unhealthy iSCSI sessions that could occur after a network connectivity issue. In the case of a stale session, Trident waits seven minutes before logging out to reestablish the connection with a portal.

> For example, if CHAP secrets were rotated on the storage controller and the network loses connectivity, the old (*stale*) CHAP secrets could persist. Self-healing can recognize this and automatically reestablish the session to apply the updated CHAP secrets.

- Missing iSCSI sessions
- Missing LUNs

Points to consider before upgrading Trident

- If only per-node igroups (introduced in 23.04+) are in use, iSCSI self-healing will initiate SCSI rescans for all devices in the SCSI bus.
- If only backend-scoped igroups (deprecated as of 23.04) are in use, iSCSI self-healing will initiate SCSI rescans for exact LUN IDs in the SCSI bus.
- If a mix of per-node igroups and backend-scoped igroups are in use, iSCSI self-healing will initiate SCSI rescans for exact LUN IDs in the SCSI bus.

Install the iSCSI tools

Install the iSCSI tools using the commands for your operating system.

Before you begin

- Each node in the Kubernetes cluster must have a unique IQN. **This is a necessary prerequisite**.
- If using RHCOS version 4.5 or later, or other RHEL-compatible Linux distribution, with the `solidfire-san` driver and Element OS 12.5 or earlier, ensure that the CHAP authentication algorithm is set to MD5 in `/etc/iscsi/iscsid.conf`. Secure FIPS-compliant CHAP algorithms SHA1, SHA-256, and SHA3-256 are available with Element 12.7.

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

- When using worker nodes that run RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) with iSCSI PVs, specify the `discard` mountOption in the StorageClass to perform inline space reclamation. Refer to Red Hat documentation.
- Ensure that you have upgraded to the latest version of the `multipath-tools`.

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi iscsi-initiator-utils device-mapper-
multipath
```

2. Check that iscsi-initiator-utils version is 6.2.0.874-2.el7 or later:

```
rpm -q iscsi-initiator-utils
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `iscsid` and `multipathd` are running:

```
sudo systemctl enable --now iscsid multipathd
```

6. Enable and start `iscsi`:

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. Install the following system packages:

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools
scsitools
```

2. Check that open-iscsi version is 2.0.874-5ubuntu2.10 or later (for bionic) or 2.0.874-7.1ubuntu6.1 or later (for focal):

```
dpkg -l open-iscsi
```

3. Set scanning to manual:

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

4. Enable multipathing:

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

5. Ensure that `open-iscsi` and `multipath-tools` are enabled and running:

```
sudo systemctl status multipath-tools
sudo systemctl enable --now open-iscsi.service
sudo systemctl status open-iscsi
```

> For Ubuntu 18.04, you must discover target ports with `iscsiadm` before starting `open-iscsi` for the iSCSI daemon to start. You can alternatively modify the `iscsi` service to start `iscsid` automatically.

Configure or disable iSCSI self healing

You can configure the following Trident iSCSI self-healing settings to fix stale sessions:

- **iSCSI self-healing interval**: Determines the frequency at which iSCSI self-healing is invoked (default: 5 minutes). You can configure it to run more frequently by setting a smaller number or less frequently by setting a larger number.

> Setting the iSCSI self-healing interval to 0 stops iSCSI self-healing completely. We do not recommend disabling iSCSI Self-healing; it should only be disabled in certain scenarios when iSCSI self-healing is not working as intended or for debugging purposes.

- **iSCSI Self-Healing Wait Time**: Determines the duration iSCSI self-healing waits before logging out of an unhealthy session and trying to log in again (default: 7 minutes). You can configure it to a larger number so that sessions that are identified as unhealthy have to wait longer before being logged out and then an attempt is made to log back in, or a smaller number to log out and log in earlier.

Helm

To configure or change iSCSI self-healing settings, pass the `iscsiSelfHealingInterval` and `iscsiSelfHealingWaitTime` parameters during the helm installation or helm update.

The following example sets the iSCSI self-healing interval to 3 minutes and self-healing wait time to 6 minutes:

```
helm install trident trident-operator-100.2602.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

tridentctl

To configure or change iSCSI self-healing settings, pass the `iscsi-self-healing-interval` and `iscsi-self-healing-wait-time` parameters during the tridentctl installation or update.

The following example sets the iSCSI self-healing interval to 3 minutes and self-healing wait time to 6 minutes:

```
tridentctl install --iscsi-self-healing-interval=3m0s --iscsi-self
-healing-wait-time=6m0s -n trident
```

NVMe/TCP volumes

Install the NVMe tools using the commands for your operating system.

- NVMe requires RHEL 9 or later.
- If the kernel version of your Kubernetes node is too old or if the NVMe package is not available for your kernel version, you might have to update the kernel version of your node to one with the NVMe package.

RHEL 9

```
sudo yum install nvme-cli
sudo yum install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

Ubuntu

```
sudo apt install nvme-cli
sudo apt -y install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

Verify installation

After installation, verify that each node in the Kubernetes cluster has a unique NQN using the command:

```
cat /etc/nvme/hostnqn
```

Trident modifies the `ctrl_device_tmo` value to ensure NVMe doesn't give up on the path if it goes down. Do not change this setting.

SCSI over FC volumes

You can now use the Fibre Channel (FC) protocol with Trident to provision and manage storage resources on ONTAP system.

Prerequisites

Configure the required network and node settings for FC.

Network settings

1. Get the WWPN of the target interfaces. Refer to network interface show for more information.
2. Get the WWPN for the interfaces on initiator (Host).

   Refer to the corresponding host operating system utilities.

3. Configure zoning on the FC switch using WWPNs of the Host and target.

   Refer to the respecive switch vendor documentation for information.

   Refer to the following ONTAP documentation for details:

   - Fibre Channel and FCoE zoning overview
   - Ways to configure FC & FC-NVMe SAN hosts

Install the FC tools

Install the FC tools using the commands for your operating system.

- When using worker nodes that run RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) with FC PVs, specify the `discard` mountOption in the StorageClass to perform inline space reclamation. Refer to Red Hat documentation.

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. Enable multipathing:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipathd` is running:

```
sudo systemctl enable --now multipathd
```

Ubuntu

1. Install the following system packages:

```
sudo apt-get install -y lsscsi sg3-utils multipath-tools scsitools
```

2. Enable multipathing:

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> Ensure `/etc/multipath.conf` contains `find_multipaths no` under `defaults`.

3. Ensure that `multipath-tools` is enabled and running:

```
sudo systemctl status multipath-tools
```

Prepare to provision SMB volumes

You can provision SMB volumes using `ontap-nas` drivers.

> You must configure both NFS and SMB/CIFS protocols on the SVM to create an `ontap-nas-economy` SMB volume for ONTAP on-premises clusters. Failure to configure either of these protocols will cause SMB volume creation to fail.

> `autoExportPolicy` is not supported for SMB volumes.

Before you begin

Before you can provision SMB volumes, you must have the following.

- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials. To generate secret `smbcreds`:

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Steps

1. For on-premises ONTAP, you can optionally create an SMB share or Trident can create one for you.

   > SMB shares are required for Amazon FSx for ONTAP.

   You can create the SMB admin shares in one of two ways either using the Microsoft Management Console Shared Folders snap-in or using the ONTAP CLI. To create the SMB shares using the ONTAP CLI:

   a. If necessary, create the directory path structure for the share.

      The `vserver cifs share create` command checks the path specified in the -path option during share creation. If the specified path does not exist, the command fails.

   b. Create an SMB share associated with the specified SVM:

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

   c. Verify that the share was created:

```
vserver cifs share show -share-name share_name
```

Refer to Create an SMB share for full details.

2. When creating the backend, you must configure the following to specify SMB volumes. For all FSx for ONTAP backend configuration options, refer to FSx for ONTAP configuration options and examples.

Parameter	Description	Example
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`
`nasType`	**Must set to `smb`.** If null, defaults to `nfs`.	`smb`
`securityStyle`	Security style for new volumes. **Must be set to `ntfs` or `mixed` for SMB volumes.**	`ntfs` or `mixed` for SMB volumes
`unixPermissions`	Mode for new volumes. **Must be left empty for SMB volumes.**	""

Configure and manage backends

Configure backends

A backend defines the relationship between Trident and a storage system. It tells Trident how to communicate with that storage system and how Trident should provision volumes from it.

Trident automatically offers up storage pools from backends that match the requirements defined by a storage class. Learn how to configure the backend for your storage system.

- Configure an Azure NetApp Files backend
- Configure a Google Cloud NetApp Volumes backend
- Configure a NetApp HCI or SolidFire backend
- Configure a backend with ONTAP or Cloud Volumes ONTAP NAS drivers
- Configure a backend with ONTAP or Cloud Volumes ONTAP SAN drivers
- Use Trident with Amazon FSx for NetApp ONTAP

Azure NetApp Files

Configure an Azure NetApp Files backend

Use Azure NetApp Files as a backend for Trident.
This backend supports NFS and SMB volumes.
Trident supports managed identities and workload identity for Azure Kubernetes Service (AKS) clusters.

Supported Azure cloud environments

Trident supports Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When you deploy Trident or configure an Azure NetApp Files backend, ensure that Azure Resource Manager and authentication endpoints match your Azure cloud environment.

Review Azure NetApp Files driver support

Trident provides the following Azure NetApp Files storage driver.

Supported access modes include *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), and *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`azure-netapp-files`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	`nfs`, `smb`

Review considerations

- Azure NetApp Files does not support volumes smaller than 50 GiB.
  Trident creates a 50-GiB volume when a smaller volume is requested.
- Trident supports SMB volumes mounted to pods running on Windows nodes only.
- Azure NetApp Files deployments in non-Commercial Azure clouds require cloud-specific Azure Resource Manager and authentication endpoints.
  Ensure that Trident and any backend configuration use the endpoints appropriate for your Azure cloud environment.

Use managed identities for AKS

Trident supports managed identities for AKS clusters.

If you use `tridentctl` to create or manage Azure NetApp Files backends, ensure that it is configured for the correct Azure cloud environment.

To use managed identities, you must have:

- A Kubernetes cluster deployed using AKS

- Managed identities configured on the AKS Kubernetes cluster
- Trident installed with `cloudProvider` set to `"Azure"`

Trident operator

Edit `tridentorchestrator_cr.yaml` and set `cloudProvider` to `"Azure"`.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
```

Helm

The following example installs Trident and sets `cloudProvider` using the environment variable `$CP`:

```
helm install trident trident-operator-100.2602.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

`tridentctl`

The following example installs Trident and sets the `cloud-provider` flag to `Azure`:

```
tridentctl install --cloud-provider="Azure" -n trident
```

Use workload identity for AKS

Workload identity enables Kubernetes pods to access Azure resources by authenticating as a workload identity.

If you use `tridentctl` to create or manage Azure NetApp Files backends, ensure that it is configured for the correct Azure cloud environment.

To use workload identity, you must have:

- A Kubernetes cluster deployed using AKS
- Workload identity and oidc-issuer configured on the AKS Kubernetes cluster
- Trident installed with `cloudProvider` set to `"Azure"` and `cloudIdentity` set to the workload identity value

Trident operator

Edit `tridentorchestrator_cr.yaml` and set `cloudProvider` to `"Azure"`.
Set `cloudIdentity` to `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
  cloudIdentity: 'azure.workload.identity/client-id: xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxx' # Edit
```

Helm

Set the values for the **cloud-provider (CP)** and **cloud-identity (CI)** flags using the following environment variables:

```
export CP="Azure"
export CI="'azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx'"
```

The following example installs Trident and sets `cloudProvider` using `$CP` and sets `cloudIdentity` using `$CI`:

```
helm install trident trident-operator-100.6.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$CI"
```

`tridentctl`

Set the values for the **cloud provider** and **cloud identity** flags using the following environment variables:

```
export CP="Azure"
export CI="azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx"
```

The following example installs Trident and sets `cloud-provider` to `$CP` and `cloud-identity` to `$CI`:

```
tridentctl install --cloud-provider=$CP --cloud-identity="$CI" -n
trident
```

Prepare to configure an Azure NetApp Files backend

Before you can configure your Azure NetApp Files backend, you need to ensure the following requirements are met.

Supported Azure cloud environments

Trident supports Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When preparing your environment, ensure that your Azure subscription, identity configuration, and Azure NetApp Files resources are created in the appropriate Azure cloud environment.

Prerequisites for NFS and SMB volumes

If you are using Azure NetApp Files for the first time or in a new location, some initial configuration is required to set up Azure NetApp Files and create an NFS volume. Refer to Azure: Set up Azure NetApp Files and create an NFS volume.

To configure and use an Azure NetApp Files backend, you need the following:

- `subscriptionID`, `tenantID`, `clientID`, `location`, and `clientSecret` are optional when using managed identities on an AKS cluster.
- `tenantID`, `clientID`, and `clientSecret` are optional when using a cloud identity on an AKS cluster.
- Azure NetApp Files deployments in non-Commercial Azure clouds require cloud-specific Azure Resource Manager and authentication endpoints. Ensure that Trident and any backend configuration use the endpoints appropriate for your Azure cloud environment.

- A capacity pool. Refer to Microsoft: Create a capacity pool for Azure NetApp Files.
- A subnet delegated to Azure NetApp Files. Refer to Microsoft: Delegate a subnet to Azure NetApp Files.
- `subscriptionID` from an Azure subscription with Azure NetApp Files enabled.
- `tenantID`, `clientID`, and `clientSecret` from an App Registration in Azure Active Directory with sufficient permissions to the Azure NetApp Files service. The App Registration should use either:
  - The Owner or Contributor role predefined by Azure.
  - A custom Contributor role at the subscription level (`assignableScopes`) with the following permissions that are limited to only what Trident requires. After creating the custom role, assign the role using the Azure portal.

Custom contributor role

```
{
  "id": "/subscriptions/<subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-
definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTarge
ts/read",
          "Microsoft.Network/virtualNetworks/read",
          "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/delete",
```

```
            "Microsoft.Features/features/read",
            "Microsoft.Features/operations/read",
            "Microsoft.Features/providers/features/read",

"Microsoft.Features/providers/features/register/action",

"Microsoft.Features/providers/features/unregister/action",

"Microsoft.Features/subscriptionFeatureRegistrations/read"
          ],
          "notActions": [],
          "dataActions": [],
          "notDataActions": []
        }
      ]
    }
}
```

- The Azure `location` that contains at least one delegated subnet. As of Trident 22.01, the `location` parameter is a required field at the top level of the backend configuration file. Location values specified in virtual pools are ignored.
- To use `Cloud Identity`, get the `client ID` from a user-assigned managed identity and specify that ID in `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`.

Additional requirements for SMB volumes

To create an SMB volume, you must have:

- Active Directory configured and connected to Azure NetApp Files. Refer to Microsoft: Create and manage Active Directory connections for Azure NetApp Files.
- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials so Azure NetApp Files can authenticate to Active Directory. To generate secret `smbcreds`:

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Azure NetApp Files backend configuration options and examples

Learn about NFS and SMB backend configuration options for Azure NetApp Files and review configuration examples.

Backend configuration options

Trident uses your backend configuration (subnet, virtual network, service level, and location) to create Azure NetApp Files volumes on capacity pools that are available in the requested location and match the requested service level and subnet.

Azure NetApp Files backends provide these configuration options.

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

Parameter	Description	Default
`capacityPools`	List of capacity pools for filtering discovered resources	"[]" (no filter, random)
`virtualNetwork`	Name of a virtual network with a delegated subnet	""
`subnet`	Name of a subnet delegated to `Microsoft.Netapp/volumes`	""
`networkFeatures`	Set of VNet features for a volume, may be `Basic` or `Standard`. Network Features is not available in all regions and might have to be enabled in a subscription. Specifying `networkFeatures` when the functionality is not enabled causes volume provisioning to fail.	""
`nfsMountOptions`	Fine-grained control of NFS mount options. Ignored for SMB volumes. To mount volumes using NFS version 4.1, include `nfsvers=4` in the comma-delimited mount options list to choose NFS v4.1. Mount options set in a storage class definition override mount options set in backend configuration.	"nfsvers=3"
`limitVolumeSize`	Fail provisioning if the requested volume size is above this value	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `\{"api": false, "method": true, "discovery": true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes.	`nfs`

Parameter	Description	Default
`supportedTopologies`	Represents a list of regions and zones that are supported by this backend. For more information, refer to Use CSI Topology.	
`qosType`	Represents the QoS type: Auto or Manual.	Auto
`maxThroughput`	Sets the maximum throughput allowed in MiB/sec. Supported only for manual QoS capacity pools.	`4 MiB/sec`

For more information on Network Features, refer to Configure network features for an Azure NetApp Files volume.

Consider Azure cloud environments (26.02)

Starting with the 26.02 release, Trident supports creating and managing Azure NetApp Files backends in multiple Azure cloud environments.

Supported Azure clouds include:

- Azure Commercial
- Azure Government (Azure Government / MAG)

When you deploy Trident or create an Azure NetApp Files backend, ensure that Azure Resource Manager and authentication endpoints match your Azure cloud environment.
If the endpoints do not match, `tridentctl` cannot authenticate and backend creation fails.

Required permissions and resources

If you receive a "No capacity pools found" error when creating a PVC, it is likely your app registration doesn't have the required permissions and resources (subnet, virtual network, capacity pool) associated.
If debug is enabled, Trident logs the Azure resources discovered when the backend is created.
Verify an appropriate role is being used.

The values for `resourceGroups`, `netappAccounts`, `capacityPools`, `virtualNetwork`, and `subnet` can be specified using short or fully-qualified names.
Fully-qualified names are recommended in most situations as short names can match multiple resources with the same name.

If the vNet is located in a different resource group from the Azure NetApp Files (ANF) storage account, specify the resource group for the virtual network while configuring the resourceGroups list for the backend.

The `resourceGroups`, `netappAccounts`, and `capacityPools` values are filters that restrict the set of discovered resources to those available to this storage backend and may be specified in any combination.
Fully-qualified names follow this format:

Type	Format
Resource group	<resource group>
NetApp account	<resource group>/<netapp account>
Capacity pool	<resource group>/<netapp account>/<capacity pool>
Virtual network	<resource group>/<virtual network>
Subnet	<resource group>/<virtual network>/<subnet>

Volume provisioning

You can control default volume provisioning by specifying the following options in a special section of the configuration file.
Refer to Example configurations for details.

Parameter	Description	Default
`exportRule`	Export rules for new volumes. `exportRule` must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation. Ignored for SMB volumes.	"0.0.0.0/0"
`snapshotDir`	Access to the `.snapshot` directory	`true`, `false` (Set explicitly).
`size`	The default size of new volumes	"100G"
`unixPermissions`	The unix permissions of new volumes (4 octal digits). Ignored for SMB volumes.	"" (preview feature, requires whitelisting in subscription)

Example configurations

The following examples show basic configurations that leave most parameters to default.
This is the easiest way to define a backend.

Minimal configuration

This is the absolute minimum backend configuration.
With this configuration, Trident discovers all of your NetApp accounts, capacity pools, and subnets delegated to Azure NetApp Files in the configured location, and places new volumes on one of those pools and subnets randomly.
Because `nasType` is omitted, the `nfs` default applies and the backend will provision for NFS volumes.

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

```
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
  tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
  clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
  clientSecret: SECRET
  location: eastus
```

Managed identities for AKS

This backend configuration omits `subscriptionID`, `tenantID`, `clientID`, and `clientSecret`, which are optional when using managed identities.

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - resource-group-1/netapp-account-1/ultra-pool
  resourceGroups:
    - resource-group-1
  netappAccounts:
    - resource-group-1/netapp-account-1
  virtualNetwork: resource-group-1/eastus-prod-vnet
  subnet: resource-group-1/eastus-prod-vnet/eastus-anf-subnet
```

Cloud identity for AKS

This backend configuration omits `tenantID`, `clientID`, and `clientSecret`, which are optional when using a cloud identity.

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
  location: eastus
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
```

Specific service level configuration with capacity pool filters

This backend configuration places volumes in Azure's `eastus` location in an `Ultra` capacity pool. Trident automatically discovers all of the subnets delegated to Azure NetApp Files in that location and places a new volume on one of them randomly.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
```

Backend example with manual QoS capacity pools

This backend configuration places volumes in Azure's `eastus` location with manual QoS capacity pools.

```
---
version: 1
storageDriverName: azure-netapp-files
backendName: anf1
location: eastus
labels:
  clusterName: test-cluster-1
  cloud: anf
  nasType: nfs
defaults:
  qosType: Manual
storage:
  - serviceLevel: Ultra
    labels:
      performance: gold
    defaults:
      maxThroughput: 10
  - serviceLevel: Premium
    labels:
      performance: silver
    defaults:
      maxThroughput: 5
  - serviceLevel: Standard
    labels:
      performance: bronze
    defaults:
      maxThroughput: 3
```

Advanced configuration

This backend configuration further reduces the scope of volume placement to a single subnet, and also modifies some volume provisioning defaults.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
virtualNetwork: application-group-1/eastus-prod-vnet
subnet: application-group-1/eastus-prod-vnet/my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: "true"
  size: 200Gi
  unixPermissions: "0777"
```

Virtual pool configuration

This backend configuration defines multiple storage pools in a single file.
This is useful when you have multiple capacity pools supporting different service levels and you want to create storage classes in Kubernetes that represent those.
Virtual pool labels were used to differentiate the pools based on `performance`.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
resourceGroups:
  - application-group-1
networkFeatures: Basic
nfsMountOptions: vers=3,proto=tcp,timeo=600
labels:
  cloud: azure
storage:
  - labels:
      performance: gold
    serviceLevel: Ultra
    capacityPools:
      - application-group-1/netapp-account-1/ultra-1
      - application-group-1/netapp-account-1/ultra-2
    networkFeatures: Standard
  - labels:
      performance: silver
    serviceLevel: Premium
    capacityPools:
      - application-group-1/netapp-account-1/premium-1
  - labels:
      performance: bronze
    serviceLevel: Standard
    capacityPools:
      - application-group-1/netapp-account-1/standard-1
      - application-group-1/netapp-account-1/standard-2
```

Supported topologies configuration

Trident facilitates provisioning of volumes for workloads based on regions and availability zones.
The `supportedTopologies` block in this backend configuration is used to provide a list of regions and zones per backend.
The region and zone values specified here must match the region and zone values from the labels on each Kubernetes cluster node.
These regions and zones represent the list of permissible values that can be provided in a storage class.
For storage classes that contain a subset of the regions and zones provided in a backend, Trident creates volumes in the mentioned region and zone.
For more information, refer to Use CSI Topology.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
supportedTopologies:
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-1
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-2
```

Storage class definitions

The following `StorageClass` definitions refer to the storage pools above.

Example definitions using `parameter.selector` field

Using `parameter.selector` you can specify for each `StorageClass` the virtual pool that is used to host a volume.
The volume will have the aspects defined in the chosen pool.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: silver
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: bronze
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze
allowVolumeExpansion: true
```

Example definitions for SMB volumes

Using `nasType`, `node-stage-secret-name`, and `node-stage-secret-namespace`, you can specify an SMB volume and provide the required Active Directory credentials.

Basic configuration on default namespace

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

Using different secrets per namespace

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

Using different secrets per volume

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` filters for pools which support SMB volumes.
`nasType: nfs` or `nasType: null` filters for NFS pools.

Create the backend

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

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

If you use a non-Commercial Azure cloud, ensure that `tridentctl` is configured to use the Azure Resource Manager and authentication endpoints for your Azure cloud environment.
If the backend creation fails, check your backend configuration and view the logs to determine the cause:

```
tridentctl logs
```

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

Google Cloud NetApp Volumes

Configure Google Cloud NetApp Volumes

You can configure Google Cloud NetApp Volumes as a backend for Trident to provision storage for Kubernetes workloads.

Overview

Trident supports Google Cloud NetApp Volumes for both NAS (NFS and SMB) and block (iSCSI) workloads.

- NAS workloads use the `google-cloud-netapp-volumes` backend
- Block (iSCSI) workloads use the `google-cloud-netapp-volumes-san` backend

NAS volumes provide file-based storage and are accessed using NFS or SMB protocols. These volumes support shared access across multiple pods or nodes.

Block volumes provide raw block storage and are accessed as iSCSI devices attached to Kubernetes nodes. These volumes are used when applications require block-level access.

This applies to the following environments:

- Trident 26.02 and later
- Google Kubernetes Engine (GKE) or Red Hat OpenShift
- Google Cloud NetApp Volumes storage pools

To configure block (iSCSI) storage, see
Configure block storage (iSCSI).

Prepare to configure

Cloud identity enables Kubernetes workloads to access Google Cloud resources by authenticating as a workload identity instead of using static credentials.

To use cloud identity with Google Cloud NetApp Volumes, you must have:

- A Kubernetes cluster deployed using Google Kubernetes Engine (GKE)
- Workload identity enabled on the GKE cluster and the metadata server enabled on the node pools
- A Google Cloud service account with the Google Cloud NetApp Volumes Admin role (`roles/netapp.admin`) or an equivalent custom role
- Trident installed with the cloud provider set to `GCP` and the cloud identity annotation configured

Trident operator

To install Trident using the Trident operator, edit `tridentorchestrator_cr.yaml`:

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  cloudProvider: "GCP"
  cloudIdentity: "iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com"
```

Helm

Set the cloud provider and cloud identity when installing Trident with Helm:

```
helm install trident trident-operator-100.6.0.tgz \
  --set cloudProvider=GCP \
  --set cloudIdentity="iam.gke.io/gcp-service-account: cloudvolumes-
admin-sa@mygcpproject.iam.gserviceaccount.com"
```

tridentctl

Install Trident by specifying the cloud provider and cloud identity:

```
tridentctl install \
  --cloud-provider=GCP \
  --cloud-identity="iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com" \
  -n trident
```

Configure NAS storage

For Google Cloud NetApp Volumes UNIFIED storage pools, Trident applies UNIFIED-specific naming and validation rules during volume operations.

When locating a volume, Trident can evaluate multiple compatible volume name variants (for example, hyphen and underscore formats) to improve import and discovery reliability.

Driver details

Trident provides the `google-cloud-netapp-volumes` driver to provision NAS storage from Google Cloud NetApp Volumes.

The driver supports the following access modes:

- ReadWriteOnce (RWO)
- ReadOnlyMany (ROX)
- ReadWriteMany (RWX)
- ReadWriteOncePod (RWOP)

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`google-cloud-netapp-volumes`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	`nfs`, `smb`

Configure a Trident NAS backend

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
    network: "<vpc-network>"
```

Provision NAS volumes

NAS volumes are provisioned using the `google-cloud-netapp-volumes` backend and support NFS and

SMB protocols.

StorageClass for NFS volumes

To provision NFS volumes, set `nasType` to `nfs`.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "nfs"
allowVolumeExpansion: true
```

StorageClass for SMB volumes

To provision SMB volumes, set `nasType` to `smb` and provide credentials.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
allowVolumeExpansion: true
```

PersistentVolumeClaim example (RWX)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwx
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

PersistentVolumeClaim example (RWO)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

NAS volumes use `volumeMode: Filesystem`.

Configure Google Cloud NetApp Volumes for SAN workloads

You can configure Trident to provision block storage volumes using the iSCSI protocol from Google Cloud NetApp Volumes. SAN volumes are provisioned from Flex Unified storage pools by using the `google-cloud-netapp-volumes-san` storage driver.

This driver is dedicated to block workloads and does not support NAS protocols.

The `google-cloud-netapp-volumes-san` backend is required to provision iSCSI block volumes. The `google-cloud-netapp-volumes` backend supports NAS protocols only and cannot be used for SAN workloads.

Overview

Trident supports Google Cloud NetApp Volumes SAN (iSCSI) workloads using the `google-cloud-netapp-volumes-san` driver.

SAN volumes are provisioned from Flex Unified storage pools and presented to Kubernetes nodes as iSCSI block devices.

This applies to the following environments:

- Trident 26.02 and later
- Google Kubernetes Engine (GKE) or Red Hat OpenShift
- Google Cloud NetApp Volumes Flex Unified storage pools
- iSCSI-based workloads

Flex Unified storage pools

Flex Unified storage pools provide block storage using the iSCSI protocol and are required for SAN provisioning:

- Flex Unified REGIONAL pools are supported.
- Flex Unified ZONAL pools are supported starting with Trident 26.02.1.
- Only the **Flex** service level is supported for SAN workloads.

Configure a Trident SAN backend

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-san
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
      performance: flex
    network: "<vpc-network>"
    serviceLevel: Flex
```

Create a StorageClass

After configuring the SAN backend, create a StorageClass that references the `google-cloud-netapp-volumes-san` driver.

The filesystem type is defined in the StorageClass, not in the backend.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

Supported filesystem types:

- `ext4` (default)
- `ext3`
- `xfs`

> The SAN driver supports only the Flex service level and does not use NAS-specific backend parameters such as `exportRule`, `unixPermissions`, `nasType`, `snapshotDir`, `nfsMountOptions`, or tiering-related settings.

Provision block volumes

ReadWriteOnce (RWO)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadWriteOncePod (RWOP)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwop
spec:
  accessModes:
    - ReadWriteOncePod
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadOnlyMany (ROX)

A common pattern for ROX is to clone an existing ReadWriteOnce volume and mount the clone as read-only.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rox
spec:
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
  dataSource:
    kind: PersistentVolumeClaim
    name: gcnv-san-rwo
```

ReadWriteMany (RWX) — raw block only

ReadWriteMany is supported only when `volumeMode: Block`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-raw-rwx
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

Block volume behavior

Block volumes are provisioned as iSCSI LUNs and presented to Kubernetes nodes as block devices.

Block volumes:

- Use the iSCSI protocol
- Support filesystem and raw block presentation
- Are attached and managed by Trident
- Support multiple Kubernetes access modes

Access modes

Block volumes provisioned by Trident support the following access modes:

- `ReadWriteOnce` (RWO)
- `ReadOnlyMany` (ROX)
- `ReadWriteOncePod` (RWOP)
- `ReadWriteMany` (RWX), supported only when `volumeMode: Block`

volumeMode behavior

The `volumeMode` field controls how a block volume is exposed:

- `Filesystem`
  Trident formats and mounts the volume.
- `Block`
  Trident attaches the device and exposes it as a raw block device.

Supported operations

Block volumes provisioned using the `google-cloud-netapp-volumes-san` driver support:

- Create

- Delete
- Clone
- Snapshot
- Resize
- Import

Extra GiB overprovisioning behavior

Google Cloud NetApp Volumes block volumes include internal metadata overhead. This overhead reduces the kernel-visible device size compared to the provisioned capacity.

Testing shows:

- Approximately 300 KiB overhead on initial creation
- Up to approximately 107 MiB overhead after a resize

Because Google Cloud NetApp Volumes accepts only whole-GiB allocations, Trident ensures that the usable device size always meets or exceeds the PVC request by:

- Rounding the requested size up to the next whole GiB
- Adding an additional 1 GiB buffer

Example:

- PVC request: 100 GiB
- Provisioned size in Google Cloud NetApp Volumes: 101 GiB
- Usable space visible to the application: at least 100 GiB

Pod examples

Filesystem-mounted block volume (RWO)

```
apiVersion: v1
kind: Pod
metadata:
  name: app-rwo
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeMounts:
    - name: data
      mountPath: /mnt/data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-rwo
```

Raw block device (RWX)

```
apiVersion: v1
kind: Pod
metadata:
  name: app-raw-rwx
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeDevices:
    - name: data
      devicePath: /dev/xda
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-raw-rwx
```

Attach and mount behavior

For SAN volumes provisioned from Google Cloud NetApp Volumes:

- Trident creates a Logical Unit Number (LUN) in a Flex Unified storage pool.
- During publish, Trident maps the LUN to a per-node host group.
- During node staging, Trident:
  - Logs in to the iSCSI target

    - Discovers the LUN
    - Configures multipath
- If `volumeMode: Filesystem`, Trident formats the device if required and mounts it.
- If `volumeMode: Block`, Trident attaches the device and exposes it directly to the pod without formatting or mounting.

> SAN block volumes do not provide distributed locking or write coordination. When a block volume is accessed by multiple nodes (ReadWriteMany with `volumeMode: Block`), the application or filesystem must manage concurrency.

Prepare to configure a Google Cloud NetApp Volumes backend

Before you can configure your Google Cloud NetApp Volumes backend, you need to ensure the following requirements are met.

Prerequisites for NFS or SMB volumes

If you are using Google Cloud NetApp Volumes for the first time or in a new location, some initial configuration is required to set up Google Cloud NetApp Volumes and create an NFS or SMB volume. Refer to Before you begin.

Ensure that you have the following before configuring Google Cloud NetApp Volumes backend:

- A Google Cloud account configured with Google Cloud NetApp Volumes service. Refer to Google Cloud NetApp Volumes.
- Project number of your Google Cloud account. Refer to Identifying projects.
- A Google Cloud service account with the NetApp Volumes Admin (`roles/netapp.admin`) role. Refer to Identity and Access Management roles and permissions.
- API key file for your GCNV account. Refer to Create a service account key
- A storage pool. Refer to Storage pools overview .

For more information about how to set up access to Google Cloud NetApp Volumes, refer to Set up access to Google Cloud NetApp Volumes.

Google Cloud NetApp Volumes backend configuration options and examples

Learn about backend configuration options for Google Cloud NetApp Volumes and review configuration examples.

Backend configuration options

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

Parameter	Description	Default
`version`		Always 1

Parameter	Description	Default
`storageDriverName`	Name of the storage driver	The value of `storageDriverName` must be specified as "google-cloud-netapp-volumes".
`backendName`	(Optional) Custom name of the storage backend	Driver name + "_" + part of API key
`storagePools`	Optional parameter used to specify storage pools for volume creation.	
`projectNumber`	Google Cloud account project number. The value is found on the Google Cloud portal home page.	
`location`	The Google Cloud location where Trident creates GCNV volumes. When creating cross-region Kubernetes clusters, volumes created in a `location` can be used in workloads scheduled on nodes across multiple Google Cloud regions. Cross-region traffic incurs an additional cost.	
`apiKey`	API key for the Google Cloud service account with the `netapp.admin` role. It includes the JSON-formatted contents of a Google Cloud service account's private key file (copied verbatim into the backend configuration file). The `apiKey` must include key-value pairs for the following keys: `type`, `project_id`, `client_email`, `client_id`, `auth_uri`, `token_uri`, `auth_provider_x509_cert_url`, and `client_x509_cert_url`.	
`nfsMountOptions`	Fine-grained control of NFS mount options.	"nfsvers=3"
`limitVolumeSize`	Fail provisioning if the requested volume size is above this value.	"" (not enforced by default)
`serviceLevel`	The service level of a storage pool and its volumes. The values are `flex`, `standard`, `premium`, or `extreme`.	
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`network`	Google Cloud network used for GCNV volumes.	
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `{"api":false, "method":true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null

Parameter	Description	Default
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes.	`nfs`

`supportedTopologies`	Represents a list of regions and zones that are supported by this backend.
	
	For more information, refer to Use CSI Topology.
	For example:
	`supportedTopologies:`
	`- topology.kubernetes.io/region: asia-east1`
	`topology.kubernetes.io/zone: asia-east1-a`	

Volume provisioning options

You can control default volume provisioning in the `defaults` section of the configuration file.

Parameter	Description	Default
`exportRule`	The export rules for new volumes. Must be a comma-separated list of any combination of IPv4 addresses.	"0.0.0.0/0"
`snapshotDir`	Access to the `.snapshot` directory	`true`, `false` (default behavior might vary. Set explicitly) "false" for NFSv3
`snapshotReserve`	Percentage of volume reserved for snapshots	"" (accept default of 0)
`unixPermissions`	The unix permissions of new volumes (4 octal digits).	""

Example configurations

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

Minimal configuration

This is the absolute minimum backend configuration. With this configuration, Trident discovers all of your storage pools delegated to Google Cloud NetApp Volumes in the configured location, and places new volumes on one of those pools randomly. Because `nasType` is omitted, the `nfs` default applies and the backend will provision for NFS volumes.

This configuration is ideal when you are just getting started with Google Cloud NetApp Volumes and trying things out, but in practice you might need to provide additional scoping for the volumes you provision.

Replace `<id_value>` and `<key_value>` with your service account credentials.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

Configuration for SMB volumes

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv1
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123456789"
  location: asia-east1
  serviceLevel: flex
  nasType: smb
  apiKey:
    type: service_account
    project_id: cloud-native-data
    client_email: trident-sample@cloud-native-
data.iam.gserviceaccount.com
    client_id: "123456789737813416734"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/trident-
sample%40cloud-native-data.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

Configuration with StoragePools filter

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  storagePools:
    - premium-pool1-europe-west6
    - premium-pool2-europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

Virtual pool configuration

This backend configuration defines multiple virtual pools in a single file. Virtual pools are defined in the `storage` section. They are useful when you have multiple storage pools supporting different service levels and you want to create storage classes in Kubernetes that represent those. Virtual pool labels are used to differentiate the pools. For instance, in the example below `performance` label and `serviceLevel` type is used to differentiate virtual pools.

You can also set some default values to be applicable to all virtual pools, and overwrite the default values for individual virtual pools. In the following example, `snapshotReserve` and `exportRule` serve as defaults for all virtual pools.

For more information, refer to Virtual pools.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
```

```
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

Cloud identity for GKE

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcp-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '012345678901'
  network: gcnv-network
  location: us-west2
  serviceLevel: Premium
  storagePool: pool-premium1
```

Supported topologies configuration

Trident facilitates provisioning of volumes for workloads based on regions and availability zones. The `supportedTopologies` block in this backend configuration is used to provide a list of regions and zones per backend. The region and zone values specified here must match the region and zone values from the labels on each Kubernetes cluster node. These regions and zones represent the list of permissible values that can be provided in a storage class. For storage classes that contain a subset of the regions and zones provided in a backend, Trident creates volumes in the mentioned region and zone. For more information, refer to Use CSI Topology.

```
---
version: 1
storageDriverName: google-cloud-netapp-volumes
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: asia-east1
serviceLevel: flex
supportedTopologies:
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-a
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-b
```

What's next?

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

```
kubectl create -f <backend-file>
```

To verify that the backend is successfully created, run the following command:

```
kubectl get tridentbackendconfig

NAME                BACKEND NAME        BACKEND UUID
PHASE   STATUS
backend-tbc-gcnv    backend-tbc-gcnv    b2fd1ff9-b234-477e-88fd-713913294f65
Bound   Success
```

If the backend creation fails, something is wrong with the backend configuration. You can describe the backend using the `kubectl get tridentbackendconfig <backend-name>` command or 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 delete the backend and run the create command again.

Storage class definitions

The following is a basic `StorageClass` definition that refers to the backend above.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
```

Example definitions using the `parameter.selector` field:

Using `parameter.selector` you can specify for each `StorageClass` the virtual pool that is used to host a volume. The volume will have the aspects defined in the chosen pool.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: extreme-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=extreme
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: premium-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=standard
  backendType: google-cloud-netapp-volumes
```

For more details on storage classes, refer to Create a storage class.

Example definitions for SMB volumes

Using `nasType`, `node-stage-secret-name`, and `node-stage-secret-namespace`, you can specify an SMB volume and provide the required Active Directory credentials. Any Active Directory user/password with any/no permissions can be used for the node stage secret.

Basic configuration on default namespace

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

Using different secrets per namespace

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

Using different secrets per volume

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` filters for pools which support SMB volumes. `nasType: nfs` or `nasType: null` filters for NFS pools.

PVC definition example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: gcnv-nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs-sc
```

To verify if the PVC is bound, run the following command:

```
kubectl get pvc gcnv-nfs-pvc

NAME          STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS AGE
gcnv-nfs-pvc  Bound    pvc-b00f2414-e229-40e6-9b16-ee03eb79a213  100Gi
RWX            gcnv-nfs-sc  1m
```

Configure auto-tiering for Google Cloud NetApp Volumes

Auto-tiering is configured through Trident backend parameters and PersistentVolumeClaim annotations during volume provisioning. You can configure auto-tiering for Google Cloud NetApp Volumes using Trident.

Overview

Auto-tiering allows Trident to provision volumes that automatically move inactive data from a performance tier to a capacity tier.
This reduces storage cost while preserving performance for frequently accessed data.

Trident applies auto-tiering settings only at volume creation time.
Post-provisioning changes are not supported in Trident 26.02.

Concepts

Auto-tiering

Auto-tiering moves infrequently accessed data from a performance tier to a capacity tier based on access patterns.

Data movement occurs asynchronously and is not immediate.

Tiering policy

The tiering policy determines whether auto-tiering is enabled for a volume.

The following policies are supported:
* `auto`: Enables automatic tiering based on access patterns
* `none`: Disables auto-tiering

Cooling days

Cooling days specify the minimum number of days a block of data must remain inactive before it becomes eligible for tiering.
Cooling days apply only when the tiering policy is set to `auto`.

Configuration model

Configuration scopes

Auto-tiering can be configured at multiple scopes:

- **Storage pool scope**
  Applies to all volumes provisioned from the pool.
- **Volume scope**
  Applies to a single volume through PersistentVolumeClaim annotations.

Trident determines the effective configuration based on where each setting is defined.

Configuration precedence

When the same setting is defined at multiple scopes, Trident applies the following precedence order:

1. PersistentVolumeClaim annotations
2. Trident backend configuration
3. Storage pool defaults

Settings defined at a higher precedence override lower-level values.

Supported functionality in Trident 26.02

Trident 26.02 supports the following auto-tiering capabilities for Google Cloud NetApp Volumes:

- Enabling or disabling auto-tiering during volume provisioning
- Defining a tiering policy in the Trident backend configuration
- Overriding the tiering policy and cooling days per volume using PVC annotations
- Configuring cooling days for volumes with auto-tiering enabled

Unsupported functionality in Trident 26.02

The following operations are not supported:

- Modifying auto-tiering settings after volume creation
- Changing tiering policies on existing volumes using Kubernetes updates
- Applying auto-tiering settings outside of Trident-managed provisioning workflows

Backend configuration parameters

The following parameters control auto-tiering behavior when defined in the Trident backend configuration:

Parameter	Required	Description
tieringPolicy	No	Tiering policy for volumes (`auto` or `none`)
tieringMinimumCoolingDays	No	Number of inactive days before data is tiered (range: 2–183, default: 31)

Volume-level overrides using PersistentVolumeClaim annotations

Supported annotations

PersistentVolumeClaim annotations allow per-volume overrides of auto-tiering settings.

Annotation	Description
`trident.netapp.io/tieringPolicy`	Overrides the tiering policy for the volume
`trident.netapp.io/tieringMinimumCoolingDays`	Overrides the cooling days value for the volume

Example: PersistentVolumeClaim with auto-tiering overrides

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-tiering-pvc
  annotations:
    trident.netapp.io/tieringPolicy: auto
    trident.netapp.io/tieringMinimumCoolingDays: "45"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: google-cloud-netapp-volumes-auto-tiering
  resources:
    requests:
      storage: 500Gi
```

Behavior and limitations

Provisioning behavior

- Auto-tiering settings are evaluated and applied only at volume creation time.
- Trident does not reconcile tiering configuration after provisioning.
- Cooling days are ignored when the tiering policy is set to `none`.

Platform limitations

- Auto-tiering is supported only for NAS volumes (NFS and SMB).
- Block volumes (iSCSI) do not support auto-tiering.
- The Google Cloud NetApp Volumes storage pool must have auto-tiering enabled in Google Cloud.

Supported values

- Valid range for `tieringMinimumCoolingDays`: 2 to 183
- Default value: 31

Configure a NetApp HCI or SolidFire backend

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

Element driver details

Trident provides the `solidfire-san` storage driver to communicate with the cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

The `solidfire-san` storage driver supports *file* and *block* volume modes. For the `Filesystem` volumeMode, Trident creates a volume and creates a filesystem. The filesystem type is specified by the StorageClass.

Driver	Protocol	VolumeMode	Access modes supported	File systems supported
`solidfire-san`	iSCSI	Block	RWO, ROX, RWX, RWOP	No Filesystem. Raw block device.
`solidfire-san`	iSCSI	Filesystem	RWO, RWOP	`xfs`, `ext3`, `ext4`

Before you begin

You'll need the following before creating an Element backend.

- A supported storage system that runs Element software.
- Credentials to a NetApp HCI/SolidFire cluster admin or tenant user that can manage volumes.
- All of your Kubernetes worker nodes should have the appropriate iSCSI tools installed. Refer to worker node preparation information.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	Always "solidfire-san"
`backendName`	Custom name or the storage backend	"solidfire_" + storage (iSCSI) IP address
`Endpoint`	MVIP for the SolidFire cluster with tenant credentials	
`SVIP`	Storage (iSCSI) IP address and port	
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes.	""
`TenantName`	Tenant name to use (created if not found)	
`InitiatorIFace`	Restrict iSCSI traffic to a specific host interface	"default"
`UseCHAP`	Use CHAP to authenticate iSCSI. Trident uses CHAP.	true
`AccessGroups`	List of Access Group IDs to use	Finds the ID of an access group named "trident"
`Types`	QoS specifications	
`limitVolumeSize`	Fail provisioning if requested volume size is above this value	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true}	null

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

Example 1: Backend configuration for `solidfire-san` driver with three volume types

This example shows a backend file using CHAP authentication and modeling three volume types with specific QoS guarantees. Most likely you would then define storage classes to consume each of these using the `IOPS` storage class parameter.

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
labels:
  k8scluster: dev1
  backend: dev1-element-cluster
UseCHAP: true
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
  - Type: Silver
    Qos:
      minIOPS: 4000
      maxIOPS: 6000
      burstIOPS: 8000
  - Type: Gold
    Qos:
      minIOPS: 6000
      maxIOPS: 8000
      burstIOPS: 10000
```

Example 2: Backend and storage class configuration for `solidfire-san` driver with virtual pools

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

Trident copies labels present on a storage pool to the backend storage LUN at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the `type` at Silver. The virtual pools are defined in the `storage` section. In this example, some of the storage pools set their own type, and some pools override the default values set above.

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
UseCHAP: true
```

```
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
  - Type: Silver
    Qos:
      minIOPS: 4000
      maxIOPS: 6000
      burstIOPS: 8000
  - Type: Gold
    Qos:
      minIOPS: 6000
      maxIOPS: 8000
      burstIOPS: 10000
type: Silver
labels:
  store: solidfire
  k8scluster: dev-1-cluster
region: us-east-1
storage:
  - labels:
      performance: gold
      cost: "4"
    zone: us-east-1a
    type: Gold
  - labels:
      performance: silver
      cost: "3"
    zone: us-east-1b
    type: Silver
  - labels:
      performance: bronze
      cost: "2"
    zone: us-east-1c
    type: Bronze
  - labels:
      performance: silver
      cost: "1"
    zone: us-east-1d
```

The following StorageClass definitions refer to the above virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pool(s) can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

The first StorageClass (`solidfire-gold-four`) will map to the first virtual pool. This is the only pool offering gold performance with a `Volume Type QoS` of Gold. The last StorageClass (`solidfire-silver`) calls out any storage pool which offers a silver performance. Trident will decide which virtual pool is selected and ensures the storage requirement is met.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold-four
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold; cost=4
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-three
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=3
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze; cost=2
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=1
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: solidfire-silver
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
  fsType: ext4
```

Find more information

- Volume access groups

ONTAP SAN drivers

ONTAP SAN driver overview

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

ONTAP SAN driver details

Trident provides the following SAN storage drivers to communicate with the ONTAP cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-san`	iSCSI SCSI over FC	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device
`ontap-san`	iSCSI SCSI over FC	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`
`ontap-san`	NVMe/TCP Refer to Additional considerations for NVMe/TCP.	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-san`	NVMe/TCP Refer to Additional considerations for NVMe/TCP.	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`
`ontap-san-economy`	iSCSI	Block	RWO, ROX, RWX, RWOP	No filesystem; raw block device
`ontap-san-economy`	iSCSI	Filesystem	RWO, RWOP ROX and RWX are not available in Filesystem volume mode.	`xfs`, `ext3`, `ext4`

- Use `ontap-san-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits.
- Use `ontap-nas-economy` only if persistent volume usage count is expected to be higher than supported ONTAP volume limits and the `ontap-san-economy` driver cannot be used.
- Do not use use `ontap-nas-economy` if you anticipate the need for data protection, disaster recovery, or mobility.
- NetApp does not recommend using Flexvol autogrow in all ONTAP drivers, except ontap-san. As a workaround, Trident supports the use of snapshot reserve and scales Flexvol volumes accordingly.

User permissions

Trident expects to be run as either an ONTAP or SVM administrator, typically using the `admin` cluster user or a `vsadmin` SVM user, or a user with a different name that has the same role. For Amazon FSx for NetApp ONTAP deployments, Trident expects to be run as either an ONTAP or SVM administrator, using the cluster `fsxadmin` user or a `vsadmin` SVM user, or a user with a different name that has the same role. The `fsxadmin` user is a limited replacement for the cluster admin user.

If you use the `limitAggregateUsage` parameter, cluster admin permissions are required. When using Amazon FSx for NetApp ONTAP with Trident, the `limitAggregateUsage` parameter will not work with the `vsadmin` and `fsxadmin` user accounts. The configuration operation will fail if you specify this parameter.

While it is possible to create a more restrictive role within ONTAP that a Trident driver can use, we don't recommend it. Most new releases of Trident will call additional APIs that would have to be accounted for, making upgrades difficult and error-prone.

Additional considerations for NVMe/TCP

Trident supports the non-volatile memory express (NVMe) protocol using the `ontap-san` driver including:

- IPv6
- Snapshots and clones of NVMe volumes
- Resizing an NVMe volume
- Importing an NVMe volume that was created outside of Trident so that its lifecycle can be managed by Trident
- NVMe-native multipathing
- Graceful or ungraceful shutdown of the K8s nodes (24.06)

Trident does not support:

- DH-HMAC-CHAP that is natively supported by NVMe
- Device mapper (DM) multipathing
- LUKS encryption

> NVMe is supported only with ONTAP REST APIs and not supported with ONTAPI (ZAPI).

Prepare to configure backend with ONTAP SAN drivers

Understand the requirements and authentication options for configuring an ONTAP backend with ONTAP SAN drivers.

Requirements

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

> ASA r2 systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. In ASA r2 systems, storage availability zones are used instead of aggregates. Refer to this Knowledge Base article on how to assign aggregates to SVMs in ASA r2 systems.

Remember that you can also run more than one driver, and create storage classes that point to one or the other. For example, you could configure a `san-dev` class that uses the `ontap-san` driver and a `san-default` class that uses the `ontap-san-economy` one.

All your Kubernetes worker nodes must have the appropriate iSCSI tools installed. Refer to Prepare the worker node for details.

Authenticate the ONTAP backend

Trident offers two modes of authenticating an ONTAP backend.

- Credential-based: The username and password to an ONTAP user with the required permissions. It is recommended to use a pre-defined security login role, such as `admin` or `vsadmin` to ensure maximum compatibility with ONTAP versions.
- Certificate-based: Trident can also communicate with an ONTAP cluster using a certificate installed on the backend. Here, the backend definition must contain Base64-encoded values of the client certificate, key,

and the trusted CA certificate if used (recommended).

You can update existing backends to move between credential-based and certificate-based methods. However, only one authentication method is supported at a time. To switch to a different authentication method, you must remove the existing method from the backend configuration.

> If you attempt to provide **both credentials and certificates**, backend creation will fail with an error that more than one authentication method was provided in the configuration file.

Enable credential-based authentication

Trident requires the credentials to an SVM-scoped/cluster-scoped admin to communicate with the ONTAP backend. It is recommended to make use of standard, pre-defined roles such as `admin` or `vsadmin`. This ensures forward compatibility with future ONTAP releases that might expose feature APIs to be used by future Trident releases. A custom security login role can be created and used with Trident, but is not recommended.

A sample backend definition will look like this:

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: password
```

JSON

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

Keep in mind that the backend definition is the only place the credentials are stored in plain text. After the backend is created, usernames/passwords are encoded with Base64 and stored as Kubernetes secrets. The creation or update of a backend is the only step that requires knowledge of the credentials. As such, it is an admin-only operation, to be performed by the Kubernetes/storage administrator.

Enable certificate-based authentication

New and existing backends can use a certificate and communicate with the ONTAP backend. Three parameters are required in the backend definition.

- clientCertificate: Base64-encoded value of client certificate.
- clientPrivateKey: Base64-encoded value of associated private key.
- trustedCACertificate: Base64-encoded value of trusted CA certificate. If using a trusted CA, this parameter must be provided. This can be ignored if no trusted CA is used.

A typical workflow involves the following steps.

Steps

1. Generate a client certificate and key. When generating, set Common Name (CN) to the ONTAP user to authenticate as.

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=admin"
```

2. Add trusted CA certificate to the ONTAP cluster. This might be already handled by the storage administrator. Ignore if no trusted CA is used.

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. Install the client certificate and key (from step 1) on the ONTAP cluster.

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

> After running this command, ONTAP prompts for certificate input. Paste the contents of the `k8senv.pem` file generated in step 1, then enter `END` to complete the installation.

4. Confirm the ONTAP security login role supports `cert` authentication method.

```
security login create -user-or-group-name admin -application ontapi
-authentication-method cert
security login create -user-or-group-name admin -application http
-authentication-method cert
```

5. Test authentication using certificate generated. Replace <ONTAP Management LIF> and <vserver name> with Management LIF IP and SVM name.

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. Encode certificate, key and trusted CA certificate with Base64.

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. Create backend using the values obtained from the previous step.

```
cat cert-backend.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"trustedCACertificate": "QNFinfO...SiqOyN",
"storagePrefix": "myPrefix_"
}

tridentctl create backend -f cert-backend.json -n trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       0 |
+------------+----------------+--------------------------------------
+--------+---------+
```

Update authentication methods or rotate credentials

You can update an existing backend to use a different authentication method or to rotate their credentials. This works both ways: backends that make use of username/password can be updated to use certificates; backends that utilize certificates can be updated to username/password based. To do this, you must remove the existing authentication method and add the new authentication method. Then use the updated backend.json file containing the required parameters to execute `tridentctl backend update`.

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

When rotating passwords, the storage administrator must first update the password for the user on ONTAP. This is followed by a backend update. When rotating certificates, multiple certificates can be added to the user. The backend is then updated to use the new certificate, following which the old certificate can be deleted from the ONTAP cluster.

Updating a backend does not disrupt access to volumes that have already been created, nor impact volume connections made after. A successful backend update indicates that Trident can communicate with the ONTAP backend and handle future volume operations.

Create custom ONTAP role for Trident

You can create an ONTAP cluster role with minimum privileges so that you do not have to use the ONTAP admin role to perform operations in Trident. When you include the username in a Trident backend configuration, Trident uses the ONTAP cluster role you created to perform the operations.

Refer to Trident custom-role generator for more information about creating Trident custom roles.

Using ONTAP CLI

1. Create a new role using the following command:

```
security login role create <role_name\> -cmddirname "command" -access all
–vserver <svm_name\>
```

2. Create a usename for the Trident user:

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> –vserver
<svm_name\> -comment "user_description"
```

3. Map the role to the user:

```
security login modify username <user_name\> –vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

Using System Manager

Perform the following steps in ONTAP System Manager:

1. **Create a custom role**:
   a. To create a custom role at the cluster-level, select **Cluster > Settings**.

      (Or) To create a custom role at the SVM level, select **Storage > Storage VMs > `required SVM`> Settings > Users and Roles**.

   b. Select the arrow icon (→) next to **Users and Roles**.
   c. Select **+Add** under **Roles**.
   d. Define the rules for the role and click **Save**.
2. **Map the role to the Trident user**:
   + Perform the following steps on the **Users and Roles** page:
   a. Select Add icon **+** under **Users**.
   b. Select the required username, and select a role in the drop-down menu for **Role**.
   c. Click **Save**.

Refer to the following pages for more information:

- Custom roles for administration of ONTAP or Define custom roles
- Work with roles and users

Authenticate connections with bidirectional CHAP

Trident can authenticate iSCSI sessions with bidirectional CHAP for the `ontap-san` and `ontap-san-economy` drivers. This requires enabling the `useCHAP` option in your backend definition. When set to `true`, Trident configures the SVM's default initiator security to bidirectional CHAP and set the username and secrets

from the backend file. NetApp recommends using bidirectional CHAP to authenticate connections. See the following sample configuration:

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
```

The `useCHAP` parameter is a Boolean option that can be configured only once. It is set to false by default. After you set it to true, you cannot set it to false.

In addition to `useCHAP=true`, the `chapInitiatorSecret`, `chapTargetInitiatorSecret`, `chapTargetUsername`, and `chapUsername` fields must be included in the backend definition. The secrets can be changed after a backend is created by running `tridentctl update`.

How it works

By setting `useCHAP` to true, the storage administrator instructs Trident to configure CHAP on the storage backend. This includes the following:

- Setting up CHAP on the SVM:
  - If the SVM's default initiator security type is none (set by default) **and** there are no pre-existing LUNs already present in the volume, Trident will set the default security type to `CHAP` and proceed to configuring the CHAP initiator and target username and secrets.
  - If the SVM contains LUNs, Trident will not enable CHAP on the SVM. This ensures that access to LUNs that are already present on the SVM isn't restricted.
- Configuring the CHAP initiator and target username and secrets; these options must be specified in the backend configuration (as shown above).

After the backend is created, Trident creates a corresponding `tridentbackend` CRD and stores the CHAP secrets and usernames as Kubernetes secrets. All PVs that are created by Trident on this backend will be mounted and attached over CHAP.

Rotate credentials and update backends

You can update the CHAP credentials by updating the CHAP parameters in the `backend.json` file. This will require updating the CHAP secrets and using the `tridentctl update` command to reflect these changes.

When updating the CHAP secrets for a backend, you must use `tridentctl` to update the backend. Do not update the credentials on the storage cluster using the ONTAP CLI or ONTAP System Manager as Trident will not be able to pick up these changes.

```
cat backend-san.json
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "ontap_san_chap",
    "managementLIF": "192.168.0.135",
    "svm": "ontap_iscsi_svm",
    "useCHAP": true,
    "username": "vsadmin",
    "password": "password",
    "chapInitiatorSecret": "cl9qxUpDaTeD",
    "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
}

./tridentctl update backend ontap_san_chap -f backend-san.json -n trident
+----------------+----------------+--------------------------------------
+--------+---------+
|   NAME         | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+----------------+----------------+--------------------------------------
+--------+---------+
| ontap_san_chap | ontap-san      | aa458f3b-ad2d-4378-8a33-1a472ffbeb5c |
online |       7 |
+----------------+----------------+--------------------------------------
+--------+---------+
```

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

ONTAP SAN configuration options and examples

Learn how to create and use ONTAP SAN drivers with your Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses. ASA r2 systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. These variations impact the usage of certain parameters as notated. Learn more about the differences between ASA r2 systems and other ONTAP systems. In the Trident backend configuration, you need not specify that your system is ASA r2. When you select `ontap-san` as the `storageDriverName`,

Trident detects automatically the ASA r2 or other ONTAP systems. Some backend configuration parameters are not applicable to ASA r2 systems as noted in the table below.

> Only the `ontap-san` driver (with iSCSI, NVMe/TCP, and FC protocols) is supported for ASA r2 systems.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	`ontap-san` or `ontap-san-economy`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF. A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. For seamless MetroCluster switchover, see the MetroCluster example. > If you are using "vsadmin" credentials, `managementLIF` must be that of the SVM; if using "admin" credentials, `managementLIF` must be that of the cluster.	"10.0.0.1", "[2001:1234:abcd::fefe]"

Parameter	Description	Default
`dataLIF`	IP address of protocol LIF. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. **Do not specify for iSCSI.** Trident uses ONTAP Selective LUN Map to discover the iSCSI LIFs needed to establish a multi path session. A warning is generated if `dataLIF` is explicitly defined. **Omit for Metrocluster.** See the MetroCluster example.	Derived by the SVM
`svm`	Storage virtual machine to use **Omit for Metrocluster.** See the MetroCluster example.	Derived if an SVM `managementLIF` is specified
`useCHAP`	Use CHAP to authenticate iSCSI for ONTAP SAN drivers [Boolean]. Set to `true` for Trident to configure and use bidirectional CHAP as the default authentication for the SVM given in the backend. Refer to Prepare to configure backend with ONTAP SAN drivers for details. **Not supported for FCP or NVMe/TCP.**	`false`
`chapInitiatorSecret`	CHAP initiator secret. Required if `useCHAP=true`	""
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`chapTargetInitiatorSecret`	CHAP target initiator secret. Required if `useCHAP=true`	""
`chapUsername`	Inbound username. Required if `useCHAP=true`	""
`chapTargetUsername`	Target username. Required if `useCHAP=true`	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based authentication.	""

Parameter	Description	Default
`username`	Username needed to communicate with the ONTAP cluster. Used for credential-based authentication. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	""
`password`	Password needed to communicate with the ONTAP cluster. Used for credential-based authentication. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	""
`svm`	Storage virtual machine to use	Derived if an SVM `managementLIF` is specified
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be modified later. To update this parameter, you will need to create a new backend.	`trident`
`aggregate`	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the `ontap-nas-flexgroup` driver, this option is ignored. If not assigned, any of the available aggregates can be used to provision a FlexGroup volume. When the aggregate is updated in SVM, it is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate in Trident to provision volumes, if the aggregate is renamed or moved out of the SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate to one that is present on the SVM or remove it altogether to bring the backend back online. **Do not specify for ASA r2 systems**.	""
`limitAggregateUsage`	Fail provisioning if usage is above this percentage. If you are using an Amazon FSx for NetApp ONTAP backend, do not specify `limitAggregateUsage`. The provided `fsxadmin` and `vsadmin` do not contain the permissions required to retrieve aggregate usage and limit it using Trident. **Do not specify for ASA r2 systems**.	"" (not enforced by default)

Parameter	Description	Default
`limitVolumeSize`	Fail provisioning if requested volume size is above this value. Also restricts the maximum size of the volumes it manages for LUNs.	"" (not enforced by default)
`lunsPerFlexvol`	Maximum LUNs per Flexvol, must be in range [50, 200]	`100`
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use unless you are troubleshooting and require a detailed log dump.	`null`
`useREST`	Boolean parameter to use ONTAP REST APIs. `useREST` When set to `true`, Trident uses ONTAP REST APIs to communicate with the backend; when set to `false`, Trident uses ONTAPI (ZAPI) calls to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the `ontapi` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles. Beginning with the Trident 24.06 release and ONTAP 9.15.1 or later, `useREST` is set to `true` by default; change `useREST` to `false` to use ONTAPI (ZAPI) calls. `useREST` is fully qualified for NVMe/TCP. NVMe is supported only with ONTAP REST APIs and not supported with ONTAPI (ZAPI). **If specified, always set to `true` for ASA r2 systems**.	`true` for ONTAP 9.15.1 or later, otherwise `false`.
`sanType`	Use to select `iscsi` for iSCSI, `nvme` for NVMe/TCP or `fcp` for SCSI over Fibre Channel (FC).	`iscsi` if blank

Parameter	Description	Default
`formatOptions`	Use `formatOptions` to specify command line arguments for the `mkfs` command, which will be applied whenever a volume is formatted. This allows you to format the volume according to your preferences. Make sure to specify the formatOptions similar to that of the mkfs command options, excluding the device path. Example: "-E nodiscard" **Supported for `ontap-san` and `ontap-san-economy` drivers with iSCSI protocol. Additionally, supported for ASA r2 systems when using iSCSI and NVMe/TCP protocols.**	
`limitVolumePoolSize`	Maximum requestable FlexVol size when using LUNs in ontap-san-economy backend.	"" (not enforced by default)
`denyNewVolumePools`	Restricts `ontap-san-economy` backends from creating new FlexVol volumes to contain their LUNs. Only preexisting Flexvols are used for provisioning new PVs.	

Recommendations for using formatOptions

Trident recommends the following options to expedite the formatting process:

- **-E nodiscard (ext3, ext4):** Do not attempt to discard blocks at mkfs time (discarding blocks initially is useful on solid state devices and sparse / thin-provisioned storage). This replaces the deprecated option "-K" and it is applicable to ext3, ext4 file systems.
- **-K (xfs):** Do not attempt to discard blocks at mkfs time. This option is applicable to xfs file system.

Authenticate Trident to a backend SVM using Active Directory credentials

You can configure Trident to authenticate to a backend SVM using Active Directory (AD) credentials. Before an AD account can access the SVM, you must configure AD domain controller access to the cluster or SVM. For cluster administration with an AD account, you must create domain tunnel. Refer to Configure Active Directory domain controller access in ONTAP for details.

steps

1. Configure Domain Name System (DNS) settings for a backend SVM:

   ```
   vserver services dns create -vserver <svm_name> -dns-servers
   <dns_server_ip1>,<dns_server_ip2>
   ```

2. Run the following command to create a computer account for the SVM in Active Directory:

   ```
   vserver active-directory create -vserver DataSVM -account-name ADSERVER1
   -domain demo.netapp.com
   ```

3. Use this command to create an AD user or group to manage the cluster or SVM

```
security login create -vserver <svm_name> -user-or-group-name
<ad_user_or_group> -application <application> -authentication-method domain
-role vsadmin
```

4. In the Trident backend configuration file, set the `username` and `password` parameters to the AD user or group name and password, respectively.

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the `defaults` section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
`spaceAllocation`	Space-allocation for LUNs	"true" **If specified, set to `true` for ASA r2 systems**.
`spaceReserve`	Space reservation mode; "none" (thin) or "volume" (thick). **Set to `none` for ASA r2** systems.	"none"
`snapshotPolicy`	Snapshot policy to use. **Set to `none` for ASA r2 systems**.	"none"
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend. Using QoS policy groups with Trident requires ONTAP 9.8 or later. You should use a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group enforces the ceiling for the total throughput of all workloads.	""
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""
`snapshotReserve`	Percentage of volume reserved for snapshots. **Do not specify for ASA r2 systems**.	"0" if `snapshotPolicy` is "none", otherwise ""
`splitOnClone`	Split a clone from its parent upon creation	"false"
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	"false" **If specified, set to `true` for ASA r2 systems**.

Parameter	Description	Default
`luksEncryption`	Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS).	"" **Set to `false` for ASA r2 systems.**
`tieringPolicy`	Tiering policy to use "none" **Do not specify for ASA r2 systems** .	
`nameTemplate`	Template to create custom volume names.	""

Volume provisioning examples

Here's an example with defaults defined:

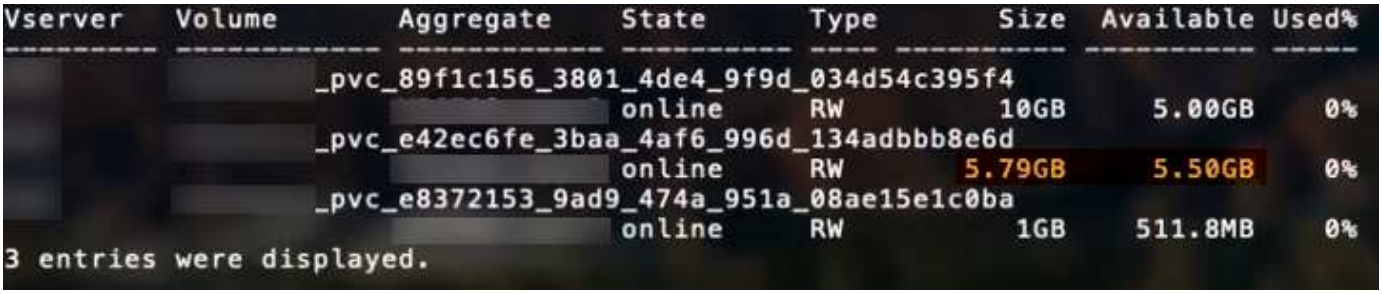
```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: <password>
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: standard
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

> For all volumes created using the `ontap-san` driver, Trident adds an extra 10 percent capacity to the FlexVol to accommodate the LUN metadata. The LUN will be provisioned with the exact size that the user requests in the PVC. Trident adds 10 percent to the FlexVol (shows as Available size in ONTAP). Users will now get the amount of usable capacity they requested. This change also prevents LUNs from becoming read-only unless the available space is fully utilized. This does not apply to ontap-san-economy.

For backends that define `snapshotReserve`, Trident calculates the size of volumes as follows:

```
Total volume size = [(PVC requested size) / (1 - (snapshotReserve
percentage) / 100)] * 1.1
```

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



Currently, resizing is the only way to use the new calculation for an existing volume.

Minimal configuration examples

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

> If you are using Amazon FSx on NetApp ONTAP with Trident, NetApp recommends that you specify DNS names for LIFs instead of IP addresses.

ONTAP SAN example

This is a basic configuration using the `ontap-san` driver.

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
username: vsadmin
password: <password>
```

MetroCluster example

You can configure the backend to avoid having to manually update the backend definition after switchover and switchback during SVM replication and recovery.

For seamless switchover and switchback, specify the SVM using `managementLIF` and omit the `svm` parameters. For example:

```
version: 1
storageDriverName: ontap-san
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

ONTAP SAN economy example

```
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
username: vsadmin
password: <password>
```

Certificate-based authentication example

In this basic configuration example `clientCertificate`, `clientPrivateKey`, and `trustedCACertificate` (optional, if using trusted CA) are populated in `backend.json` and take the base64-encoded values of the client certificate, private key, and trusted CA certificate, respectively.

```
---
version: 1
storageDriverName: ontap-san
backendName: DefaultSANBackend
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

Bidirectional CHAP examples

These examples create a backend with `useCHAP` set to `true`.

ONTAP SAN CHAP example

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

ONTAP SAN economy CHAP example

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

NVMe/TCP example

You must have an SVM configured with NVMe on your ONTAP backend. This is a basic backend configuration for NVMe/TCP.

```
---
version: 1
backendName: NVMeBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nvme
username: vsadmin
password: password
sanType: nvme
useREST: true
```

SCSI over FC (FCP) example

You must have an SVM configured with FC on your ONTAP backend. This is a basic backend configuration for FC.

```
---
version: 1
backendName: fcp-backend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_fc
username: vsadmin
password: password
sanType: fcp
useREST: true
```

Backend configuration example with nameTemplate

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap-san-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

formatOptions example for ontap-san-economy driver

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: ""
svm: svm1
username: ""
password: "!"
storagePrefix: whelk_
debugTraceFlags:
  method: true
  api: true
defaults:
  formatOptions: -E nodiscard
```

Examples of backends with virtual pools

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

Trident sets provisioning labels in the "Comments" field. Comments are set on the FlexVol volume Trident copies all labels present on a virtual pool to the storage volume at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In these examples, some of the storage pools set their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools override the default values.

ONTAP SAN example

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "40000"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
      adaptiveQosPolicy: adaptive-extreme
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
      qosPolicy: premium
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

ONTAP SAN economy example

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
labels:
  store: san_economy_store
region: us_east_1
storage:
  - labels:
      app: oracledb
      cost: "30"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
  - labels:
      app: postgresdb
      cost: "20"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
  - labels:
      app: mysqldb
      cost: "10"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1c
```

```
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

NVMe/TCP example

```
---
version: 1
storageDriverName: ontap-san
sanType: nvme
managementLIF: 10.0.0.1
svm: nvme_svm
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceAllocation: "false"
  encryption: "true"
storage:
  - labels:
      app: testApp
      cost: "20"
    defaults:
      spaceAllocation: "false"
      encryption: "false"
```

Map backends to StorageClasses

The following StorageClass definitions refer to the Examples of backends with virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

- The `protection-gold` StorageClass will map to the first virtual pool in the `ontap-san` backend. This is the only pool offering gold-level protection.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- The `protection-not-gold` StorageClass will map to the second and third virtual pool in `ontap-san` backend. These are the only pools offering a protection level other than gold.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- The `app-mysqldb` StorageClass will map to the third virtual pool in `ontap-san-economy` backend. This is the only pool offering storage pool configuration for the mysqldb type app.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- The `protection-silver-creditpoints-20k` StorageClass will map to the second virtual pool in `ontap-san` backend. This is the only pool offering silver-level protection and 20000 creditpoints.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- The `creditpoints-5k` StorageClass will map to the third virtual pool in `ontap-san` backend and the fourth virtual pool in the `ontap-san-economy` backend. These are the only pool offerings with 5000 creditpoints.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

- The `my-test-app-sc` StorageClass will map to the `testAPP` virtual pool in the `ontap-san` driver with `sanType: nvme`. This is the only pool offering `testApp`.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: my-test-app-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=testApp"
  fsType: "ext4"
```

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

ONTAP NAS drivers

ONTAP NAS driver overview

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

ONTAP NAS driver details

Trident provides the following NAS storage drivers to communicate with the ONTAP cluster. Supported access modes are: *ReadWriteOnce* (RWO), *ReadOnlyMany* (ROX), *ReadWriteMany* (RWX), *ReadWriteOncePod* (RWOP).

Driver	Protocol	volumeMode	Access modes supported	File systems supported
`ontap-nas`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", `nfs`, `smb`
`ontap-nas-economy`	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", `nfs`, `smb`

Driver	Protocol	volumeMode	Access modes supported	File systems supported
ontap-nas-flexgroup	NFS SMB	Filesystem	RWO, ROX, RWX, RWOP	"", nfs, smb

- Use ontap-san-economy only if persistent volume usage count is expected to be higher than supported ONTAP volume limits.
- Use ontap-nas-economy only if persistent volume usage count is expected to be higher than supported ONTAP volume limits and the ontap-san-economy driver cannot be used.
- Do not use use ontap-nas-economy if you anticipate the need for data protection, disaster recovery, or mobility.
- NetApp does not recommend using Flexvol autogrow in all ONTAP drivers, except ontap-san. As a workaround, Trident supports the use of snapshot reserve and scales Flexvol volumes accordingly.

User permissions

Trident expects to be run as either an ONTAP or SVM administrator, typically using the admin cluster user or a vsadmin SVM user, or a user with a different name that has the same role.

For Amazon FSx for NetApp ONTAP deployments, Trident expects to be run as either an ONTAP or SVM administrator, using the cluster fsxadmin user or a vsadmin SVM user, or a user with a different name that has the same role. The fsxadmin user is a limited replacement for the cluster admin user.

If you use the limitAggregateUsage parameter, cluster admin permissions are required. When using Amazon FSx for NetApp ONTAP with Trident, the limitAggregateUsage parameter will not work with the vsadmin and fsxadmin user accounts. The configuration operation will fail if you specify this parameter.

While it is possible to create a more restrictive role within ONTAP that a Trident driver can use, we don't recommend it. Most new releases of Trident will call additional APIs that would have to be accounted for, making upgrades difficult and error-prone.

Prepare to configure a backend with ONTAP NAS drivers

Understand the requirements, authentication options, and export policies for configuring an ONTAP backend with ONTAP NAS drivers. Beginning with the 25.10 release, NetApp Trident supports NetApp AFX storage system. NetApp AFX storage systems differ from other ONTAP systems (ASA, AFF, and FAS) in the implementation of their storage layer. In the Trident backend configuration, you need not specify that your system is AFX. When you select ontap-nas as the storageDriverName, Trident detects automatically the AFX systems.

Only the ontap-nas driver (with NFS protocol) is supported for AFX systems; SMB protocol is not supported.

Requirements

- For all ONTAP backends, Trident requires at least one aggregate be assigned to the SVM.
- You can run more than one driver, and create storage classes that point to one or the other. For example, you could configure a Gold class that uses the `ontap-nas` driver and a Bronze class that uses the `ontap-nas-economy` one.
- All your Kubernetes worker nodes must have the appropriate NFS tools installed. Refer to here for more details.
- Trident supports SMB volumes mounted to pods running on Windows nodes only. Refer to Prepare to provision SMB volumes for details.

Authenticate the ONTAP backend

Trident offers two modes of authenticating an ONTAP backend.

- Credential-based: This mode requires sufficient permissions to the ONTAP backend. It is recommended to use an account associated with a pre-defined security login role, such as `admin` or `vsadmin` to ensure maximum compatibility with ONTAP versions.
- Certificate-based: This mode requires a certificate installed on the backend for Trident to communicate with an ONTAP cluster. Here, the backend definition must contain Base64-encoded values of the client certificate, key, and the trusted CA certificate if used (recommended).

You can update existing backends to move between credential-based and certificate-based methods. However, only one authentication method is supported at a time. To switch to a different authentication method, you must remove the existing method from the backend configuration.

If you attempt to provide **both credentials and certificates**, backend creation will fail with an error that more than one authentication method was provided in the configuration file.

Enable credential-based authentication

Trident requires the credentials to an SVM-scoped/cluster-scoped admin to communicate with the ONTAP backend. It is recommended to make use of standard, pre-defined roles such as `admin` or `vsadmin`. This ensures forward compatibility with future ONTAP releases that might expose feature APIs to be used by future Trident releases. A custom security login role can be created and used with Trident, but is not recommended.

A sample backend definition will look like this:

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
credentials:
  name: secret-backend-creds
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "credentials": {
        "name": "secret-backend-creds"
    }
}
```

Keep in mind that the backend definition is the only place the credentials are stored in plain text. After the backend is created, usernames/passwords are encoded with Base64 and stored as Kubernetes secrets. The creation/updation of a backend is the only step that requires knowledge of the credentials. As such, it is an admin-only operation, to be performed by the Kubernetes/storage administrator.

Enable certificate-based Authentication

New and existing backends can use a certificate and communicate with the ONTAP backend. Three parameters are required in the backend definition.

- clientCertificate: Base64-encoded value of client certificate.
- clientPrivateKey: Base64-encoded value of associated private key.
- trustedCACertificate: Base64-encoded value of trusted CA certificate. If using a trusted CA, this parameter must be provided. This can be ignored if no trusted CA is used.

A typical workflow involves the following steps.

Steps

1. Generate a client certificate and key. When generating, set Common Name (CN) to the ONTAP user to authenticate as.

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=vsadmin"
```

2. Add trusted CA certificate to the ONTAP cluster. This might be already handled by the storage administrator. Ignore if no trusted CA is used.

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. Install the client certificate and key (from step 1) on the ONTAP cluster.

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

4. Confirm the ONTAP security login role supports `cert` authentication method.

```
security login create -user-or-group-name vsadmin -application ontapi
-authentication-method cert -vserver <vserver-name>
security login create -user-or-group-name vsadmin -application http
-authentication-method cert -vserver <vserver-name>
```

5. Test authentication using certificate generated. Replace <ONTAP Management LIF> and <vserver name> with Management LIF IP and SVM name. You must ensure the LIF has its service policy set to `default-data-management`.

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. Encode certificate, key and trusted CA certificate with Base64.

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. Create backend using the values obtained from the previous step.

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

Update authentication methods or rotate credentials

You can update an existing backend to use a different authentication method or to rotate their credentials. This works both ways: backends that make use of username/password can be updated to use certificates; backends that utilize certificates can be updated to username/password based. To do this, you must remove the existing authentication method and add the new authentication method. Then use the updated backend.json file containing the required parameters to execute `tridentctl update backend`.

```
cat cert-backend-updated.json
```

```
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}
```

```
#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

When rotating passwords, the storage administrator must first update the password for the user on ONTAP. This is followed by a backend update. When rotating certificates, multiple certificates can be added to the user. The backend is then updated to use the new certificate, following which the old certificate can be deleted from the ONTAP cluster.

Updating a backend does not disrupt access to volumes that have already been created, nor impact volume connections made after. A successful backend update indicates that Trident can communicate with the ONTAP backend and handle future volume operations.

Create custom ONTAP role for Trident

You can create an ONTAP cluster role with minimum privileges so that you do not have to use the ONTAP admin role to perform operations in Trident. When you include the username in a Trident backend configuration, Trident uses the ONTAP cluster role you created to perform the operations.

Refer to Trident custom-role generator for more information about creating Trident custom roles.

Using ONTAP CLI

1. Create a new role using the following command:

```
security login role create <role_name\> -cmddirname "command" -access all
–vserver <svm_name\>
```

2. Create a usename for the Trident user:

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> –vserver
<svm_name\> -comment "user_description"
```

3. Map the role to the user:

```
security login modify username <user_name\> –vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

Using System Manager

Perform the following steps in ONTAP System Manager:

1. **Create a custom role**:
   a. To create a custom role at the cluster-level, select **Cluster > Settings**.

      (Or) To create a custom role at the SVM level, select **Storage > Storage VMs > `required SVM`> Settings > Users and Roles**.

   b. Select the arrow icon (→) next to **Users and Roles**.
   c. Select **+Add** under **Roles**.
   d. Define the rules for the role and click **Save**.
2. **Map the role to the Trident user**:
   + Perform the following steps on the **Users and Roles** page:
   a. Select Add icon **+** under **Users**.
   b. Select the required username, and select a role in the drop-down menu for **Role**.
   c. Click **Save**.

Refer to the following pages for more information:

- Custom roles for administration of ONTAP or Define custom roles
- Work with roles and users

Manage NFS export policies

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

Trident provides two options when working with export policies:

- Trident can dynamically manage the export policy itself; in this mode of operation, the storage administrator

specifies a list of CIDR blocks that represent admissible IP addresses. Trident adds applicable node IPs that fall in these ranges to the export policy automatically at publish time. Alternatively, when no CIDRs are specified, all global-scoped unicast IPs found on the node that the volume being published to will be added to the export policy.

- Storage administrators can create an export policy and add rules manually. Trident uses the default export policy unless a different export policy name is specified in the configuration.

Dynamically manage export policies

Trident provides the ability to dynamically manage export policies for ONTAP backends. This provides the storage administrator the ability to specify a permissible address space for worker node IPs, rather than defining explicit rules manually. It greatly simplifies export policy management; modifications to the export policy no longer require manual intervention on the storage cluster. Moreover, this helps restrict access to the storage cluster only to worker nodes that are mounting volumes and have IPs in the range specified, supporting a fine-grained and automated management.

> Do not use Network Address Translation (NAT) when using dynamic export policies. With NAT, the storage controller sees the frontend NAT address and not the actual IP host address, so access will be denied when no match is found in the export rules.

Example

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

```
---
version: 1
storageDriverName: ontap-nas-economy
backendName: ontap_nas_auto_export
managementLIF: 192.168.0.135
svm: svm1
username: vsadmin
password: password
autoExportCIDRs:
  - 192.168.0.0/24
autoExportPolicy: true
```

> When using this feature, you must ensure that the root junction in your SVM has a previously created export policy with an export rule that permits the node CIDR block (such as the default export policy). Always follow NetApp recommended best practice to dedicate an SVM for Trident.

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

- `autoExportPolicy` is set to `true`. This indicates that Trident creates an export policy for each volume provisioned with this backend for the `svm1` SVM and handle the addition and deletion of rules using `autoexportCIDRs` address blocks. Until a volume is attached to a node, the volume uses an empty export policy with no rules to prevent unwanted access to that volume. When a volume is published to a node Trident creates an export policy with the same name as the underlying qtree containing the node IP within the specified CIDR block. These IPs will also be added to the export policy used by the parent

FlexVol volume

- For example:
  - backend UUID 403b5326-8482-40db-96d0-d83fb3f4daec
  - `autoExportPolicy` set to `true`
  - storage prefix `trident`
  - PVC UUID a79bcf5f-7b6d-4a40-9876-e2551f159c1c
  - qtree named trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c creates an export policy for the FlexVol named `trident-403b5326-8482-40db-96d0-d83fb3f4daec` , an export policy for the qtree named `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`, and an empty export policy named `trident_empty` on the SVM. The rules for the FlexVol export policy will be a superset of any rules contained in the qtree export policies. The empty export policy will be reused by any volumes that are not attached.

- `autoExportCIDRs` contains a list of address blocks. This field is optional and it defaults to ["0.0.0.0/0", "::/0"]. If not defined, Trident adds all globally-scoped unicast addresses found on the worker nodes with publications.

In this example, the `192.168.0.0/24` address space is provided. This indicates that Kubernetes node IPs that fall within this address range with publications will be added to the export policy that Trident creates. When Trident registers a node it runs on, it retrieves the IP addresses of the node and checks them against the address blocks provided in `autoExportCIDRs`. At publish time, after filtering the IPs, Trident creates the export policy rules for the client IPs for the node it is publishing to.

You can update `autoExportPolicy` and `autoExportCIDRs` for backends after you create them. You can append new CIDRs for a backend that is automatically managed or delete existing CIDRs. Exercise care when deleting CIDRs to ensure that existing connections are not dropped. You can also choose to disable `autoExportPolicy` for a backend and fall back to a manually created export policy. This will require setting the `exportPolicy` parameter in your backend config.

After Trident creates or updates a backend, you can check the backend using `tridentctl` or the corresponding `tridentbackend` CRD:

```
./tridentctl get backends ontap_nas_auto_export -n trident -o yaml
items:
- backendUUID: 403b5326-8482-40db-96d0-d83fb3f4daec
  config:
    aggregate: ""
    autoExportCIDRs:
    - 192.168.0.0/24
    autoExportPolicy: true
    backendName: ontap_nas_auto_export
    chapInitiatorSecret: ""
    chapTargetInitiatorSecret: ""
    chapTargetUsername: ""
    chapUsername: ""
    dataLIF: 192.168.0.135
    debug: false
    debugTraceFlags: null
    defaults:
      encryption: "false"
      exportPolicy: <automatic>
      fileSystemType: ext4
```

When a node is removed, Trident checks all export policies to remove the access rules corresponding to the node. By removing this node IP from the export policies of managed backends, Trident prevents rogue mounts, unless this IP is reused by a new node in the cluster.

For previously existing backends, updating the backend with `tridentctl update backend` ensures that Trident manages the export policies automatically. This creates two new export policies named after the backend's UUID and qtree name when they are needed. Volumes that are present on the backend will use the newly created export policies after they are unmounted and mounted again.

> Deleting a backend with auto-managed export policies will delete the dynamically created export policy. If the backend is re-created, it is treated as a new backend and will result in the creation of a new export policy.

If the IP address of a live node is updated, you must restart the Trident pod on the node. Trident will then update the export policy for backends it manages to reflect this IP change.

Prepare to provision SMB volumes

With a little additional preparation, you can provision SMB volumes using `ontap-nas` drivers.

> You must configure both NFS and SMB/CIFS protocols on the SVM to create an `ontap-nas-economy` SMB volume for ONTAP on-premises clusters. Failure to configure either of these protocols will cause SMB volume creation to fail.

> `autoExportPolicy` is not supported for SMB volumes.

Before you begin

Before you can provision SMB volumes, you must have the following.

- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2022. Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Trident secret containing your Active Directory credentials. To generate secret `smbcreds`:

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- A CSI proxy configured as a Windows service. To configure a `csi-proxy`, refer to GitHub: CSI Proxy or GitHub: CSI Proxy for Windows for Kubernetes nodes running on Windows.

Steps

1. For on-premises ONTAP, you can optionally create an SMB share or Trident can create one for you.

   SMB shares are required for Amazon FSx for ONTAP.

   You can create the SMB admin shares in one of two ways either using the Microsoft Management Console Shared Folders snap-in or using the ONTAP CLI. To create the SMB shares using the ONTAP CLI:

   a. If necessary, create the directory path structure for the share.

      The `vserver cifs share create` command checks the path specified in the -path option during share creation. If the specified path does not exist, the command fails.

   b. Create an SMB share associated with the specified SVM:

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

   c. Verify that the share was created:

```
vserver cifs share show -share-name share_name
```

      Refer to Create an SMB share for full details.

2. When creating the backend, you must configure the following to specify SMB volumes. For all FSx for ONTAP backend configuration options, refer to FSx for ONTAP configuration options and examples.

Parameter	Description	Example
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`
`nasType`	**Must set to `smb`.** If null, defaults to `nfs`.	`smb`
`securityStyle`	Security style for new volumes. **Must be set to `ntfs` or `mixed` for SMB volumes.**	`ntfs` or `mixed` for SMB volumes
`unixPermissions`	Mode for new volumes. **Must be left empty for SMB volumes.**	""

Enable secure SMB

Beginning with the 25.06 release, NetApp Trident supports secure provisioning of SMB volumes created using `ontap-nas` and `ontap-nas-economy` backends. When secure SMB is enabled, you can provide controlled access to SMB the shares for Active Directory (AD) users and user groups using Access Control Lists (ACLs).

Points to remember

- Importing `ontap-nas-economy` volumes is not supported.
- Only read-only clones are supported for `ontap-nas-economy` volumes.
- If Secure SMB is enabled, Trident will ignore the SMB share mentioned in the backend.
- Updating the PVC annotation, storage class annotation, and backend field does not update the SMB share ACL.
- The SMB share ACL specified in the annotation of the clone PVC will take precedence over those in the source PVC.
- Ensure that you provide valid AD users while enabling secure SMB. Invalid users will not be added to the ACL.
- If you provide the same AD user in the backend, storage class, and PVC with different permissions, the permission priority will be: PVC, storage class, and then backend.
- Secure SMB is supported for `ontap-nas` managed volume imports and not applicable to unmanaged volume imports.

Steps

1. Specify adAdminUser in TridentBackendConfig as shown in the following example:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.193.176.x
  svm: svm0
  useREST: true
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

2. Add the annotation in the storage class.

   Add the `trident.netapp.io/smbShareAdUser` annotation to the storage class to enable secure SMB without fail.
   The user value specified for the annotation `trident.netapp.io/smbShareAdUser` should be the same as the username specified in the `smbcreds` secret.
   You can choose one of the following for `smbShareAdUserPermission`: `full_control`, `change`, or `read`. The default permission is `full_control`.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

3. Create a PVC.

   The following example creates a PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/snapshotDirectory: "true"
    trident.netapp.io/smbShareAccessControl: |
      read:
        - tridentADtest
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

ONTAP NAS configuration options and examples

Learn to create and use ONTAP NAS drivers with your Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses. Beginning with the 25.10 release, NetApp Trident supports NetApp AFX storage systems. NetApp AFX storage systems differ from other ONTAP-based systems (ASA, AFF, and FAS) in the implementation of their storage layer.

> Only the `ontap-nas` driver (with NFS protocol) is supported for NetApp AFX systems; SMB protocol is not supported.

Backend configuration options

In the Trident backend configuration, you need not specify that your system is an NetApp AFX storage system. When you select `ontap-nas` as the `storageDriverName`, Trident detects automatically the AFX storage system. Some backend configuration parameters are not applicable to AFX storage systems.

The following table displays backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver For NetApp AFX systems, only `ontap-nas` is supported.	`ontap-nas`, `ontap-nas-economy`, or `ontap-nas-flexgroup`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF

Parameter	Description	Default
`managementLIF`	IP address of a cluster or SVM management LIF A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. For seamless MetroCluster switchover, see the MetroCluster example.	"10.0.0.1", "[2001:1234:abcd::fefe]"
`dataLIF`	IP address of protocol LIF. NetApp recommends specifying `dataLIF`. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. Can be changed after initial setting. Refer to Update `dataLIF` after initial configuration. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`. **Omit for Metrocluster.** See the MetroCluster example.	Specified address or derived from SVM, if not specified (not recommended)
`svm`	Storage virtual machine to use **Omit for Metrocluster.** See the MetroCluster example.	Derived if an SVM `managementLIF` is specified
`autoExportPolicy`	Enable automatic export policy creation and updating [Boolean]. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	false
`autoExportCIDRs`	List of CIDRs to filter Kubernetes' node IPs against when `autoExportPolicy` is enabled. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	["0.0.0.0/0", "::/0"]`

Parameter	Description	Default
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based auth	""
`username`	Username to connect to the cluster/SVM. Used for credential-based auth. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	
`password`	Password to connect to the cluster/SVM. Used for credential-based auth. For Active Directory authentication, see Authenticate Trident to a backend SVM using Active Directory credentials.	
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be updated after you set it When using ontap-nas-economy and a storagePrefix that is 24 or more characters, the qtrees will not have the storage prefix embedded, though it will be in the volume name.	"trident"

Parameter	Description	Default
`aggregate`	Aggregate for provisioning (optional; if set, must be assigned to the SVM). For the `ontap-nas-flexgroup` driver, this option is ignored. If not assigned, any of the available aggregates can be used to provision a FlexGroup volume. When the aggregate is updated in SVM, it is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate in Trident to provision volumes, if the aggregate is renamed or moved out of the SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate to one that is present on the SVM or remove it altogether to bring the backend back online. **Do not specify for AFX storage systems.**	""
`limitAggregateUsage`	Fail provisioning if usage is above this percentage. **Does not apply to Amazon FSx for ONTAP**. **Do not specify for AFX storage systems.**	"" (not enforced by default)
flexgroupAggregateList	List of aggregates for provisioning (optional; if set, must be assigned to the SVM). All aggregates assigned to the SVM are used to provision a FlexGroup volume. Supported for the **ontap-nas-flexgroup** storage driver. When the aggregate list is updated in SVM, the list is updated in Trident automatically by polling SVM without having to restart the Trident Controller. When you have configured a specific aggregate list in Trident to provision volumes, if the aggregate list is renamed or moved out of SVM, the backend will move to failed state in Trident while polling the SVM aggregate. You must either change the aggregate list to one that is present on the SVM or remove it altogether to bring the backend back online.	""

Parameter	Description	Default
`limitVolumeSize`	Fail provisioning if requested volume size is above this value.	"" (not enforced by default)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.	null
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb` or null. Setting to null defaults to NFS volumes. **If specified, always set to `nfs` for AFX storage systems**.	`nfs`
`nfsMountOptions`	Comma-separated list of NFS mount options. The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in the storage class or the configuration file, Trident will not set any mount options on an associated persistent volume.	""
`qtreesPerFlexvol`	Maximum Qtrees per FlexVol, must be in range [50, 300]	"200"
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI; a name to allow Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes. This parameter is optional for on-premises ONTAP. This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.	`smb-share`

Parameter	Description	Default
`useREST`	Boolean parameter to use ONTAP REST APIs. `useREST` When set to `true`, Trident uses ONTAP REST APIs to communicate with the backend; when set to `false`, Trident uses ONTAPI (ZAPI) calls to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the `ontapi` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles. Beginning with the Trident 24.06 release and ONTAP 9.15.1 or later, `useREST` is set to `true` by default; change `useREST` to `false` to use ONTAPI (ZAPI) calls. **If specified, always set to `true` for AFX storage systems**.	`true` for ONTAP 9.15.1 or later, otherwise `false`.
`limitVolumePoolSize`	Maximum requestable FlexVol size when using Qtrees in ontap-nas-economy backend.	"" (not enforced by default)
`denyNewVolumePools`	Restricts `ontap-nas-economy` backends from creating new FlexVol volumes to contain their Qtrees. Only preexisting Flexvols are used for provisioning new PVs.	
`adAdminUser`	Active Directory admin user or user group with full access to SMB shares. Use this parameter to provide admin rights to the SMB share with full control.	

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the `defaults` section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
`spaceAllocation`	Space-allocation for Qtrees	"true"
`spaceReserve`	Space reservation mode; "none" (thin) or "volume" (thick)	"none"
`snapshotPolicy`	Snapshot policy to use	"none"
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""

Parameter	Description	Default
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend. Not supported by ontap-nas-economy.	""
`snapshotReserve`	Percentage of volume reserved for snapshots	"0" if `snapshotPolicy` is "none", otherwise ""
`splitOnClone`	Split a clone from its parent upon creation	"false"
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	"false"
`tieringPolicy`	Tiering policy to use "none"	
`unixPermissions`	Mode for new volumes	"777" for NFS volumes; empty (not applicable) for SMB volumes
`snapshotDir`	Controls access to the `.snapshot` directory	`true`, `false` (Set explicitly).
`exportPolicy`	Export policy to use	"default"
`securityStyle`	Security style for new volumes. NFS supports `mixed` and `unix` security styles. SMB supports `mixed` and `ntfs` security styles.	NFS default is `unix`. SMB default is `ntfs`.
`nameTemplate`	Template to create custom volume names.	""

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

Volume provisioning examples

Here's an example with defaults defined:

```
---
version: 1
storageDriverName: ontap-nas
backendName: customBackendName
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
labels:
  k8scluster: dev1
  backend: dev1-nasbackend
svm: trident_svm
username: cluster-admin
password: <password>
limitAggregateUsage: 80%
limitVolumeSize: 50Gi
nfsMountOptions: nfsvers=4
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: premium
  exportPolicy: myk8scluster
  snapshotPolicy: default
  snapshotReserve: "10"
```

For `ontap-nas` and `ontap-nas-flexgroups`, Trident now uses a new calculation to ensure that the FlexVol is sized correctly with the snapshotReserve percentage and PVC. When the user requests a PVC, Trident creates the original FlexVol with more space by using the new calculation. This calculation ensures that the user receives the writable space they requested for in the PVC, and not less space than what they requested. Before v21.07, when the user requests a PVC (for example, 5 GiB), with the snapshotReserve to 50 percent, they get only 2.5 GiB of writeable space. This is because what the user requested for is the whole volume and `snapshotReserve` is a percentage of that. With Trident 21.07, what the user requests for is the writeable space and Trident defines the `snapshotReserve` number as the percentage of the whole volume. This does not apply to `ontap-nas-economy`. See the following example to see how this works:

The calculation is as follows:

```
Total volume size = <PVC requested size> / (1 - (<snapshotReserve
percentage> / 100))
```

For snapshotReserve = 50%, and PVC request = 5 GiB, the total volume size is 5/.5 = 10 GiB and the available size is 5 GiB, which is what the user requested in the PVC request. The `volume show` command should show results similar to this example:

Existing backends from previous installs will provision volumes as explained above when upgrading Trident. For volumes that you created before upgrading, you should resize their volumes for the change to be observed. For example, a 2 GiB PVC with `snapshotReserve=50` earlier resulted in a volume that provides 1 GiB of writable space. Resizing the volume to 3 GiB, for example, provides the application with 3 GiB of writable space on a 6 GiB volume.

Minimal configuration examples

The following examples show basic configurations that leave most parameters to default. This is the easiest way to define a backend.

> If you are using Amazon FSx on NetApp ONTAP with Trident, the recommendation is to specify DNS names for LIFs instead of IP addresses.

ONTAP NAS economy example

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

ONTAP NAS Flexgroup example

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

MetroCluster example

You can configure the backend to avoid having to manually update the backend definition after switchover and switchback during SVM replication and recovery.

For seamless switchover and switchback, specify the SVM using `managementLIF` and omit the `dataLIF` and `svm` parameters. For example:

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

SMB volumes example

```
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
nasType: smb
securityStyle: ntfs
unixPermissions: ""
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

Certificate-based authentication example

This is a minimal backend configuration example. `clientCertificate`, `clientPrivateKey`, and `trustedCACertificate` (optional, if using trusted CA) are populated in `backend.json` and take the base64-encoded values of the client certificate, private key, and trusted CA certificate, respectively.

```
---
version: 1
backendName: DefaultNASBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

Auto export policy example

This example shows you how you can instruct Trident to use dynamic export policies to create and manage the export policy automatically. This works the same for the `ontap-nas-economy` and `ontap-nas-flexgroup` drivers.

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-nasbackend
autoExportPolicy: true
autoExportCIDRs:
- 10.0.0.0/24
username: admin
password: password
nfsMountOptions: nfsvers=4
```

IPv6 addresses example

This example shows `managementLIF` using an IPv6 address.

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

Amazon FSx for ONTAP using SMB volumes example

The `smbShare` parameter is required for FSx for ONTAP using SMB volumes.

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
dataLIF: 10.0.0.15
svm: nfs_svm
smbShare: smb-share
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

Backend configuration example with nameTemplate

```
---
version: 1
storageDriverName: ontap-nas
backendName: ontap-nas-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

Examples of backends with virtual pools

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

Trident sets provisioning labels in the "Comments" field. Comments are set on FlexVol for `ontap-nas` or FlexGroup for `ontap-nas-flexgroup`. Trident copies all labels present on a virtual pool to the storage volume at provisioning. For convenience, storage administrators can define labels per virtual pool and group volumes by label.

In these examples, some of the storage pools set their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools override the default values.

ONTAP NAS example

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
svm: svm_nfs
username: admin
password: <password>
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: "false"
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      app: msoffice
      cost: "100"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
      adaptiveQosPolicy: adaptive-premium
  - labels:
      app: slack
      cost: "75"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      app: wordpress
```

```
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS FlexGroup example

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "50000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: gold
      creditpoints: "30000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
    defaults:
```

```
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS economy example

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: nas_economy_store
region: us_east_1
storage:
  - labels:
      department: finance
      creditpoints: "6000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: engineering
      creditpoints: "3000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      department: humanresource
      creditpoints: "2000"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
```

```
      encryption: "false"
      unixPermissions: "0775"
```

Map backends to StorageClasses

The following StorageClass definitions refer to Examples of backends with virtual pools. Using the `parameters.selector` field, each StorageClass calls out which virtual pools can be used to host a volume. The volume will have the aspects defined in the chosen virtual pool.

- The `protection-gold` StorageClass will map to the first and second virtual pool in the `ontap-nas-flexgroup` backend. These are the only pools offering gold level protection.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- The `protection-not-gold` StorageClass will map to the third and fourth virtual pool in the `ontap-nas-flexgroup` backend. These are the only pools offering protection level other than gold.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- The `app-mysqldb` StorageClass will map to the fourth virtual pool in the `ontap-nas` backend. This is the only pool offering storage pool configuration for mysqldb type app.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- The `protection-silver-creditpoints-20k` StorageClass will map to the third virtual pool in the `ontap-nas-flexgroup` backend. This is the only pool offering silver-level protection and 20000 creditpoints.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- The `creditpoints-5k` StorageClass will map to the third virtual pool in the `ontap-nas` backend and the second virtual pool in the `ontap-nas-economy` backend. These are the only pool offerings with 5000 creditpoints.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

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

Update `dataLIF` after initial configuration

You can change the dataLIF after initial configuration by running the following command to provide the new backend JSON file with updated dataLIF.

```
tridentctl update backend <backend-name> -f <path-to-backend-json-file-
with-updated-dataLIF>
```

If PVCs are attached to one or multiple pods, you must bring down all corresponding pods and then bring them back up in order to for the new dataLIF to take effect.

Secure SMB examples

Backend configuration with ontap-nas driver

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Backend configuration with ontap-nas-economy driver

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas-economy
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Backend configuration with storage pool

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm0
  useREST: false
  storage:
  - labels:
      app: msoffice
    defaults:
      adAdminUser: tridentADuser
  nasType: smb
  credentials:
    name: backend-tbc-ontap-invest-secret
```

Storage class example with ontap-nas driver

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADtest
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Ensure that you add `annotations` to enable secure SMB. Secure SMB does not work without the annotations, irrespective of configurations set in the Backend or PVC.

Storage class example with ontap-nas-economy driver

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser3
parameters:
  backendType: ontap-nas-economy
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

PVC example with a single AD user

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      change:
        - tridentADtest
      read:
        - tridentADuser
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

PVC example with multiple AD users

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-test-pvc
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      full_control:
        - tridentTestuser
        - tridentuser
        - tridentTestuser1
        - tridentuser1
      change:
        - tridentADuser
        - tridentADuser1
        - tridentADuser4
        - tridentTestuser2
      read:
        - tridentTestuser2
        - tridentTestuser3
        - tridentADuser2
        - tridentADuser3
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Amazon FSx for NetApp ONTAP

Use Trident with Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP is a fully managed AWS service that runs file systems powered by the NetApp ONTAP storage operating system. It provides ONTAP features, performance, and administration with the scalability and operational simplicity of AWS. A file system is the primary resource in Amazon FSx and is analogous to an on-premises ONTAP cluster. Each file system contains one or more storage virtual machines (SVMs), and each SVM contains one or more volumes that store files and directories. This integration enables Kubernetes clusters running in Amazon Elastic Kubernetes Service (EKS) to provision ONTAP-backed persistent volumes for block and file workloads.

Requirements

In addition to Trident requirements, to integrate FSx for ONTAP with Trident, you need:

- An existing Amazon EKS cluster or self-managed Kubernetes cluster with `kubectl` installed.

- An existing Amazon FSx for NetApp ONTAP file system and storage virtual machine (SVM) that is reachable from your cluster's worker nodes.
- Worker nodes that are prepared for NFS or iSCSI.

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

Considerations

- SMB volumes:
  - SMB volumes are supported using the `ontap-nas` driver only.
  - SMB volumes are not supported with Trident EKS add-on.
  - Trident supports SMB volumes mounted to pods running on Windows nodes only. Refer to Prepare to provision SMB volumes for details.
- Prior to Trident 24.02, volumes created on Amazon FSx file systems that have automatic backups enabled, could not be deleted by Trident. To prevent this issue in Trident 24.02 or later, specify the `fsxFilesystemID`, AWS `apiRegion`, AWS `apikey`, and AWS `secretKey` in the backend configuration file for AWS FSx for ONTAP.

> If you are specifying an IAM role to Trident, then you can omit specifying the `apiRegion`, `apiKey`, and `secretKey` fields to Trident explicitly. For more information, refer to FSx for ONTAP configuration options and examples.

Simultaneous usage of Trident SAN/iSCSI and EBS-CSI driver

If you plan to use ontap-san drivers (e.g., iSCSI) with AWS (EKS, ROSA, EC2, or any other instance), the multipath configuration required on the nodes might conflict with the Amazon Elastic Block Store (EBS) CSI driver. To ensure that multipathing functions without interfering with EBS disks on the same node, you need to exclude EBS in your multipathing setup. This example shows a `multipath.conf` file that includes the required Trident settings while excluding EBS disks from multipathing:

```
defaults {
    find_multipaths no
}
blacklist {
    device {
        vendor "NVME"
        product "Amazon Elastic Block Store"
    }
}
```

Authentication

Trident offers two modes of authentication.

- Credential-based(Recommended): Stores credentials securely in AWS Secrets Manager. You can use the `fsxadmin` user for your file system or the `vsadmin` user configured for your SVM.

Trident expects to be run as a `vsadmin` SVM user or as a user with a different name that has the same role. Amazon FSx for NetApp ONTAP has an `fsxadmin` user that is a limited replacement of the ONTAP `admin` cluster user. We strongly recommend using `vsadmin` with Trident.

- Certificate-based: Trident will communicate with the SVM on your FSx file system using a certificate installed on your SVM.

For details on enabling authentication, refer to the authentication for your driver type:

- ONTAP NAS authentication
- ONTAP SAN authentication

Tested Amazon Machine Images (AMIs)

EKS cluster supports various operating systems, but AWS has optimized certain Amazon Machine Images (AMIs) for containers and EKS. The following AMIs have been tested with NetApp Trident 25.02.

AMI	NAS	NAS-economy	iSCSI	iSCSI-economy
AL2023_x86_64_STANDARD	Yes	Yes	Yes	Yes
AL2_x86_64	Yes	Yes	Yes*	Yes*
BOTTLEROCKET_x86_64	Yes**	Yes	N/A	N/A
AL2023_ARM_64_STANDARD	Yes	Yes	Yes	Yes
AL2_ARM_64	Yes	Yes	Yes*	Yes*
BOTTLEROCKET_ARM_64	Yes**	Yes	N/A	N/A

- * Unable to delete the PV without restarting the node
- ** Doesn't work with NFSv3 with Trident version 25.02.

If your desired AMI is not listed here, it does not mean that it is not supported; it simply means it has not been tested. This list serves as a guide for AMIs are known to work.

Tests performed with:

- EKS version: 1.32
- Installation Method: Helm 25.06 and as an AWS add-On 25.06
- For NAS both NFSv3 and NFSv4.1 were tested.
- For SAN only iSCSI was tested, not NVMe-oF.

Tests performed:

- Create: Storage Class, pvc, pod
- Delete: pod, pvc (regular, qtree/lun – economy, NAS with AWS backup)

Find more information

- Amazon FSx for NetApp ONTAP documentation
- Blog post on Amazon FSx for NetApp ONTAP

Create an IAM role and AWS Secret

You can configure Kubernetes pods to access AWS resources by authenticating as an AWS IAM role instead of by providing explicit AWS credentials.

To authenticate using an AWS IAM role, you must have a Kubernetes cluster deployed using EKS.

Create AWS Secrets Manager secret

Since Trident will be issuing APIs against an FSx vserver to manage the storage for you, it will need credentials to do so. The secure way to pass those credentials is through an AWS Secrets Manager secret. Therefore, if you don't already have one, you'll need to create an AWS Secrets Manager secret that contains the credentials for the vsadmin account.

This example creates an AWS Secrets Manager secret to store Trident CSI credentials:

```
aws secretsmanager create-secret --name trident-secret --description
"Trident CSI credentials"\
  --secret-string
"{\"username\":\"vsadmin\",\"password\":\"<svmpassword>\"}"
```

Create IAM Policy

Trident also needs AWS permissions to run correctly. Therefore, you need to create a policy that gives Trident the permissions it needs.

The following examples creates an IAM policy using the AWS CLI:

```
aws iam create-policy --policy-name AmazonFSxNCSIDriverPolicy --policy
-document file://policy.json
  --description "This policy grants access to Trident CSI to FSxN and
Secrets manager"
```

Policy JSON example:

```
{
  "Statement": [
    {
      "Action": [
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes",
        "fsx:CreateVolume",
        "fsx:RestoreVolumeFromSnapshot",
        "fsx:DescribeStorageVirtualMachines",
        "fsx:UntagResource",
        "fsx:UpdateVolume",
        "fsx:TagResource",
        "fsx:DeleteVolume"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Resource": "arn:aws:secretsmanager:<aws-region>:<aws-account-
id>:secret:<aws-secret-manager-name>*"
    }
  ],
  "Version": "2012-10-17"
}
```

Create Pod Identity or IAM role for Service account association (IRSA)

You can configure a Kubernetes service account to assume an AWS Identity and Access Management (IAM) role with EKS Pod Identity or IAM role for Service account association (IRSA). Any Pods that are configured to use the service account can then access any AWS service that the role has permissions to access.

Pod Identity

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

Install Pod Identity on your EKS cluster:

You can create Pod identity via the AWS console or using the following AWS CLI command:

```
aws eks create-addon --cluster-name <EKS_CLUSTER_NAME> --addon-name
eks-pod-identity-agent
```

For more information refer to Set up the Amazon EKS Pod Identity Agent.

Create trust-relationship.json:

Create trust-relationship.json to enable EKS Service Principal to assume this role for Pod Identity. Then create a role with this trust policy:

```
aws iam create-role \
  --role-name fsxn-csi-role --assume-role-policy-document file://trust-
relationship.json \
  --description "fsxn csi pod identity role"
```

trust-relationship.json file:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

Attach the role policy to the IAM role:

Attach the role policy from the previous step to the IAM role that was created:

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:111122223333:policy/fsxn-csi-policy \
  --role-name fsxn-csi-role
```

Create a pod identity association:

Create a pod identity association between IAM role and the Trident service account(trident-controller)

```
aws eks create-pod-identity-association \
  --cluster-name <EKS_CLUSTER_NAME> \
  --role-arn arn:aws:iam::111122223333:role/fsxn-csi-role \
  --namespace trident --service-account trident-controller
```

IAM role for Service account association (IRSA)

Using the AWS CLI:

```
aws iam create-role --role-name AmazonEKS_FSxN_CSI_DriverRole \
  --assume-role-policy-document file://trust-relationship.json
```

trust-relationship.json file:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<account_id>:oidc-
provider/<oidc_provider>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<oidc_provider>:aud": "sts.amazonaws.com",
          "<oidc_provider>:sub":
"system:serviceaccount:trident:trident-controller"
        }
      }
    }
  ]
}
```

Update the following values in the `trust-relationship.json` file:

- **<account_id>** - Your AWS account ID
- **<oidc_provider>** - The OIDC of your EKS cluster. You can obtain the oidc_provider by running:

```
aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer"\
  --output text | sed -e "s/^https:\/\///"
```

Attach the IAM role with the IAM policy:

Once the role has been created, attach the policy (that was created in the step above) to the role using this command:

```
aws iam attach-role-policy --role-name my-role --policy-arn <IAM policy
ARN>
```

Verify OICD provider is associated:

Verify that your OIDC provider is associated with your cluster. You can verify it using this command:

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

If the output is empty, use the following command to associate IAM OIDC to your cluster:

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name
--approve
```

If you are using eksctl, use the following example to create an IAM role for service account in EKS:

```
eksctl create iamserviceaccount --name trident-controller --namespace
trident \
  --cluster <my-cluster> --role-name AmazonEKS_FSxN_CSI_DriverRole
--role-only \
  --attach-policy-arn <IAM-Policy ARN> --approve
```

Install Trident

Trident streamlines Amazon FSx for NetApp ONTAP storage management in Kubernetes to enable your developers and administrators focus on application deployment. You can install Trident using one of the following methods:

- Helm
- EKS add-on

If you want to make use of the snapshot functionality, install the CSI snapshot controller add-on. Refer to Enable snapshot functionality for CSI volumes for more information.

Install Trident via helm

Pod Identity

1. Add the Trident Helm repository:

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

2. Install Trident using the following example:

```
helm install trident-operator netapp-trident/trident-operator --version 100.2502.1 --namespace trident --create-namespace
```

You can use the `helm list` command to review installation details such as name, namespace, chart, status, app version, and revision number.

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                           APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2502.0    25.02.0
```

Service account association (IRSA)

1. Add the Trident Helm repository:

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

2. Set the values for **cloud provider** and **cloud identity**:

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1  \
--set cloudProvider="AWS" \
--set cloudIdentity="'eks.amazonaws.com/role-arn:
arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'" \
--namespace trident \
--create-namespace
```

You can use the `helm list` command to review installation details such as name, namespace, chart, status, app version, and revision number.

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2510.0    25.10.0
```

If you're planning to use iSCSI, make sure iSCSI is enabled on your client machine. If you're using AL2023 Worker node OS, you can automate the installation of the iSCSI client by adding the node prep parameter in the helm installation:

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1 --namespace trident --create-namespace --
set nodePrep={iscsi}
```

Install Trident via the EKS add-on

The Trident EKS add-on includes the latest security patches, bug fixes, and is validated by AWS to work with Amazon EKS. The EKS add-on enables you to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons.

Prerequisites

Ensure that you have the following before configuring the Trident add-on for AWS EKS:

- An Amazon EKS cluster account with add-on subscription
- AWS permissions to the AWS marketplace:
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI type: Amazon Linux 2 (AL2_x86_64) or Amazon Linux 2 Arm(AL2_ARM_64)
- Node type: AMD or ARM
- An existing Amazon FSx for NetApp ONTAP file system

Enable the Trident add-on for AWS

Management console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. On the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to configure the NetApp Trident CSI add-on for.
4. Select **Add-ons** and then select **Get more add-ons**.
5. Follow these steps to select the add-on:
   a. Scroll down to the **AWS Marketplace add-ons** section and type **"Trident"** in the search box.
   b. Select the check box at the top right corner of the Trident by NetApp box.
   c. Select **Next**.
6. On the **Configure selected add-ons** settings page, do the following:

   Skip these steps if you are using Pod Identity association.

   a. Select the **Version** you would like to use.
   b. If you're using IRSA authentication, make sure to set configuration values available in the Optional configuration settings:
      - Select the **Version** you would like to use.
      - Follow the **Add-on configuration schema** and set the **configurationValues** parameter on the **Configuration values** section to the role-arn you created on the previous step (Value should be in the following format):

```
{

  "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
  "cloudProvider": "AWS"

}
```

   If you select Override for the Conflict resolution method, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

7. Choose **Next**.
8. On the **Review and add** page, choose **Create**.

   After the add-on installation is complete, you see your installed add-on.

AWS CLI

1. Create the `add-on.json` file:

For Pod Identity, use the following format:

Use the

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
}
```

For IRSA authentication, use the following format:

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
  "serviceAccountRoleArn": "<role ARN>",
  "configurationValues": {
    "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
    "cloudProvider": "AWS"
  }
}
```

Replace `<role ARN>` with the ARN of the role that was created in the previous step.

2. Install the Trident EKS add-on.

```
aws eks create-addon --cli-input-json file://add-on.json
```

eksctl

The following example command installs the Trident EKS add-on:

```
eksctl create addon --name netapp_trident-operator --cluster
<cluster_name> --force
```

Update the Trident EKS add-on

Management console

1. Open the Amazon EKS console https://console.aws.amazon.com/eks/home#/clusters.
2. On the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to update the NetApp Trident CSI add-on for.
4. Select the **Add-ons** tab.
5. Select **Trident by NetApp** and then select **Edit**.
6. On the **Configure Trident by NetApp** page, do the following:
   a. Select the **Version** you would like to use.
   b. Expand the **Optional configuration settings** and modify as needed.
   c. Select **Save changes**.

AWS CLI

The following example updates the EKS add-on:

```
aws eks update-addon --cluster-name <eks_cluster_name> --addon-name
netapp_trident-operator --addon-version v25.6.0-eksbuild.1 \
  --service-account-role-arn <role-ARN> --resolve-conflict preserve \
  --configuration-values "{\"cloudIdentity\":
\"'eks.amazonaws.com/role-arn: <role ARN>'\"}"
```

eksctl

- Check the current version of your FSxN Trident CSI add-on. Replace `my-cluster` with your cluster name.

```
eksctl get addon --name netapp_trident-operator --cluster my-cluster
```

Example output:

```
NAME                          VERSION              STATUS    ISSUES
IAMROLE    UPDATE AVAILABLE    CONFIGURATION VALUES
netapp_trident-operator    v25.6.0-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- Update the add-on to the version returned under UPDATE AVAILABLE in the output of the previous step.

```
eksctl update addon --name netapp_trident-operator --version
v25.6.0-eksbuild.1 --cluster my-cluster --force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the Amazon EKS add-on fails; you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on does not manage settings that you need to manage, because those settings are overwritten with this option.
For more information about other options for this setting, see Addons.
For more information about Amazon EKS Kubernetes field management, see Kubernetes field management.

Uninstall/remove the Trident EKS add-on

You have two options for removing an Amazon EKS add-on:

- **Preserve add-on software on your cluster** – This option removes Amazon EKS management of any settings. It also removes the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update. However, it preserves the add-on software on your cluster. This option makes the add-on a self-managed installation, rather than an Amazon EKS add-on. With this option, there's no downtime for the add-on. Retain the `--preserve` option in the command to preserve the add-on.
- **Remove add-on software entirely from your cluster** – NetApp recommends that you remove the Amazon EKS add-on from your cluster only if there are no resources on your cluster that are dependent on it. Remove the `--preserve` option from the `delete` command to remove the add-on.

If the add-on has an IAM account associated with it, the IAM account is not removed.

Management console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. In the left navigation pane, select **Clusters**.
3. Select the name of the cluster that you want to remove the NetApp Trident CSI add-on for.
4. Select the **Add-ons** tab and then select **Trident by NetApp**.*
5. Select **Remove**.
6. In the **Remove netapp_trident-operator confirmation** dialog, do the following:
   a. If you want Amazon EKS to stop managing settings for the add-on, select **Preserve on cluster**. Do this if you want to retain the add-on software on your cluster so that you can manage all of the settings of the add-on on your own.
   b. Enter **netapp_trident-operator**.
   c. Select **Remove**.

AWS CLI

Replace `my-cluster` with the name of your cluster, and then run the following command.

```
aws eks delete-addon --cluster-name my-cluster --addon-name
netapp_trident-operator --preserve
```

eksctl

The following command uninstalls the Trident EKS add-on:

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

Configure a storage class

The Kubernetes StorageClass object identifies a provisioner and instructs the provisioner how to provision volumes. This section shows you how to configure a Kubernetes StorageClass object that specifies Trident as the provisioner.

Create a StorageClass Object

When you create a StorageClass for FSx for ONTAP, Trident will automatically create the backend configuration.

> If you'd like to manually configure the storage backend, please refer to the [create-a-kubernetes-storageclass-without-automatic-backend-configuration] section for how to create the Trident backend and storage class separately.

Specify required StorageClass parameters

The following three parameters need to be defined when creating a StorageClass:

Parameter	Required	Type	Description
`fsxFilesystemID`	Yes	string	FSx for NetApp ONTAP filesystem ID
`storageDriverName`	Yes	string	Trident storage driver (for example, `ontap-nas` or `ontap-san`)
`credentialsName`	Yes	string	Name of the Kubernetes Secret that contains FSx for ONTAP credentials

Specify optional parameters

You can pass optional backend parameters through the StorageClass. Define all optional values as strings in the StorageClass `parameters` section. For a complete list of backend parameters, see: FSx for NetApp ONTAP backend configuration.

Example StorageClass configuration files.

The following example shows a StorageClass that triggers automatic backend configuration.

YAML

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-fsx-demo
  annotations:
    description: "Demo StorageClass for FSx for NetApp ONTAP"
provisioner: csi.trident.netapp.io
parameters:
  fsxFilesystemID: "fs-0abc123"
  storageDriverName: "ontap-nas"
  credentialsName: trident-fsx-credentials
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

Create the StorageClass

Once you have created you configuration file, run the following command to create the storage class.

```
kubectl create -f storage-class-ontapnas.yaml
```

You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
```

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

After you apply the StorageClass, Trident creates the backend automatically. You can then create PersistentVolumeClaims that reference this StorageClass.

Verify backend configuration status

Trident records the result of backend creation in StorageClass annotations.

Annotation	Description
`trident.netapp.io/configuratorStatus`	Configuration result (`Success` or `Failure`)
`trident.netapp.io/configuratorMessage`	Detailed status or error message
`trident.netapp.io/configuratorName`	Name of the internal configurator resource
`trident.netapp.io/managed`	Indicates the StorageClass is managed by Trident
`trident.netapp.io/additionalStoragePools`	Storage pools created for this backend

To verify status, run:

```
kubectl get storageclass ontap-fsx-demo -o yaml
```

Confirm that `trident.netapp.io/configuratorStatus` is set to `Success`. If the value is `Failure`, review `trident.netapp.io/configuratorMessage` for the error.

Add additional FSxN file systems

If you need additional storage capacity while continuing to use the same StorageClass, add additional FSxN file system IDs.

Edit the StorageClass and add the following annotation:

```
metadata:
  annotations:
    trident.netapp.io/additionalFsxNFileSystemID: '["fs-
xxxxxxxxxxxxxxxxx"]'
```

After you apply the change, Trident updates the backend configuration and updates the StorageClass annotations.

Operational considerations and limitations

- Deleting a StorageClass that as the automatic backend configuration usually deletes the associated Trident backend. This can disrupt storage connectivity and break running workloads. Validate the impact before you delete a managed StorageClass.
- Automatic backend configuration is supported only for AWS FSx for NetApp ONTAP.

Create a Kubernetes StorageClass without automatic backend configuration

If you want to create the Trident backend and StorageClass separately then follow these steps.

Understand how automatic backend configuration works

Trident derives backend configuration from the StorageClass definition. When you apply the StorageClass, Trident validates the required parameters, creates the backend, and annotates the StorageClass with status.

Trident creates the VolumeSnapshotClass only once. Trident reuses the same VolumeSnapshotClass for subsequent StorageClasses.

Create the Trident backend

To create a Trident backend, you need to create a configuration file in either JSON or YAML format. The file needs to specify the type of storage you want (NAS or SAN), the file system, and SVM to get it from and how to authenticate with it. The following example shows how to define NAS-based storage and using an AWS secret to store the credentials to the SVM you want to use:

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

JSON

```
{
  "apiVersion": "trident.netapp.io/v1",
  "kind": "TridentBackendConfig",
  "metadata": {
    "name": "backend-tbc-ontap-nas"
    "namespace": "trident"
  },
  "spec": {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx"
    },
    "managementLIF": null,
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name",
      "type": "awsarn"
    }
  }
}
```

FSx for ONTAP driver details

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

Driver Name	Description
`ontap-san`	Each PV provisioned is a LUN within its own Amazon FSx for NetApp ONTAP volume. Recommended for block storage.
`ontap-nas`	Each PV provisioned is a full Amazon FSx for NetApp ONTAP volume. Recommended for NFS and SMB.
`ontap-san-economy`	Each PV provisioned is a LUN with a configurable number of LUNs per Amazon FSx for NetApp ONTAP volume.
`ontap-nas-economy`	Each PV provisioned is a qtree, with a configurable number of qtrees per Amazon FSx for NetApp ONTAP volume.
`ontap-nas-flexgroup`	Each PV provisioned is a full Amazon FSx for NetApp ONTAP FlexGroup volume.

For driver details, refer to NAS drivers and SAN drivers.

Create the backend

After creating the configuration file run the following commands to create and validate the Trident Backend Configuration (TBC):

- Create trident backend configuration (TBC) from yaml file and run the following command:

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- Validate the trident backend configuration (TBC) was created successfully:

```
Kubectl get tbc -n trident
```

```
NAME                          BACKEND NAME          BACKEND UUID
PHASE   STATUS

backend-tbc-ontap-nas         tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9   Bound   Success
```

For more information on other configuration options, see the [Backend-advanced-configuration-and-examples] section below.

Configure a Storage Class without automatic backend configuration

The following are examples of Storage Class configurations for use with Trident and FSx for ONTAP.

Storage Class for NFS

You can use this example to setup StorageClass for volumes using NFS (Refer to Trident Attribute section below for the full list of attributes):

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

Storage Calss for iSCSI

Use this example to setup StorageClass for volumes using iSCSI:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

Storage Class using NFSv3 and AWS Bottlerocket

To provision NFSv3 volumes on AWS Bottlerocket, add the required `mountOptions` to the storage class:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

Trident StorageClass attributes

These parameters determine which Trident-managed storage pools should be utilized to provision volumes of a given type.

Attribute	Type	Values	Offer	Request	Supported by
media[1]	string	hdd, hybrid, ssd	Pool contains media of this type; hybrid means both	Media type specified	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san
provisioningType	string	thin, thick	Pool supports this provisioning method	Provisioning method specified	thick: all ontap; thin: all ontap & solidfire-san
backendType	string	ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, solidfire-san, azure-netapp-files, ontap-san-economy	Pool belongs to this type of backend	Backend specified	All drivers
snapshots	bool	true, false	Pool supports volumes with snapshots	Volume with snapshots enabled	ontap-nas, ontap-san, solidfire-san
clones	bool	true, false	Pool supports cloning volumes	Volume with clones enabled	ontap-nas, ontap-san, solidfire-san

Attribute	Type	Values	Offer	Request	Supported by
encryption	bool	true, false	Pool supports encrypted volumes	Volume with encryption enabled	ontap-nas, ontap-nas-economy, ontap-nas-flexgroups, ontap-san
IOPS	int	positive integer	Pool is capable of guaranteeing IOPS in this range	Volume guaranteed these IOPS	solidfire-san

[1]: Not supported by ONTAP Select or FSx for ONTAP systems

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Create the storage class

Once you have configured the StorageClass, you can create it in Kubernetes.

Steps

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

```
kubectl create -f storage-class-ontapnas.yaml
```

2. You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
```

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

Provision SMB volumes

You can provision SMB volumes using the `ontap-nas` driver.
However, to do so you must complete these steps: Prepare to provision SMB volumes.

Backend advanced configuration and examples

See the following table for the backend configuration options:

Parameter	Description	Example
`version`		Always 1

Parameter	Description	Example
`storageDriverName`	Name of the storage driver	`ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`
`backendName`	Custom name or the storage backend	Driver name + "_" + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF A fully-qualified domain name (FQDN) can be specified. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]. If you provide the `fsxFilesystemID` under the `aws` field, you need not to provide the `managementLIF` because Trident retrieves the SVM `managementLIF` information from AWS. So, you must provide credentials for a user under the SVM (For example: vsadmin) and the user must have the `vsadmin` role.	"10.0.0.1", "[2001:1234:abcd::fefe]"

Parameter	Description	Example
`dataLIF`	IP address of protocol LIF. **ONTAP NAS drivers**: NetApp recommends specifying dataLIF. If not provided, Trident fetches dataLIFs from the SVM. You can specify a fully-qualified domain name (FQDN) to be used for the NFS mount operations, allowing you to create a round-robin DNS to load-balance across multiple dataLIFs. Can be changed after initial setting. **ONTAP SAN drivers**: Do not specify for iSCSI. Trident uses ONTAP Selective LUN Map to discover the iSCI LIFs needed to establish a multi path session. A warning is generated if dataLIF is explicitly defined. Can be set to use IPv6 addresses if Trident was installed using the IPv6 flag. IPv6 addresses must be defined in square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555].	
`autoExportPolicy`	Enable automatic export policy creation and updating [Boolean]. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	`false`
`autoExportCIDRs`	List of CIDRs to filter Kubernetes' node IPs against when `autoExportPolicy` is enabled. Using the `autoExportPolicy` and `autoExportCIDRs` options, Trident can manage export policies automatically.	"["0.0.0.0/0", "::/0"]"
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	""
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	""

Parameter	Description	Example
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	""
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based authentication.	""
`username`	Username to connect to the cluster or SVM. Used for credential-based authentication. For example, vsadmin.	
`password`	Password to connect to the cluster or SVM. Used for credential-based authentication.	
`svm`	Storage virtual machine to use	Derived if an SVM managementLIF is specified.
`storagePrefix`	Prefix used when provisioning new volumes in the SVM. Cannot be modified after creation. To update this parameter, you will need to create a new backend.	`trident`
`limitAggregateUsage`	**Do not specify for Amazon FSx for NetApp ONTAP.** The provided `fsxadmin` and `vsadmin` do not contain the permissions required to retrieve aggregate usage and limit it using Trident.	Do not use.
`limitVolumeSize`	Fail provisioning if requested volume size is above this value. Also restricts the maximum size of the volumes it manages for qtrees and LUNs, and the `qtreesPerFlexvol` option allows customizing the maximum number of qtrees per FlexVol volume	"" (not enforced by default)
`lunsPerFlexvol`	Maximum LUNs per Flexvol volume, must be in range [50, 200]. SAN only.	“100”

Parameter	Description	Example
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, {"api":false, "method":true} Do not use `debugTraceFlags` unless you are troubleshooting and require a detailed log dump.	null
`nfsMountOptions`	Comma-separated list of NFS mount options. The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in the storage class or the configuration file, Trident will not set any mount options on an associated persistent volume.	""
`nasType`	Configure NFS or SMB volumes creation. Options are `nfs`, `smb`, or null. **Must set to `smb` for SMB volumes.** Setting to null defaults to NFS volumes.	`nfs`
`qtreesPerFlexvol`	Maximum Qtrees per FlexVol volume, must be in range [50, 300]	`"200"`
`smbShare`	You can specify one of the following: the name of an SMB share created using the Microsoft Management Console or ONTAP CLI or a name to allow Trident to create the SMB share. This parameter is required for Amazon FSx for ONTAP backends.	`smb-share`

Parameter	Description	Example
useREST	Boolean parameter to use ONTAP REST APIs. When set to true, Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the ontap application. This is satisfied by the pre-defined vsadmin and cluster-admin roles.	false
aws	You can specify the following in the configuration file for AWS FSx for ONTAP: - fsxFilesystemID: Specify the ID of the AWS FSx file system. - apiRegion: AWS API region name. - apikey: AWS API key. - secretKey: AWS secret key.	"" "" ""
credentials	Specify the FSx SVM credentials to store in AWS Secrets Manager. - name: Amazon Resource Name (ARN) of the secret, which contains the credentials of SVM. - type: Set to awsarn. Refer to Create an AWS Secrets Manager secret for more information.	

Backend configuration options for provisioning volumes

You can control default provisioning using these options in the defaults section of the configuration. For an example, see the configuration examples below.

Parameter	Description	Default
spaceAllocation	Space-allocation for LUNs	true
spaceReserve	Space reservation mode; "none" (thin) or "volume" (thick)	none
snapshotPolicy	Snapshot policy to use	none

Parameter	Description	Default
`qosPolicy`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend. Using QoS policy groups with Trident requires ONTAP 9.8 or later. You should use a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group enforces the ceiling for the total throughput of all workloads.	""
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend. Not supported by ontap-nas-economy.	""
`snapshotReserve`	Percentage of volume reserved for snapshots "0"	If `snapshotPolicy` is `none`, `else` ""
`splitOnClone`	Split a clone from its parent upon creation	`false`
`encryption`	Enable NetApp Volume Encryption (NVE) on the new volume; defaults to `false`. NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Trident will be NAE enabled. For more information, refer to: How Trident works with NVE and NAE.	`false`
`luksEncryption`	Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS). SAN only.	""
`tieringPolicy`	Tiering policy to use `none`	
`unixPermissions`	Mode for new volumes. **Leave empty for SMB volumes.**	""

Parameter	Description	Default
`securityStyle`	Security style for new volumes. NFS supports `mixed` and `unix` security styles. SMB supports `mixed` and `ntfs` security styles.	NFS default is `unix`. SMB default is `ntfs`.

Configure a PVC

This sections includes instructions on how to create a PersistentVolumeClaim (PVC) that uses the configured Kubernetes StorageClass to request a PV.
If successful, you can then mount the PV to a pod.

Create the PVC

A *PersistentVolumeClaim* (PVC) is a request for access to the PersistentVolume on the cluster. The PVC can be configured to request storage of a certain size or access mode. Using the associated StorageClass, the cluster administrator can control more than PersistentVolume size and access mode—such as performance or service level.

After you create the Trident backend and StorageClass you can create a PVC. After the PVC is created, you can mount the volume in a pod.

Sample manifests

The following examples show basic PVC configuration options.

PVC with RWX access

This example shows a basic PVC with RWX access that is associated with a StorageClass named `basic-csi`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-gold
```

PVC using iSCSI example

This example shows a basic PVC for iSCSI with RWO access that is associated with a StorageClass named `protection-gold`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: protection-gold
```

Create PVC

Steps

1. Create the PVC.

```
kubectl create -f pvc.yaml
```

2. Verify the PVC status.

```
kubectl get pvc
```

```
NAME        STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                          5m
```

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Deploy an application

When the storage class and PVC are created, you can mount the PV to a pod. This section lists the example command and configuration to attach the PV to a pod.

Deploy a sample application

Steps

1. Mount the volume in a pod.

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

These examples show basic configurations to attach the PVC to a pod:

Basic configuration:

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

You can monitor the progress using `kubectl get pod --watch`.

2. Verify that the volume is mounted on `/my/mount/path`.

```
kubectl exec -it pv-pod -- df -h /my/mount/path
```

```
Filesystem                                                      Size
Used  Avail Use%  Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1%   /my/mount/path
```

You can now delete the Pod. The Pod application will no longer exist, but the volume will remain.

```
kubectl delete pod pv-pod
```

Configure the Trident EKS add-on on an EKS cluster

NetApp Trident streamlines Amazon FSx for NetApp ONTAP storage management in Kubernetes to enable your developers and administrators focus on application deployment. The NetApp Trident EKS add-on includes the latest security patches, bug fixes, and is validated by AWS to work with Amazon EKS. The EKS add-on enables you

to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons.

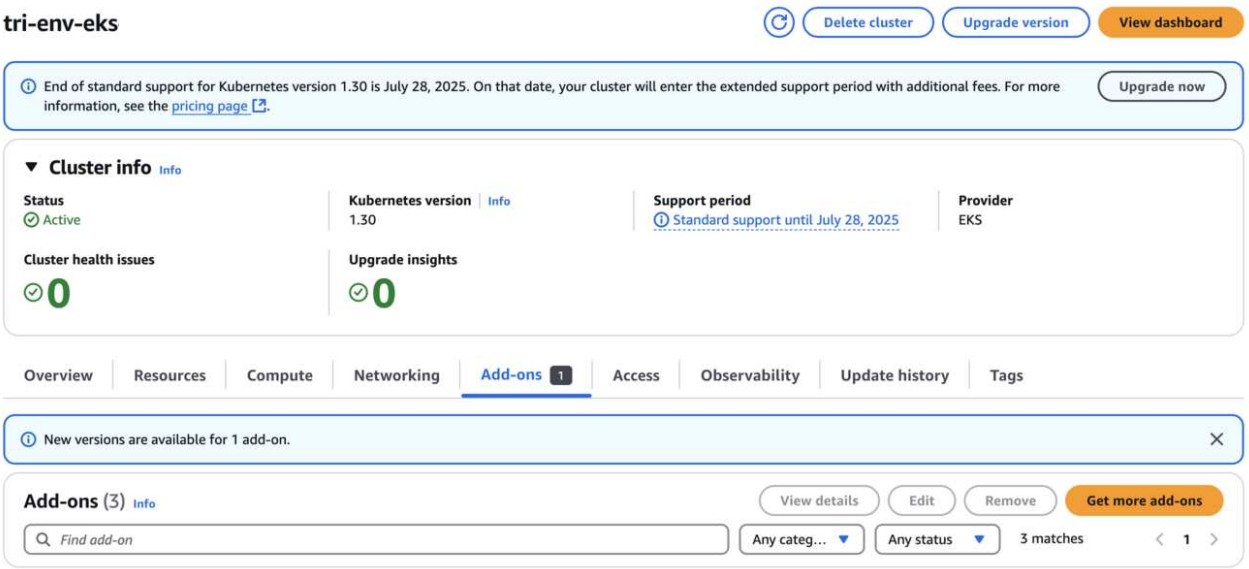
Prerequisites

Ensure that you have the following before configuring the Trident add-on for AWS EKS:

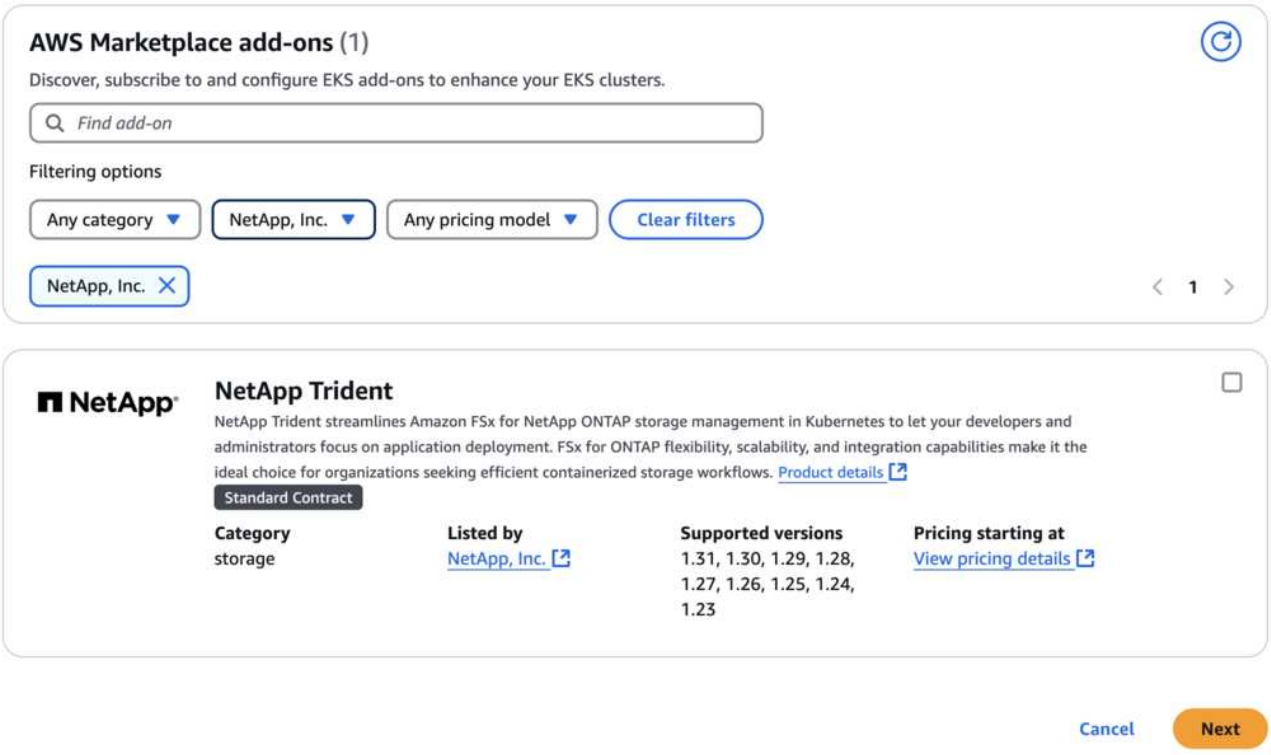
- An Amazon EKS cluster account with permissions to work with add-ons. Refer to Amazon EKS add-ons.
- AWS permissions to the AWS marketplace:
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMI type: Amazon Linux 2 (AL2_x86_64) or Amazon Linux 2 Arm(AL2_ARM_64)
- Node type: AMD or ARM
- An existing Amazon FSx for NetApp ONTAP file system

Steps

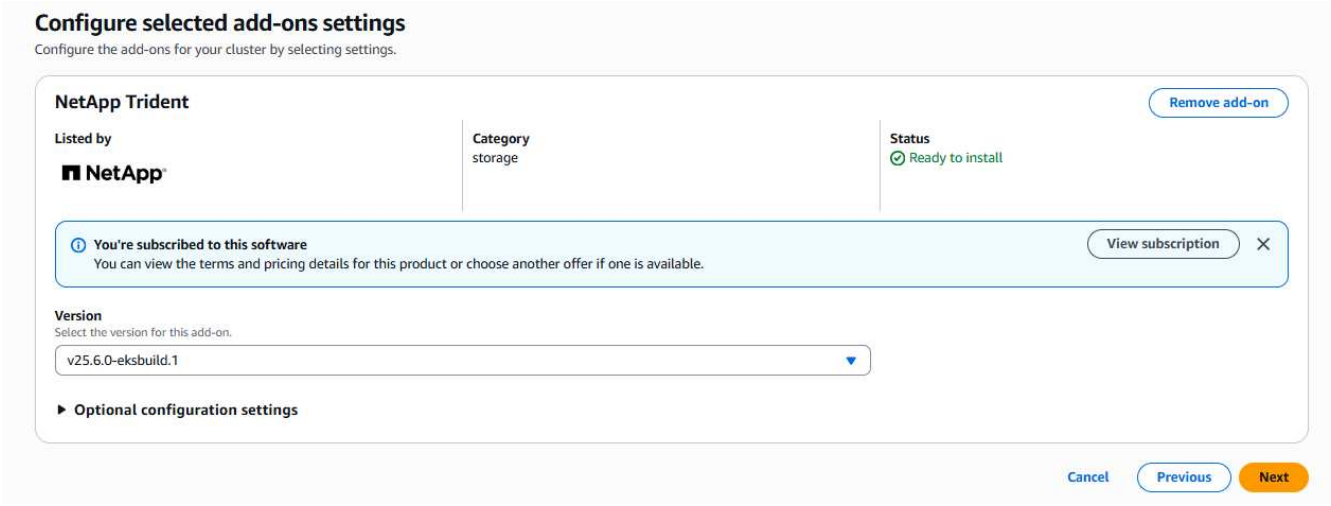
1. Make sure to create IAM role and AWS secret to enable EKS pods to access AWS resources. For instructions, see Create an IAM role and AWS Secret.
2. On your EKS Kubernetes cluster, navigate to the **Add-ons** tab.



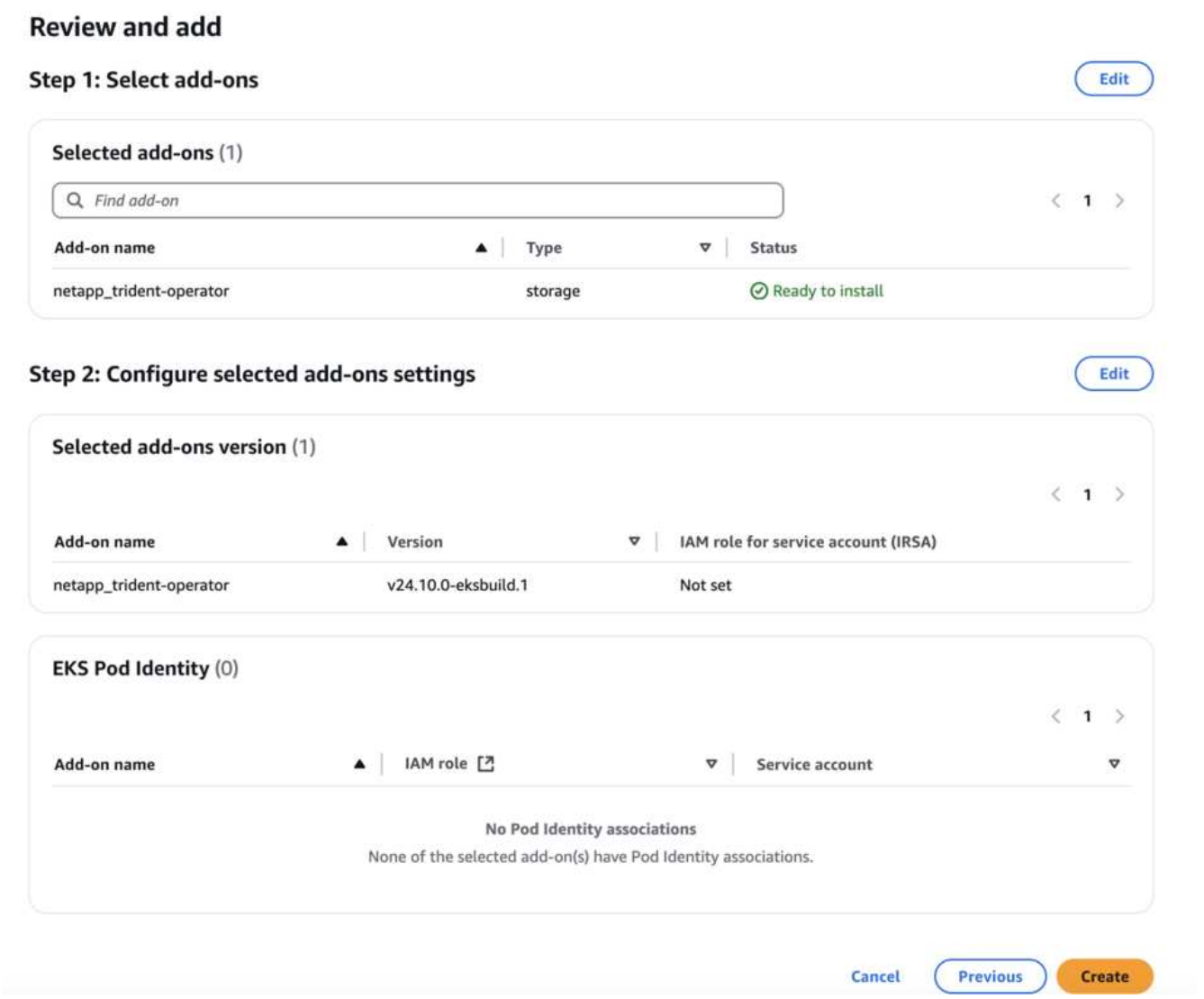
3. Go to **AWS Marketplace add-ons** and choose the *storage* category.



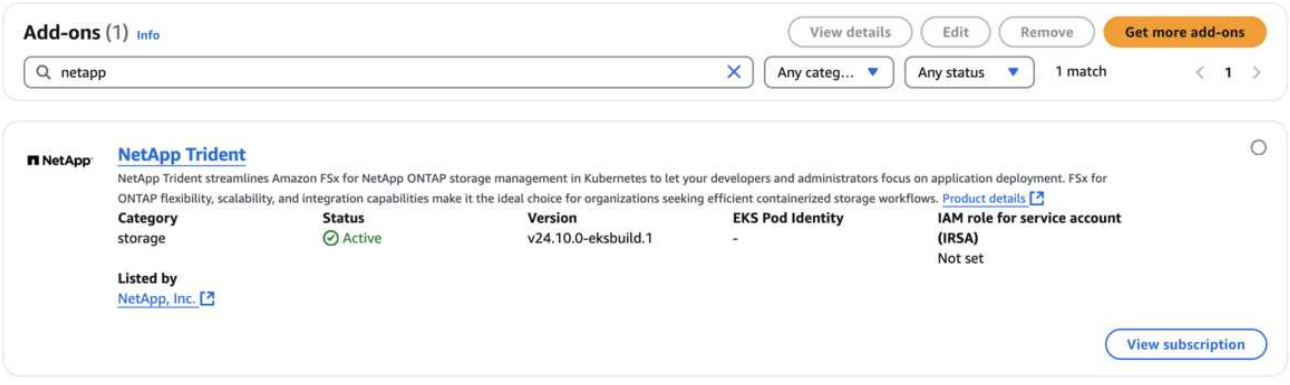
4. Locate **NetApp Trident** and select the checkbox for the Trident add-on, and click **Next**.
5. Choose the desired version of the add-on.



6. Configure the required add-on settings.



7. If you are using IRSA (IAM roles for service account), refer to the additional configuration steps here.
8. Select **Create**.
9. Verify that the status of the add-on is *Active*.



10. Run the following command to verify that Trident is properly installed on the cluster:

```
kubectl get pods -n trident
```

11. Continue the setup and configure the storage backend. For information, see Configure the Storage Backend.

Install/uninstall the Trident EKS add-on using CLI

Install the NetApp Trident EKS add-on using CLI:

The following example command installs the Trident EKS add-on:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.0-eksbuild.1` (with a dedicated version)

The following example command installs the Trident EKS add-on version 25.6.1:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.1-eksbuild.1` (with a dedicated version)

The following example command installs the Trident EKS add-on version 25.6.2:
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.2-eksbuild.1` (with a dedicated version)

Uninstall the NetApp Trident EKS add-on using CLI:

The following command uninstalls the Trident EKS add-on:

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

Create backends with kubectl

A backend defines the relationship between Trident and a storage system. It tells Trident how to communicate with that storage system and how Trident should provision volumes from it. After Trident is installed, the next step is to create a backend. The `TridentBackendConfig` Custom Resource Definition (CRD) enables you to create and manage Trident backends directly through the Kubernetes interface. You can do this by using `kubectl` or the equivalent CLI tool for your Kubernetes distribution.

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`, `tbconfig`, `tbackendconfig`) is a frontend, namespaced CRD that enables you to manage Trident backends using `kubectl`. Kubernetes and storage admins can now create and manage backends directly through the Kubernetes CLI without requiring a dedicated command-line utility (`tridentctl`).

Upon the creation of a `TridentBackendConfig` object, the following happens:

- A backend is created automatically by Trident based on the configuration you provide. This is represented internally as a `TridentBackend` (`tbe`, `tridentbackend`) CR.
- The `TridentBackendConfig` is uniquely bound to a `TridentBackend` that was created by Trident.

Each `TridentBackendConfig` maintains a one-to-one mapping with a `TridentBackend`. The former is the interface provided to the user to design and configure backends; the latter is how Trident represents the actual backend object.

> `TridentBackend` CRs are created automatically by Trident. You **should not** modify them. If you want to make updates to backends, do this by modifying the `TridentBackendConfig` object.

See the following example for the format of the `TridentBackendConfig` CR:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

You can also take a look at the examples in the trident-installer directory for sample configurations for the desired storage platform/service.

The `spec` takes backend-specific configuration parameters. In this example, the backend uses the `ontap-san` storage driver and uses the configuration parameters that are tabulated here. For the list of configuration options for your desired storage driver, refer to the backend configuration information for your storage driver.

The `spec` section also includes `credentials` and `deletionPolicy` fields, which are newly introduced in the `TridentBackendConfig` CR:

- `credentials`: This parameter is a required field and contains the credentials used to authenticate with the storage system/service. This is set to a user-created Kubernetes Secret. The credentials cannot be passed in plain text and will result in an error.
- `deletionPolicy`: This field defines what should happen when the `TridentBackendConfig` is deleted. It can take one of two possible values:
  - `delete`: This results in the deletion of both `TridentBackendConfig` CR and the associated backend. This is the default value.
  - `retain`: When a `TridentBackendConfig` CR is deleted, the backend definition will still be present and can be managed with `tridentctl`. Setting the deletion policy to `retain` lets users downgrade to an earlier release (pre-21.04) and retain the created backends. The value for this field can be updated after a `TridentBackendConfig` is created.

The name of a backend is set using `spec.backendName`. If unspecified, the name of the backend is set to the name of the `TridentBackendConfig` object (metadata.name). It is recommended to explicitly set backend names using `spec.backendName`.

Backends that were created with `tridentctl` do not have an associated `TridentBackendConfig` object. You can choose to manage such backends with `kubectl` by creating a `TridentBackendConfig` CR. Care must be taken to specify identical config parameters (such as `spec.backendName`, `spec.storagePrefix`, `spec.storageDriverName`, and so on). Trident will automatically bind the newly-created `TridentBackendConfig` with the pre-existing backend.

Steps overview

To create a new backend by using `kubectl`, you should do the following:

1. Create a Kubernetes Secret. The secret contains the credentials Trident needs to communicate with the storage cluster/service.
2. Create a `TridentBackendConfig` object. This contains specifics about the storage cluster/service and references the secret created in the previous step.

After you create a backend, you can observe its status by using `kubectl get tbc <tbc-name> -n <trident-namespace>` and gather additional details.

Step 1: Create a Kubernetes Secret

Create a Secret that contains the access credentials for the backend. This is unique to each storage service/platform. Here's an example:

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: password
```

This table summarizes the fields that must be included in the Secret for each storage platform:

Storage platform Secret Fields description	Secret	Fields description
Azure NetApp Files	clientID	The client ID from an app registration

Storage platform Secret Fields description	Secret	Fields description
Element (NetApp HCI/SolidFire)	Endpoint	MVIP for the SolidFire cluster with tenant credentials
ONTAP	username	Username to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	password	Password to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	clientPrivateKey	Base64-encoded value of client private key. Used for certificate-based authentication
ONTAP	chapUsername	Inbound username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetUsername	Target username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP target initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`

The Secret created in this step will be referenced in the `spec.credentials` field of the `TridentBackendConfig` object that is created in the next step.

Step 2: Create the `TridentBackendConfig` CR

You are now ready to create your `TridentBackendConfig` CR. In this example, a backend that uses the `ontap-san` driver is created by using the `TridentBackendConfig` object shown below:

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

Step 3: Verify the status of the `TridentBackendConfig` CR

Now that you created the `TridentBackendConfig` CR, you can verify the status. See the following example:

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                     BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

A backend was successfully created and bound to the `TridentBackendConfig` CR.

Phase can take one of the following values:

- `Bound`: The `TridentBackendConfig` CR is associated with a backend, and that backend contains `configRef` set to the `TridentBackendConfig` CR's uid.
- `Unbound`: Represented using `""`. The `TridentBackendConfig` object is not bound to a backend. All newly created `TridentBackendConfig` CRs are in this phase by default. After the phase changes, it cannot revert to Unbound again.
- `Deleting`: The `TridentBackendConfig` CR's `deletionPolicy` was set to delete. When the `TridentBackendConfig` CR is deleted, it transitions to the Deleting state.
  - If no persistent volume claims (PVCs) exist on the backend, deleting the `TridentBackendConfig` will result in Trident deleting the backend as well as the `TridentBackendConfig` CR.
  - If one or more PVCs are present on the backend, it goes to a deleting state. The `TridentBackendConfig` CR subsequently also enters deleting phase. The backend and `TridentBackendConfig` are deleted only after all PVCs are deleted.
- `Lost`: The backend associated with the `TridentBackendConfig` CR was accidentally or deliberately deleted and the `TridentBackendConfig` CR still has a reference to the deleted backend. The `TridentBackendConfig` CR can still be deleted irrespective of the `deletionPolicy` value.

- `Unknown`: Trident is unable to determine the state or existence of the backend associated with the `TridentBackendConfig` CR. For example, if the API server is not responding or if the `tridentbackends.trident.netapp.io` CRD is missing. This might require intervention.

At this stage, a backend is successfully created! There are several operations that can additionally be handled, such as backend updates and backend deletions.

(Optional) Step 4: Get more details

You can run the following command to get more information about your backend:

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

```
NAME                      BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success   ontap-san        delete
```

In addition, you can also obtain a YAML/JSON dump of `TridentBackendConfig`.

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo` contains the `backendName` and the `backendUUID` of the backend that got created in response to the `TridentBackendConfig` CR. The `lastOperationStatus` field represents the status of the last operation of the `TridentBackendConfig` CR, which can be user-triggered (for example, user changed something in `spec`) or triggered by Trident (for example, during Trident restarts). It can either be Success or Failed. `phase` represents the status of the relation between the `TridentBackendConfig` CR and the backend. In the example above, `phase` has the value Bound, which means that the `TridentBackendConfig` CR is associated with the backend.

You can run the `kubectl -n trident describe tbc <tbc-cr-name>` command to get details of the event logs.

You cannot update or delete a backend which contains an associated `TridentBackendConfig` object using `tridentctl`. To understand the steps involved in switching between `tridentctl` and `TridentBackendConfig`, see here.

Manage backends

Perform backend management with kubectl

Learn about how to perform backend management operations by using `kubectl`.

Delete a backend

By deleting a `TridentBackendConfig`, you instruct Trident to delete/retain backends (based on `deletionPolicy`). To delete a backend, ensure that `deletionPolicy` is set to delete. To delete just the `TridentBackendConfig`, ensure that `deletionPolicy` is set to retain. This ensures the backend is still present and can be managed by using `tridentctl`.

Run the following command:

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

Trident does not delete the Kubernetes Secrets that were in use by `TridentBackendConfig`. The Kubernetes user is responsible for cleaning up secrets. Care must be taken when deleting secrets. You should delete secrets only if they are not in use by the backends.

View the existing backends

Run the following command:

```
kubectl get tbc -n trident
```

You can also run `tridentctl get backend -n trident` or `tridentctl get backend -o yaml -n trident` to obtain a list of all backends that exist. This list will also include backends that were created with `tridentctl`.

Update a backend

There can be multiple reasons to update a backend:

- Credentials to the storage system have changed. To update credentials, the Kubernetes Secret that is used in the `TridentBackendConfig` object must be updated. Trident will automatically update the backend with the latest credentials provided. Run the following command to update the Kubernetes Secret:

  ```
  kubectl apply -f <updated-secret-file.yaml> -n trident
  ```

- Parameters (such as the name of the ONTAP SVM being used) need to be updated.
  - You can update `TridentBackendConfig` objects directly through Kubernetes using the following command:

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

- Alternatively, you can make changes to the existing `TridentBackendConfig` CR using the following command:

```
kubectl edit tbc <tbc-name> -n trident
```

- If a backend update fails, the backend continues to remain in its last known configuration. You can view the logs to determine the cause by running `kubectl get tbc <tbc-name> -o yaml -n trident` or `kubectl describe tbc <tbc-name> -n trident`.
- After you identify and correct the problem with the configuration file, you can re-run the update command.

Perform backend management with tridentctl

Learn about how to perform backend management operations by using `tridentctl`.

Create a backend

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

```
tridentctl create backend -f <backend-file> -n trident
```

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

```
tridentctl logs -n trident
```

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

Delete a backend

To delete a backend from Trident, do the following:

1. Retrieve the backend name:

```
tridentctl get backend -n trident
```

2. Delete the backend:

```
tridentctl delete backend <backend-name> -n trident
```

If Trident has provisioned volumes and snapshots from this backend that still exist, deleting the backend prevents new volumes from being provisioned by it. The backend will continue to exist in a "Deleting" state.

View the existing backends

To view the backends that Trident knows about, do the following:

- To get a summary, run the following command:

```
tridentctl get backend -n trident
```

- To get all the details, run the following command:

```
tridentctl get backend -o json -n trident
```

Update a backend

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

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

If backend update fails, something was wrong with the backend configuration or you attempted an invalid update. You can view the logs to determine the cause by running the following command:

```
tridentctl logs -n trident
```

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

Identify the storage classes that use a backend

This is an example of the kind of questions you can answer with the JSON that `tridentctl` outputs for backend objects. This uses the `jq` utility, which you need to install.

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

This also applies for backends that were created by using `TridentBackendConfig`.

Move between backend management options

Learn about the different ways of managing backends in Trident.

Options for managing backends

With the introduction of `TridentBackendConfig`, administrators now have two unique ways of managing backends. This poses the following questions:

- Can backends created using `tridentctl` be managed with `TridentBackendConfig`?
- Can backends created using `TridentBackendConfig` be managed using `tridentctl`?

Manage `tridentctl` backends using `TridentBackendConfig`

This section covers the steps required to manage backends that were created using `tridentctl` directly through the Kubernetes interface by creating `TridentBackendConfig` objects.

This will apply to the following scenarios:

- Pre-existing backends, that don't have a `TridentBackendConfig` because they were created with `tridentctl`.
- New backends that were created with `tridentctl`, while other `TridentBackendConfig` objects exist.

In both scenarios, backends will continue to be present, with Trident scheduling volumes and operating on them. Administrators have one of two choices here:

- Continue using `tridentctl` to manage backends that were created using it.
- Bind backends created using `tridentctl` to a new `TridentBackendConfig` object. Doing so would mean the backends will be managed using `kubectl` and not `tridentctl`.

To manage a pre-existing backend using `kubectl`, you will need to create a `TridentBackendConfig` that binds to the existing backend. Here is an overview of how that works:

1. Create a Kubernetes Secret. The secret contains the credentials Trident needs to communicate with the storage cluster/service.
2. Create a `TridentBackendConfig` object. This contains specifics about the storage cluster/service and references the secret created in the previous step. Care must be taken to specify identical config parameters (such as `spec.backendName`, `spec.storagePrefix`, `spec.storageDriverName`, and so on). `spec.backendName` must be set to the name of the existing backend.

Step 0: Identify the backend

To create a `TridentBackendConfig` that binds to an existing backend, you will need to obtain the backend configuration. In this example, let us assume a backend was created using the following JSON definition:

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

Step 1: Create a Kubernetes Secret

Create a Secret that contains the credentials for the backend, as shown in this example:

```
cat tbc-ontap-nas-backend-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password
```

```
kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

Step 2: Create a `TridentBackendConfig` CR

The next step is to create a `TridentBackendConfig` CR that will automatically bind to the pre-existing `ontap-nas-backend` (as in this example). Ensure the following requirements are met:

- The same backend name is defined in `spec.backendName`.
- Configuration parameters are identical to the original backend.
- Virtual pools (if present) must retain the same order as in the original backend.
- Credentials are provided through a Kubernetes Secret and not in plain text.

In this case, the `TridentBackendConfig` will look like this:

```
cat backend-tbc-ontap-nas.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

Step 3: Verify the status of the `TridentBackendConfig` CR

After the `TridentBackendConfig` has been created, its phase must be `Bound`. It should also reflect the same backend name and UUID as that of the existing backend.

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

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

The backend will now be completely managed using the `tbc-ontap-nas-backend` `TridentBackendConfig` object.

Manage `TridentBackendConfig` backends using `tridentctl`

`tridentctl` can be used to list backends that were created using `TridentBackendConfig`. In addition, administrators can also choose to completely manage such backends through `tridentctl` by deleting `TridentBackendConfig` and making sure `spec.deletionPolicy` is set to `retain`.

Step 0: Identify the backend

For example, let us assume the following backend was created using `TridentBackendConfig`:

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

From the output, it is seen that `TridentBackendConfig` was created successfully and is bound to a backend [observe the backend's UUID].

Step 1: Confirm `deletionPolicy` is set to `retain`

Let us take a look at the value of `deletionPolicy`. This needs to be set to `retain`. This ensures that when a `TridentBackendConfig` CR is deleted, the backend definition will still be present and can be managed with `tridentctl`.

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

Do not proceed to the next step unless `deletionPolicy` is set to `retain`.

Step 2: Delete the `TridentBackendConfig` CR

The final step is to delete the `TridentBackendConfig` CR. After confirming the `deletionPolicy` is set to `retain`, you can go ahead with the deletion:

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE   | VOLUMES |
+------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+------------------+----------------
+--------------------------------------+--------+---------+
```

Upon the deletion of the `TridentBackendConfig` object, Trident simply removes it without actually deleting the backend itself.

Create and manage storage classes

Create a storage class

Configure a Kubernetes StorageClass object and create the storage class to instruct Trident how to provision volumes.

Configure a Kubernetes StorageClass object

The Kubernetes StorageClass object identifies Trident as the provisioner that is used for that class and instructs Trident how to provision a volume. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Create a storage class

After you create the StorageClass object, you can create the storage class. Storage class samples provides some basic samples you can use or modify.

Steps

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

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

2. You should now see a **basic-csi** storage class in both Kubernetes and Trident, and Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
```

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

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```


Storage class samples

Trident provides simple storage class definitions for specific backends.

Alternatively, you can edit `sample-input/storage-class-csi.yaml.templ` file that comes with the installer and replace *`BACKEND_TYPE`* with the storage driver name.

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

Manage storage classes

You can view existing storage classes, set a default storage class, identify the storage class backend, and delete storage classes.

View the existing storage classes

- To view existing Kubernetes storage classes, run the following command:

```
kubectl get storageclass
```

- To view Kubernetes storage class detail, run the following command:

```
kubectl get storageclass <storage-class> -o json
```

- To view Trident's synchronized storage classes, run the following command:

```
tridentctl get storageclass
```

- To view Trident's synchronized storage class detail, run the following command:

```
tridentctl get storageclass <storage-class> -o json
```

Set a default storage class

Kubernetes 1.6 added the ability to set a default storage class. This is the storage class that will be used to provision a Persistent Volume if a user does not specify one in a Persistent Volume Claim (PVC).

- Define a default storage class by setting the annotation `storageclass.kubernetes.io/is-default-class` to true in the storage class definition. According to the specification, any other value or absence of the annotation is interpreted as false.
- You can configure an existing storage class to be the default storage class by using the following command:

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- Similarly, you can remove the default storage class annotation by using the following command:

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

There are also examples in the Trident installer bundle that include this annotation.

> There should be only one default storage class in your cluster at a time. Kubernetes does not technically prevent you from having more than one, but it will behave as if there is no default storage class at all.

Identify the backend for a storage class

This is an example of the kind of questions you can answer with the JSON that `tridentctl` outputs for Trident backend objects. This uses the `jq` utility, which you may need to install first.

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

Delete a storage class

To delete a storage class from Kubernetes, run the following command:

```
kubectl delete storageclass <storage-class>
```

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

Any persistent volumes that were created through this storage class will remain untouched, and Trident will continue to manage them.

Trident enforces a blank `fsType` for the volumes it creates. For iSCSI backends, it is recommended to enforce `parameters.fsType` in the StorageClass. You should delete existing StorageClasses and re-create them with `parameters.fsType` specified.

Provision and manage volumes

Provision a volume

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

Overview

A *PersistentVolumeClaim* (PVC) is a request for access to the PersistentVolume on the cluster.

The PVC can be configured to request storage of a certain size or access mode. Using the associated StorageClass, the cluster administrator can control more than PersistentVolume size and access mode—such as performance or service level.

After you create the PVC you can mount the volume in a pod.

Create the PVC

Steps

1. Create the PVC.

```
kubectl create -f pvc.yaml
```

2. Verify the PVC status.

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS AGE
pvc-storage   Bound    pv-name    1Gi        RWO                    5m
```

1. Mount the volume in a pod.

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

You can monitor the progress using `kubectl get pod --watch`.

2. Verify that the volume is mounted on `/my/mount/path`.

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. You can now delete the Pod. The Pod application will no longer exist, but the volume will remain.

```
kubectl delete pod pv-pod
```

Sample manifests

PersistentVolumeClaim sample manifests

These examples show basic PVC configuration options.

PVC with RWO access

This example shows a basic PVC with RWO access that is associated with a StorageClass named `basic-csi`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

PVC with NVMe/TCP

This example shows a basic PVC for NVMe/TCP with RWO access that is associated with a StorageClass named `protection-gold`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Pod manifest samples

These examples show basic configurations to attach the PVC to a pod.

Basic configuration

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

Basic NVMe/TCP configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

Refer to Kubernetes and Trident objects for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Trident provisions volumes.

Expand volumes

Trident provides Kubernetes users the ability to expand their volumes after they are created. Find information about the configurations required to expand iSCSI, NFS, SMB, NVMe/TCP, and FC volumes.

Expand an iSCSI volume

You can expand an iSCSI Persistent Volume (PV) by using the CSI provisioner.

> iSCSI volume expansion is supported by the `ontap-san`, `ontap-san-economy`, `solidfire-san` drivers and requires Kubernetes 1.16 and later.

Step 1: Configure the StorageClass to support volume expansion

Edit the StorageClass definition to set the `allowVolumeExpansion` field to `true`.

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

For an already existing StorageClass, edit it to include the `allowVolumeExpansion` parameter.

Step 2: Create a PVC with the StorageClass you created

Edit the PVC definition and update the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.

```
cat pvc-ontapsan.yaml
```

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

Trident creates a Persistent Volume (PV) and associates it with this Persistent Volume Claim (PVC).

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

Step 3: Define a pod that attaches the PVC

Attach the PV to a pod for it to be resized. There are two scenarios when resizing an iSCSI PV:

- If the PV is attached to a pod, Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Trident expands the volume on the storage backend. After the PVC is bound to a pod, Trident rescans the device and resizes the filesystem. Kubernetes then updates the PVC size after the expand operation has successfully completed.

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

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

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

Step 4: Expand the PV

To resize the PV that has been created from 1Gi to 2Gi, edit the PVC definition and update the `spec.resources.requests.storage` to 2Gi.

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

Step 5: Validate the expansion

You can validate the expansion worked correctly by checking the size of the PVC, PV, and the Trident volume:

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

Expand an FC volume

You can expand an FC Persistent Volume (PV) by using the CSI provisioner.

> FC volume expansion is supported by the `ontap-san` driver and requires Kubernetes 1.16 and later.

Step 1: Configure the StorageClass to support volume expansion

Edit the StorageClass definition to set the `allowVolumeExpansion` field to `true`.

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

For an already existing StorageClass, edit it to include the `allowVolumeExpansion` parameter.

Step 2: Create a PVC with the StorageClass you created

Edit the PVC definition and update the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.

```
cat pvc-ontapsan.yaml
```

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

Trident creates a Persistent Volume (PV) and associates it with this Persistent Volume Claim (PVC).

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

Step 3: Define a pod that attaches the PVC

Attach the PV to a pod for it to be resized. There are two scenarios when resizing an FC PV:

- If the PV is attached to a pod, Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Trident expands the volume on the storage backend. After the PVC is bound to a pod, Trident rescans the device and resizes the filesystem. Kubernetes then updates the PVC size after the expand operation has successfully completed.

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

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

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

Step 4: Expand the PV

To resize the PV that has been created from 1Gi to 2Gi, edit the PVC definition and update the `spec.resources.requests.storage` to 2Gi.

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

Step 5: Validate the expansion

You can validate the expansion worked correctly by checking the size of the PVC, PV, and the Trident volume:

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

Expand an NFS volume

Trident supports volume expansion for NFS PVs provisioned on `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, and `azure-netapp-files` backends.

Step 1: Configure the StorageClass to support volume expansion

To resize an NFS PV, the admin first needs to configure the storage class to allow volume expansion by setting the `allowVolumeExpansion` field to `true`:

```
cat storageclass-ontapnas.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

If you have already created a storage class without this option, you can simply edit the existing storage class

by using `kubectl edit storageclass` to allow volume expansion.

Step 2: Create a PVC with the StorageClass you created

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident should create a 20 MiB NFS PV for this PVC:

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

Step 3: Expand the PV

To resize the newly created 20 MiB PV to 1 GiB, edit the PVC and set `spec.resources.requests.storage` to 1 GiB:

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

Step 4: Validate the expansion

You can validate the resize worked correctly by checking the size of the PVC, PV, and the Trident volume:

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

Understand RWX NVMe subsystem limits

ReadWriteMany (RWX) volumes that use the NVMe protocol have a scalability limit of 64 nodes per volume. The following includes the limitations, explains the NVMe subsystem architecture involved, and outlines the required resolution steps.

Understand the 64-node limit

If you plan to use ReadWriteMany (RWX) volumes with the NVMe protocol, a single RWX NVMe volume cannot be mounted by more than 64 nodes in a Kubernetes cluster.

Do not schedule workloads that mount the same RWX NVMe PersistentVolumeClaim across more than 64 nodes.

This limitation applies only to RWX volumes that use the NVMe protocol.

Understand NVMe subsystem models

Per-volume subsystem model (Trident releases earlier than 26.02)

In Trident releases earlier than 26.02, RWX NVMe volumes are provisioned using a per-volume subsystem

model.
Each RWX NVMe volume is mapped to its own dedicated NVMe subsystem on ONTAP.

This model is simple, but it has a lower scalability limit.
In large Kubernetes clusters, subsystem controller limits are reached quickly because each RWX volume consumes a dedicated subsystem.

Super-subsystem model (introduced in Trident 26.02)

Starting with Trident 26.02, RWX NVMe volumes use a shared super-subsystem model.
Multiple RWX NVMe volumes share the same NVMe subsystem.

Each super-subsystem supports up to 1024 namespaces (volumes).
This model significantly improves scalability for RWX workloads and reduces the likelihood of reaching ONTAP subsystem limits.

Each RWX NVMe volume supports up to 64 nodes.

Identify error symptoms

If you create or attach RWX NVMe volumes at scale, you might observe errors similar to the following:

```
Maximum number of controllers reached. No more controllers can be created.
```

This error indicates that the ONTAP NVMe subsystem controller limit has been reached.

Resolve subsystem limit errors

To move beyond per-volume subsystem limitations and take advantage of the super-subsystem model, upgrade to Trident 26.02 or later.

Upgrade Trident to apply the super-subsystem model

To apply the super-subsystem model for RWX NVMe volumes:

1. Upgrade Trident to version 26.02 or later.
2. Scale down all pods that use RWX NVMe volumes to zero replicas.
3. Verify that no workloads are actively using RWX NVMe volumes.
4. Scale the pods back up.

This restart sequence ensures that RWX NVMe volumes are attached using the super-subsystem model.

- This limitation applies only to RWX volumes that use the NVMe protocol.
- The 64-node limit applies per RWX NVMe volume.
- Other access modes and other protocols are not affected.

Controller scalability

Trident introduces controller scalability through improved concurrency across multiple storage drivers. Customers can identify which Trident drivers support controller scalability

at general availability and which drivers are available as a technical preview in Trident 26.02. This enables informed deployment decisions and appropriate risk management for scalable Kubernetes environments.

Key concepts and definitions

Controller scalability

Controller scalability refers to the Trident controller's ability to process multiple storage operations in parallel rather than serializing them behind a single lock.
These operations include volume creation, deletion, resizing, snapshot creation and deletion, volume publish and unpublish, and backend management.

When controller scalability is enabled, operations on different volumes and backends proceed concurrently. This increases throughput and reduces end-to-end operation time in environments with high numbers of concurrent PersistentVolumeClaim and VolumeSnapshot operations.

Controller scalability support

Trident supports controller scalability with different maturity levels depending on the storage driver.

General availability

The following drivers support controller scalability at general availability in Trident 26.02:

- `ontap-san`
- `ontap-nas`
- `google-cloud-netapp-volumes`

> The `google-cloud-netapp-volumes` and `google-cloud-netapp-volumes-san` drivers are different.
> Only `google-cloud-netapp-volumes` is supported. Do not use `google-cloud-netapp-volumes-san` in backend configurations or examples.

Enable controller scalability

Controller scalability is controlled by the `enableConcurrency` configuration option. This option must be explicitly enabled during Trident installation or by updating an existing deployment.

Trident operator deployment

To enable controller scalability with the Trident operator, set `enableConcurrency` to `true` in the `TridentOrchestrator` custom resource.

New installation

Create or edit the `TridentOrchestrator` CR with `enableConcurrency` set to `true`:

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

Apply the CR:

```
kubectl apply -f tridentorchestrator_cr.yaml
```

Existing installation

Patch the existing `TridentOrchestrator` CR to enable controller scalability:

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

Verify the setting was applied:

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm deployment

To enable controller scalability with Helm, set the `enableConcurrency` value to `true`.

New installation

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

Existing installation

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

Alternatively, set `enableConcurrency` to `true` in a custom `values.yaml` file:

```
# values.yaml
enableConcurrency: true
```

Then install or upgrade using the values file:

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl deployment

To enable controller scalability with `tridentctl`, pass the `--enable-concurrency` flag during installation.

New installation

```
tridentctl install -n trident --enable-concurrency
```

Existing installation

To enable controller scalability on an existing `tridentctl`-based deployment, uninstall and reinstall with the flag:

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

Verify controller scalability is enabled

After enabling controller scalability, verify that the Trident controller is running with concurrency enabled by checking the controller pod logs:

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

You should see a log entry indicating that concurrency is enabled.

Technical preview

The following drivers support controller scalability as a technical preview in Trident 26.02:

- `nas-eco`
- `san-eco`

For these drivers:

- Controller concurrency is available for evaluation and testing

- Behavior may change in future releases
- Use in production environments is not recommended

Concurrency behavior

When controller scalability is enabled:

- Trident replaces the single global lock with fine-grained, per-resource locking
- Operations that modify the same resource are serialized to maintain data consistency
- Operations that only read from a resource can proceed concurrently with other read operations on that resource
- Trident limits concurrent ONTAP API requests to 20 per management LIF to prevent overloading backend storage systems
- If multiple backends share the same management LIF, they share this 20-request limit

Known limitations and considerations

The following considerations apply to controller scalability:

- Concurrency is managed internally by the Trident controller
- There are no user-configurable concurrency limits in this release
- Overall throughput depends on:
  - The storage driver in use
  - Backend responsiveness
  - Kubernetes API server performance
- High concurrency can increase load on backend storage systems

Caveats and limitations

The following limitations apply in Trident 26.02:

- Controller scalability behavior is not identical across all drivers
- Technical preview drivers may exhibit:
  - Inconsistent performance under high load
  - Changes in behavior between releases
- Debugging concurrent operations may be more complex due to parallel execution
- Metrics and logs may show interleaved operation output

Recommendations

- Use general availability (GA) drivers for production environments that require high scalability
- Evaluate technical preview drivers in non-production environments
- Monitor backend and controller performance when operating at scale
- Avoid assuming operation ordering in automation scripts

Automatic volume expansion

Automatic volume expansion enables Persistent Volumes provisioned by Trident to grow automatically when used capacity reaches a defined threshold. This capability reduces operational overhead and helps prevent application disruption caused by capacity exhaustion. Automatic volume expansion is implemented using Autogrow Policies. An Autogrow Policy defines:

- The utilization threshold that triggers expansion
- The amount by which the volume grows
- The maximum size the volume can reach

> Volumes increase in size automatically when the defined utilization threshold is exceeded. Volumes are never automatically reduced.

Requirements

Before configuring automatic volume expansion, ensure that the following requirements are met:

- Trident 26.02 or later
- Role-based access control permissions to create `TridentAutogrowPolicy` custom resources
- StorageClasses configured with `allowVolumeExpansion: true`
- Supported ONTAP protocols:
  - Network File System (NFS)
  - Internet Small Computer Systems Interface (iSCSI)
  - Non-Volatile Memory Express (NVMe)
  - Fibre Channel Protocol (FCP)

Limitations

- ONTAP Non-Volatile Memory Express raw block volumes earlier than ONTAP 9.16.1 do not support automatic expansion.
- For storage area network volumes, if `growthAmount` is less than or equal to 50 mebibytes, Trident automatically increases the value to 51 mebibytes before resizing, provided the resulting size does not exceed `maxSize`.
- In brownfield environments, automatic expansion might not function for certain existing volumes due to volume publication migration behavior.
- When a volume reaches `maxSize`, no further expansion occurs.
- Supported protocols for automatic volume expansion:
  - Network File System (NFS)
  - Internet Small Computer Systems Interface (iSCSI)
  - Non-Volatile Memory Express (NVMe)
  - Fibre Channel Protocol (FCP)

Provision Volumes with Autogrow Policy

Autogrow Policy can be configured at two levels:

- Storage class level: Sets default for all volumes (using annotation)
- PVC level: Overrides storage class default (using annotation)

Create an Autogrow Policy

Autogrow Policies enable automatic volume expansion when volumes reach a defined capacity threshold.

Ensure you have:

- Trident 26.02 or later installed
- Role-based access control permissions to create `TridentAutogrowPolicy` resources
- Understanding of workload growth requirements

An Autogrow Policy defines how volumes expand automatically when they reach a defined capacity threshold.

You can create Autogrow Policies at any point in your workflow:

- Before StorageClasses and volumes are created
- After StorageClasses exist
- After volumes are provisioned

This flexibility allows you to introduce automatic expansion without recreating existing resources.

Autogrow Policy specifications

Autogrow Policies are Kubernetes custom resources defined as follows:

Field	Description	Format	Required	Example	Default
name	Unique policy identifier	String	Yes	production-db-policy	None
usedThreshold	Capacity percentage that triggers expansion	Percentage string	Yes	"80%"	None
growthAmount	Amount to grow when threshold is reached	Percentage or size	No	"10%" or "5Gi"	"10%"
maxSize	Maximum volume size limit	Kubernetes quantity	No	"500Gi"	Unlimited

Create an Autogrow Policy

Steps

1. Create a YAML file that defines your Autogrow Policy:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. Apply the policy to your cluster:

```
kubectl apply -f autogrow-policy.yaml
```

3. Verify that the policy was created:

```
kubectl get tridentautogrowpolicy standard-autogrow
```

Expected output

```
NAME                USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow   80%              10%             Success
```

Policy states

After you create a policy, Trident validates the specification and assigns one of the following states:

State	Description	Action required
Success	Policy is validated and ready to use.	None.
Failed	Validation errors detected.	Review and fix the specification.
Deleting	Deletion is in progress.	Wait for completion.

Associate a policy with a StorageClass

You can associate an Autogrow Policy with a StorageClass by using the `trident.netapp.io/autogrowPolicy` annotation. All volumes provisioned from that StorageClass inherit the policy.

The StorageClass must have `allowVolumeExpansion: true`.

Steps

1. Create or modify a StorageClass with the Autogrow Policy annotation:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. Apply the StorageClass:

```
kubectl apply -f storageclass.yaml
```

3. Verify the annotation:

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Expected output

```
production-db-policy
```

Policy precedence

When Autogrow Policy annotations are set on both a StorageClass and a PVC, Trident applies the following precedence rules:

1. **PVC annotation takes priority.** If a PVC sets `trident.netapp.io/autogrowPolicy`, that value is always used.
2. **StorageClass annotation applies only when the PVC has no annotation.**
3. **If neither has the annotation,** no Autogrow Policy is applied.

Table 1. Precedence examples

StorageClass annotation	PVC annotation	Effective behavior
`trident.netapp.io/autogrowPolicy: standard-agp`	Not set	Uses `standard-agp`.

StorageClass annotation	PVC annotation	Effective behavior
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	Uses `logs-policy` (PVC overrides StorageClass).
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	No Autogrow Policy (PVC disables autogrow).
Not set	`trident.netapp.io/autogrowPolicy: dev-policy`	Uses `dev-policy`.
Not set	Not set	No Autogrow Policy.

Configuration examples

The following examples show common Autogrow Policy configurations for different use cases.

Conservative policy for production databases

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

Log storage with fixed growth increments

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

Minimal policy with defaults

When you omit `growthAmount` and `maxSize`, Trident uses the defaults (`10%` growth, unlimited size):

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

Policy with a custom maxSize and default growthAmount

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

Aggressive growth with unlimited maxSize

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

Policy with fractional percentages

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

Fractional percentages are supported. If you specify more than three decimal places, Trident rounds the value to three decimal places.

NAS StorageClass with Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass with Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

Manage Autogrow Policies

After you create Autogrow Policies, you can view, update, and delete them as needed. You can also monitor which volumes are using a given policy.

View Autogrow Policies

List all policies

Use `kubectl` to list all Autogrow Policies in your cluster:

```
kubectl get tridentautogrowpolicy
```

Alternatively, use `tridentctl`:

```
tridentctl get autogrowpolicy
```

View policy details

To view the full specification and status of a policy:

```
kubectl describe tridentautogrowpolicy production-db-policy
```

To view a policy with its associated volumes in YAML format:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

Update an Autogrow Policy

You can modify an existing policy to change its threshold, growth amount, or maximum size. Changes take effect immediately for all volumes that use the policy.

> Changes affect all volumes currently using the policy. Test changes in a non-production environment first when possible.

Steps

1. Edit the policy:

   ```
   kubectl edit tridentautogrowpolicy production-db-policy
   ```

2. Modify the `spec` fields as needed:

   ```
   spec:
     usedThreshold: "75%"     # Changed from 80%
     growthAmount: "20%"      # Changed from 10%
     maxSize: "1Ti"           # Changed from 500Gi
   ```

3. Save and exit. The changes take effect immediately.

Update considerations

- **Immediate effect:** All volumes using the policy adopt new parameters at the next growth evaluation.
- **No volume restart needed:** Changes apply to the next growth operation.
- **Test first:** Validate changes in a non-production environment when possible.
- **Communicate changes:** Notify teams when you modify shared policies.

Delete an Autogrow Policy

Autogrow Policies use finalizer protection to prevent accidental deletion while volumes are actively using them.

Steps

1. Delete the policy:

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. If volumes are still using the policy, the deletion enters a `Deleting` state. Check which volumes are affected:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. Remove the policy from each affected volume. Choose one of the following options:

   - **Option A: Explicitly disable autogrow** by setting the annotation to `"none"`:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

   - **Option B: Remove the annotation entirely:**

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

Deletion behavior

Scenario	Behavior
No volumes use the policy	Policy is deleted immediately.
Volumes are using the policy	Policy enters `Deleting` state. A finalizer blocks completion until all volumes are removed.
All volumes are removed from the policy	Finalizers are removed and the policy is deleted.

Monitor Autogrow Policy usage

Check volumes using a policy

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

Find which policy a volume uses

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Monitor policy events

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

Supported protocols

Autogrow supports the following storage protocols:

- NFS
- iSCSI
- FCP
- NVMe

> For SAN volumes, if the configured `growthAmount` is 50 MiB or less, Trident automatically increases the growth amount to 51 MB for the resize operation, as long as the resulting size does not exceed `maxSize`.

Known limitations

- **ONTAP NVMe raw block volumes:** Volumes created with ONTAP versions earlier than 9.16.1 do not support autogrow.
- **Existing volumes (brownfield deployments):** Autogrow might not work for existing volumes even if a valid Autogrow Policy is applied. This is due to an ongoing migration of volume publications. To confirm migration has completed, check the Trident controller logs for "`Migration completed`" messages.

Frequently asked questions

When does Trident evaluate the threshold?

Trident continuously monitors volume usage. When the used capacity crosses the `usedThreshold`, Trident creates an internal resize request and expands the volume by the configured `growthAmount`.

For example, this policy triggers expansion at 80% capacity and grows the volume by 10% each time, up to a maximum of 500 GiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

Can I apply a policy after volumes are already provisioned?

Yes. You can create an Autogrow Policy at any time and apply it to existing PVCs by adding or updating the `trident.netapp.io/autogrowPolicy` annotation. You do not need to recreate the PVC or the StorageClass.

To apply a policy to an existing PVC:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

To apply a policy to an existing StorageClass:

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

What happens if I set an Autogrow Policy on both the StorageClass and the PVC?

The PVC annotation always takes precedence. If a PVC has the `trident.netapp.io/autogrowPolicy` annotation, Trident uses that value regardless of what the StorageClass specifies. Refer to Policy precedence for details.

For example, given this StorageClass:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

And this PVC that overrides the StorageClass policy:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Trident uses `logs-policy` for `database-pvc`, not `standard-agp`.

How do I disable autogrow for a specific volume?

Set the PVC annotation to `"none"`. This overrides any StorageClass-level policy for that volume:

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

You can verify that autogrow is disabled:

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

Expected output

```
none
```

What happens when a volume reaches maxSize?

Trident stops expanding the volume. No further resize requests are created for that volume, even if usage continues to increase beyond the `usedThreshold`.

For example, with this policy, Trident stops growing the volume once it reaches 100 GiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

To allow unlimited growth, omit `maxSize` or set it to `0`:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

Can I change a policy without restarting volumes?

Yes. When you update a policy, all volumes using that policy adopt the new parameters at the next growth evaluation. No volume restarts are required.

To update a policy in place:

```
kubectl edit tridentautogrowpolicy production-db-policy
```

Modify the fields as needed:

```
spec:
  usedThreshold: "75%"    # Changed from 80%
  growthAmount: "20%"     # Changed from 10%
  maxSize: "1Ti"          # Changed from 500Gi
```

Save and exit. Verify the updated policy:

```
kubectl get tridentautogrowpolicy production-db-policy
```

Expected output

```
NAME                    USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy    75%              20%             Success
```

Why is my policy in a Failed state?

A `Failed` state indicates that the policy specification contains validation errors. Run the following command to view the error details:

```
kubectl describe tridentautogrowpolicy <policy-name>
```

Common causes include an invalid `usedThreshold` (must be 1–99%), a `growthAmount` that exceeds `maxSize`, or an invalid Kubernetes quantity format. Correct the specification and reapply:

```
kubectl apply -f autogrow-policy.yaml
```

Why can't I delete a policy?

Policies use finalizer protection. If volumes are still using the policy, deletion enters a `Deleting` state and waits until all volumes are removed from the policy.

Identify the affected volumes:

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

Then remove the annotation from each PVC:

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

After all volumes are removed, the finalizer is released and the policy is deleted.

Does autogrow work with all ONTAP backends?

Autogrow supports NFS, iSCSI, FCP, and NVMe protocols. However, NVMe raw block volumes require ONTAP 9.16.1 or later.

Existing volumes in brownfield deployments might require volume publication migration to complete before

autogrow takes effect. Verify migration status by checking the Trident controller logs:

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
completed"
```

The following StorageClass examples show autogrow configured for NAS and SAN backends:

NAS backend

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN backend

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

What is the minimum growth amount for SAN volumes?

For SAN volumes, the effective minimum growth amount is 51 MB. If you configure a `growthAmount` of 50 MiB or less, Trident automatically increases the growth to 51 MB for the resize operation.

For example, this policy sets a `growthAmount` of `"40Mi"`, but Trident applies a 51 MB growth for any SAN volume that uses it:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

To avoid this automatic adjustment, set `growthAmount` to a value greater than 50 MiB:

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

Import volumes

You can import existing storage volumes as a Kubernetes PV using `tridentctl import` or by creating a Persistent Volume Claim (PVC) with Trident import annotations.

Overview and considerations

You might import a volume into Trident to:

- Containerize an application and reuse its existing data set
- Use a clone of a data set for an ephemeral application
- Rebuild a failed Kubernetes cluster
- Migrate application data during disaster recovery

Considerations

Before importing a volume, review the following considerations.

- Trident can import RW (read-write) type ONTAP volumes only. DP (data protection) type volumes are SnapMirror destination volumes. You should break the mirror relationship before importing the volume into Trident.
- We suggest importing volumes without active connections. To import an actively-used volume, clone the volume and then perform the import.

This is especially important for block volumes as Kubernetes would be unaware of the previous connection and could easily attach an active volume to a pod. This can result in data corruption.

- Though `StorageClass` must be specified on a PVC, Trident does not use this parameter during import. Storage classes are used during volume creation to select from available pools based on storage characteristics. Because the volume already exists, no pool selection is required during import. Therefore, the import will not fail even if the volume exists on a backend or pool that does not match the storage class specified in the PVC.
- The existing volume size is determined and set in the PVC. After the volume is imported by the storage driver, the PV is created with a ClaimRef to the PVC.
  - The reclaim policy is initially set to `retain` in the PV. After Kubernetes successfully binds the PVC and PV, the reclaim policy is updated to match the reclaim policy of the Storage Class.
  - If the reclaim policy of the Storage Class is `delete`, the storage volume will be deleted when the PV is deleted.
- By default, Trident manages the PVC and renames the FlexVol volume and LUN on the backend. You can pass the `--no-manage` flag to import an unmanaged volume and the `--no-rename` flag to retain the volume name.
  - **`--no-manage`** - If you use the `--no-manage` flag, Trident does not perform any additional operations on the PVC or PV for the lifecycle of the objects. The storage volume is not deleted when the PV is deleted and other operations such as volume clone and volume resize are also ignored.
  - **`--no-rename`** - If you use the `--no-rename` flag, Trident retains the existing volume name while importing volumes, and manages the lifecycle of the volumes. This option is supported only for the `ontap-nas`, `ontap-san` (including ASA r2 systems), and `ontap-san-economy` drivers.

    These options are useful if you want to use Kubernetes for containerized workloads but otherwise want to manage the lifecycle of the storage volume outside of Kubernetes.
- An annotation is added to the PVC and PV that serves a dual purpose of indicating that the volume was imported and if the PVC and PV are managed. This annotation should not be modified or removed.

Import a volume

You can import a volume using either `tridentctl import` or by creating a PVC with Trident import annotations.

If you use PVC annotations, you don't need to download or use `tridentctl` to import the volume.

Using tridentctl

Steps

1. Create a PVC file (for example, `pvc.yaml`) that will be used to create the PVC. The PVC file should include `name`, `namespace`, `accessModes`, and `storageClassName`. Optionally, you can specify `unixPermissions` in your PVC definition.

   The following is an example of a minimum specification:

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

   > Include only the required parameters. Additional parameters such as PV name or volume size can cause the import command to fail.

2. Use the `tridentctl import` command to specify the name of the Trident backend containing the volume and the name that uniquely identifies the volume on the storage (for example: ONTAP FlexVol, Element Volume). The `-f` argument is required to specify the path to the PVC file.

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

Using PVC annotations

Steps

1. Create a PVC YAML file (for example, `pvc.yaml`) with the required Trident import annotations. The PVC file should include:
   - `name` and `namespace` in metadata
   - `accessModes`, `resources.requests.storage`, and `storageClassName` in spec
   - Annotations:
     - `trident.netapp.io/importOriginalName`: Volume name on the backend
     - `trident.netapp.io/importBackendUUID`: Backend UUID where volume exists
     - `trident.netapp.io/notManaged` (*Optional*): Set to `"true"` for unmanaged volumes. Default is `"false"`.

       The following is an example specification for importing a managed volume:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. Apply the PVC YAML file to your Kubernetes cluster:

```
kubectl apply -f <pvc-file>.yaml
```

Trident will automatically import the volume and bind it to the PVC.

Examples

Review the following volume import examples for supported drivers.

ONTAP NAS and ONTAP NAS FlexGroup

Trident supports volume import using the `ontap-nas` and `ontap-nas-flexgroup` drivers.

- Trident does not support volume import using the `ontap-nas-economy` driver.
- The `ontap-nas` and `ontap-nas-flexgroup` drivers do not allow duplicate volume names.

Each volume created with the `ontap-nas` driver is a FlexVol volume on the ONTAP cluster. Importing FlexVol volumes with the `ontap-nas` driver works the same. A FlexVol volumes that already exists on an ONTAP cluster can be imported as a `ontap-nas` PVC. Similarly, FlexGroup vols can be imported as `ontap-nas-flexgroup` PVCs.

ONTAP NAS examples using tridentctl

The following examples show how to import managed and unmanaged volumes using `tridentctl`.

Managed volume

The following example imports a volume named `managed_volume` on a backend named `ontap_nas`:

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
```

Unmanaged volume

When using the `--no-manage` argument, Trident does not rename the volume.

The following example imports `unmanaged_volume` on the `ontap_nas` backend:

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

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

ONTAP NAS examples using PVC annotations

The following examples show how to import managed and unmanaged volumes using PVC annotations.

Managed volume

The following example imports a 1GiB `ontap-nas` volume named `ontap_volume1` from backend `81abcb27-ea63-49bb-b606-0a5315ac5f21` with RWO access mode set using PVC annotations:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

Unmanaged volume

The following example imports 1Gi `ontap-nas` volume named `ontap-volume2` from backend `34abcb27-ea63-49bb-b606-0a5315ac5f34` with RWO access mode set using PVC annotations:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Trident supports volume import using the `ontap-san` (iSCSI, NVMe/TCP, and FC) and `ontap-san-economy` drivers.

Trident can import ONTAP SAN FlexVol volumes that contain a single LUN. This is consistent with the `ontap-san` driver, which creates a FlexVol volume for each PVC and a LUN within the FlexVol volume. Trident imports the FlexVol volume and associates it with the PVC definition. Trident can import `ontap-san-economy` volumes that contain multiple LUNs.

The following examples show how to import managed and unmanaged volumes:

Managed volume

For managed volumes, Trident renames the FlexVol volume to the `pvc-<uuid>` format and the LUN within the FlexVol volume to `lun0`.

The following example imports the `ontap-san-managed` FlexVol volume that is present on the `ontap_san_default` backend:

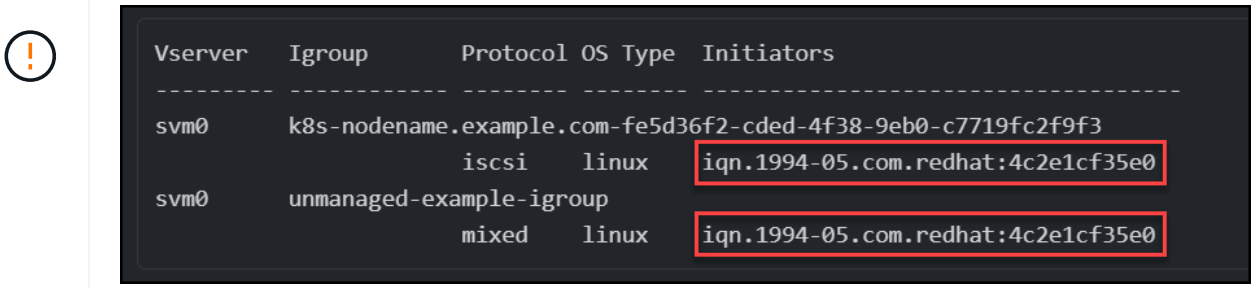
```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
```

Unmanaged volume

The following example imports `unmanaged_example_volume` on the `ontap_san` backend:

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
```

If you have LUNS mapped to igroups that share an IQN with a Kubernetes node IQN, as shown in the following example, you will receive the error: `LUN already mapped to initiator(s) in this group`. You will need to remove the initiator or unmap the LUN to import the volume.



```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economy examples

The following examples show how to import managed and unmanaged volumes for the `ontap-san-economy` backend.

Managed volume

When you import a managed volume, Trident takes ownership of the FlexVol and renames it. You must account for this renaming when you import multiple LUNs from the same FlexVol.

The following example imports `lun1` from the FlexVol `toimport` as a managed volume named `vol-managed-saneco`:

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

After importing `lun1`, Trident renames the FlexVol (for example, to `trident_lun_pool_xyz`). To import additional LUNs from the same FlexVol, use the new FlexVol name:

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

> The `ontap-san-economy` backend imports one LUN at a time. You can automate multiple imports using a script.

Unmanaged volume

When you import an unmanaged volume, Trident does not take ownership of the FlexVol. However, the FlexVol and LUN must follow Trident naming conventions.

FlexVol naming format

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX` is the value of `storagePrefix` in your backend configuration. The default is `trident`.
- `RANDOMSTRING` is any string you choose.

LUN naming requirement

The LUN must be named `lun0`.

Example

If your `storagePrefix` is `xyz`, the full path to the LUN is:

```
trident_lun_pool_xyz_randomstring/lun0
```

Element

Trident supports NetApp Element software and NetApp HCI volume import using the `solidfire-san` driver.

The Element driver supports duplicate volume names. However, Trident returns an error if there are duplicate volume names. As a workaround, clone the volume, provide a unique volume name, and import the cloned volume.

The following example imports an `element-managed` volume on backend `element_default`.

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident supports volume import using the `azure-netapp-files` driver.

To import an Azure NetApp Files volume, identify the volume by its volume path. The volume path is the portion of the volume's export path after the `:/`. For example, if the mount path is `10.0.0.2:/importvol1`, the volume path is `importvol1`.

The following example imports an `azure-netapp-files` volume on backend `azurenetappfiles_40517` with the volume path `importvol1`.

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

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

Google Cloud NetApp Volumes

Trident supports volume import using the `google-cloud-netapp-volumes` driver.

The following example imports a volume on backend `backend-tbc-gcnv1` with the volume `testvoleasiaeast1`.

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

The following example imports a `google-cloud-netapp-volumes` volume when two volumes are present in the same region:

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

Customize volume names and labels

With Trident, you can assign meaningful names and labels to volumes you create. This helps you identify and easily map volumes to their respective Kubernetes resources (PVCs). You can also define templates at the backend level for creating custom volume names and custom labels; any volumes that you create, import, or clone will adhere to the templates.

Before you begin

Customizable volume names and labels support:

- Volume create, import, and clone operations.
- In the case of the `ontap-nas-economy` driver, only the name of the Qtree volume complies with the name template.
- In the case of the `ontap-san-economy` driver, only the LUN name complies with the name template.

Limitations

- Custom volume names are compatible with ONTAP on-premises drivers only.
- Custom labels are supported only for the `ontap-san`, `ontap-nas`, and `ontap-nas-flexgroup` drivers.
- Custom volume names do not apply to existing volumes.

Key behaviors of customizable volume names

- If a failure occurs due to invalid syntax in a name template, the backend creation fails. However, if the template application fails, the volume will be named according to existing naming convention.

- Storage prefix is not applicable when a volume is named using a name template from the backend configuration. Any desired prefix value may be directly added to the template.

Backend configuration examples with name template and labels

Custom name templates can be defined at the root and/or pool level.

Root level example

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

Pool level example

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

Name template examples

Example 1:

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

Example 2:

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

Points to consider

1. In the case of volume imports, the labels are updated only if the existing volume has labels in a specific format. For example: `{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`.
2. In the case of managed volume imports, the volume name follows the name template defined at the root level in the backend definition.
3. Trident does not support the use of a slice operator with the storage prefix.
4. If the templates do not result in unique volume names, Trident will append a few random characters to create unique volume names.
5. If the custom name for a NAS economy volume exceeds 64 characters in length, Trident will name the volumes according to the existing naming convention. For all other ONTAP drivers, if the volume name exceeds the name limit, the volume creation process fails.

Share an NFS volume across namespaces

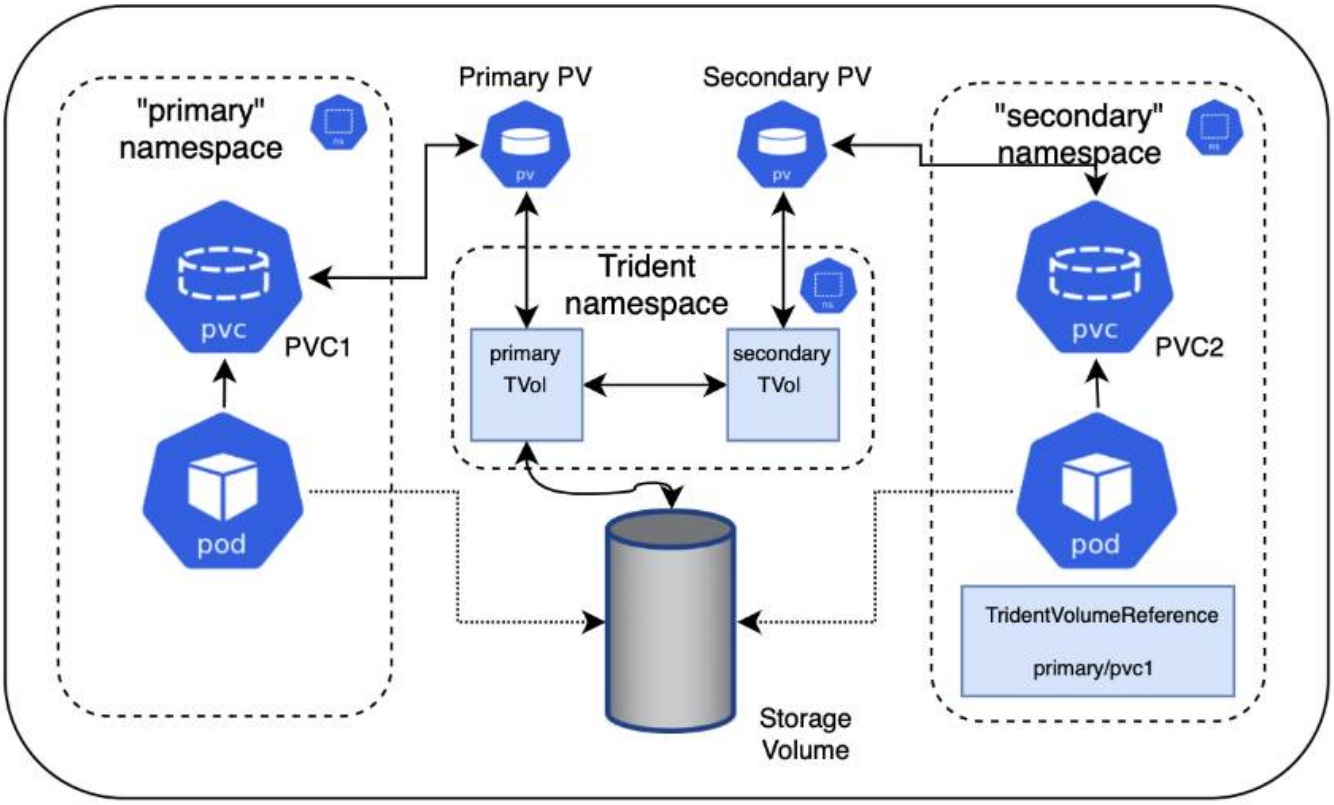
Using Trident, you can create a volume in a primary namespace and share it in one or more secondary namespaces.

Features

The TridentVolumeReference CR allows you to securely share ReadWriteMany (RWX) NFS volumes across one or more Kubernetes namespaces. This Kubernetes-native solution has the following benefits:

- Multiple levels of access control to ensure security
- Works with all Trident NFS volume drivers
- No reliance on tridentctl or any other non-native Kubernetes feature

This diagram illustrates NFS volume sharing across two Kubernetes namespaces.



Quick start

You can set up NFS volume sharing in just a few steps.

1 **Configure source PVC to share the volume**

The source namespace owner grants permission to access the data in the source PVC.

2 **Grant permission to create a CR in the destination namespace**

The cluster administrator grants permission to the owner of the destination namespace to create the TridentVolumeReference CR.

3 **Create TridentVolumeReference in the destination namespace**

The owner of the destination namespace creates the TridentVolumeReference CR to refer to the source PVC.

4 **Create the subordinate PVC in the destination namespace**

The owner of the destination namespace creates the subordinate PVC to use the data source from the source PVC.

Configure the source and destination namespaces

To ensure security, cross namespace sharing requires collaboration and action by the source namespace

owner, cluster administrator, and destination namespace owner. The user role is designated in each step.

Steps

1. **Source namespace owner:** Create the PVC (`pvc1`) in the source namespace that grants permission to share with the destination namespace (`namespace2`) using the `shareToNamespace` annotation.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident creates the PV and its backend NFS storage volume.

- You can share the PVC to multiple namespaces using a comma-delimited list. For example, `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`.
- You can share to all namespaces using `*`. For example, `trident.netapp.io/shareToNamespace: *`
- You can update the PVC to include the `shareToNamespace` annotation at any time.

2. **Cluster admin:** Ensure that proper RBAC is in place to grant permission to the destination namespace owner to create the TridentVolumeReference CR in the destination namespace.
3. **Destination namespace owner:** Create a TridentVolumeReference CR in the destination namespace that refers to the source namespace `pvc1`.

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination namespace owner:** Create a PVC (`pvc2`) in destination namespace (`namespace2`) using the `shareFromPVC` annotation to designate the source PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

The size of the destination PVC must be less than or equal than the source PVC.

Results

Trident reads the `shareFromPVC` annotation on the destination PVC and creates the destination PV as a subordinate volume with no storage resource of its own that points to the source PV and shares the source PV storage resource. The destination PVC and PV appear bound as normal.

Delete a shared volume

You can delete a volume that is shared across multiple namespaces. Trident will remove access to the volume on the source namespace and maintain access for other namespaces that share the volume. When all namespaces that reference the volume are removed, Trident deletes the volume.

Use `tridentctl get` to query subordinate volumes

Using the `tridentctl` utility, you can run the `get` command to get subordinate volumes. For more information, refer to `tridentctl` commands and options.

```
Usage:
  tridentctl get [option]
```

Flags:

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

Limitations

- Trident cannot prevent destination namespaces from writing to the shared volume. You should use file locking or other processes to prevent overwriting shared volume data.
- You cannot revoke access to the source PVC by removing the `shareToNamespace` or `shareFromNamespace` annotations or deleting the `TridentVolumeReference` CR. To revoke access, you must delete the subordinate PVC.
- Snapshots, clones, and mirroring are not possible on subordinate volumes.

For more information

To learn more about cross-namespace volume access:

- Watch the demo on NetAppTV.

Clone volumes across namespaces

Using Trident, you can create new volumes using existing volumes or volumesnapshots from a different namespace inside the same Kubernetes cluster.

Prerequisites

Before cloning volumes, ensure that the source and destination backends are of the same type and have the same storage class.

> Cloning across namespaces is supported only for the `ontap-san` and `ontap-nas` storage drivers. Read-only clones are not supported.

Quick start

You can set up volume cloning in just a few steps.

Configure source PVC to clone the volume

The source namespace owner grants permission to access the data in the source PVC.

Grant permission to create a CR in the destination namespace

The cluster administrator grants permission to the owner of the destination namespace to create the TridentVolumeReference CR.

Create TridentVolumeReference in the destination namespace

The owner of the destination namespace creates the TridentVolumeReference CR to refer to the source PVC.

Create the clone PVC in the destination namespace

The owner of the destination namespace creates PVC to clone the PVC from the source namespace.

Configure the source and destination namespaces

To ensure security, cloning volumes across namespaces requires collaboration and action by the source namespace owner, cluster administrator, and destination namespace owner. The user role is designated in each step.

Steps

1. **Source namespace owner:** Create the PVC (`pvc1`) in the source namespace (`namespace1`) that grants permission to share with the destination namespace (`namespace2`) using the `cloneToNamespace` annotation.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

   Trident creates the PV and its backend storage volume.

   - You can share the PVC to multiple namespaces using a comma-delimited list. For example, `trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`.
   - You can share to all namespaces using `*`. For example, `trident.netapp.io/cloneToNamespace: *`
   - You can update the PVC to include the `cloneToNamespace` annotation at any time.

2. **Cluster admin:** Ensure that proper RBAC is in place to grant permission to the destination namespace owner to create the TridentVolumeReference CR in the destination namespace (`namespace2`).
3. **Destination namespace owner:** Create a TridentVolumeReference CR in the destination namespace that refers to the source namespace `pvc1`.

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination namespace owner:** Create a PVC (`pvc2`) in destination namespace (`namespace2`) using the `cloneFromPVC` or `cloneFromSnapshot`, and `cloneFromNamespace` annotations to designate the source PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Limitations

- For PVCs provisioned using ontap-nas-economy drivers, read-only clones are not supported.

Replicate volumes using SnapMirror

Trident supports mirror relationships between a source volume on one cluster and the destination volume on the peered cluster for replicating data for disaster recovery. You can use a namespaced Custom Resource Definition (CRD), called Trident Mirror Relationship (TMR) to perform the following operations:

- Create mirror relationships between volumes (PVCs)
- Remove mirror relationships between volumes
- Break the mirror relationships
- Promote the secondary volume during disaster conditions (failovers)

- Perform lossless transition of applications from cluster to cluster (during planned failovers or migrations)

Replication prerequisites

Ensure that the following prerequisites are met before you begin:

ONTAP clusters

- **Trident**: Trident version 22.10 or later must exist on both the source and destination Kubernetes clusters that utilize ONTAP as a backend.
- **Licenses**: ONTAP SnapMirror asynchronous licenses using the Data Protection bundle must be enabled on both the source and destination ONTAP clusters. Refer to SnapMirror licensing overview in ONTAP for more information.

  Beginning with ONTAP 9.10.1, all licenses are delivered as a NetApp license file (NLF), which is a single file that enables multiple features. Refer to Licenses included with ONTAP One for more information.

  > Only SnapMirror asynchronous protection is supported.

Peering

- **Cluster and SVM**: The ONTAP storage backends must be peered. Refer to Cluster and SVM peering overview for more information.

  > Ensure that the SVM names used in the replication relationship between two ONTAP clusters are unique.

- **Trident and SVM**: The peered remote SVMs must be available to Trident on the destination cluster.

Supported drivers

NetApp Trident supports volume replication with NetApp SnapMirror technology using storage classes backed by the following drivers:
`ontap-nas`: NFS
`ontap-san`: iSCSI
`ontap-san`: FC
`ontap-san`: NVMe/TCP (requires minimum ONTAP version 9.15.1)

> Volume replication using SnapMirror is not supported for ASA r2 systems. For information about ASA r2 systems, see Learn about ASA r2 storage systems.

Create a mirrored PVC

Follow these steps and use the CRD examples to create mirror relationship between primary and secondary volumes.

Steps

1. Perform the following steps on the primary Kubernetes cluster:
   a. Create a StorageClass object with the `trident.netapp.io/replication: true` parameter.

Example

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "nfs"
  trident.netapp.io/replication: "true"
```

b. Create a PVC with previously created StorageClass.

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. Create a MirrorRelationship CR with local information.

Example

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident fetches the internal information for the volume and the volume's current data protection (DP) state, then populates the status field of the MirrorRelationship.

d. Get the TridentMirrorRelationship CR to obtain the internal name and SVM of the PVC.

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. Perform the following steps on the secondary Kubernetes cluster:
   a. Create a StorageClass with the trident.netapp.io/replication: true parameter.

      Example

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        trident.netapp.io/replication: true
      ```

   b. Create a MirrorRelationship CR with destination and source information.

Example

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Trident will create a SnapMirror relationship with the configured relationship policy name (or default for ONTAP) and initialize it.

c. Create a PVC with previously created StorageClass to act as the secondary (SnapMirror destination).

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Trident will check for the TridentMirrorRelationship CRD and fail to create the volume if the relationship does not exist. If the relationship exists, Trident will ensure the new FlexVol volume is placed onto an SVM that is peered with the remote SVM defined in the MirrorRelationship.

Volume Replication States

A Trident Mirror Relationship (TMR) is a CRD that represents one end of a replication relationship between PVCs. The destination TMR has a state, which tells Trident what the desired state is. The destination TMR has the following states:

- **Established**: the local PVC is the destination volume of a mirror relationship, and this is a new relationship.
- **Promoted**: the local PVC is ReadWrite and mountable, with no mirror relationship currently in effect.
- **Reestablished**: the local PVC is the destination volume of a mirror relationship and was also previously in

that mirror relationship.

- The reestablished state must be used if the destination volume was ever in a relationship with the source volume because it overwrites the destination volume contents.
- The reestablished state will fail if the volume was not previously in a relationship with the source.

Promote secondary PVC during an unplanned failover

Perform the following step on the secondary Kubernetes cluster:

- Update the *spec.state* field of TridentMirrorRelationship to `promoted`.

Promote secondary PVC during a planned failover

During a planned failover (migration), perform the following steps to promote the secondary PVC:

Steps

1. On the primary Kubernetes cluster, create a snapshot of the PVC and wait until the snapshot is created.
2. On the primary Kubernetes cluster, create the SnapshotInfo CR to obtain internal details.

   Example

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. On secondary Kubernetes cluster, update the *spec.state* field of the *TridentMirrorRelationship* CR to *promoted* and *spec.promotedSnapshotHandle* to be the internalName of the snapshot.
4. On secondary Kubernetes cluster, confirm the status (status.state field) of TridentMirrorRelationship to promoted.

Restore a mirror relationship after a failover

Before restoring a mirror relationship, choose the side that you want to make as the new primary.

Steps

1. On the secondary Kubernetes cluster, ensure that the values for the *spec.remoteVolumeHandle* field on the TridentMirrorRelationship is updated.
2. On secondary Kubernetes cluster, update the *spec.mirror* field of TridentMirrorRelationship to `reestablished`.

Additional operations

Trident supports the following operations on the primary and secondary volumes:

Replicate primary PVC to a new secondary PVC

Ensure that you already have a primary PVC and a secondary PVC.

Steps

1. Delete the PersistentVolumeClaim and TridentMirrorRelationship CRDs from the established secondary (destination) cluster.
2. Delete the TridentMirrorRelationship CRD from the primary (source) cluster.
3. Create a new TridentMirrorRelationship CRD on the primary (source) cluster for the new secondary (destination) PVC you want to establish.

Resize a mirrored, primary or secondary PVC

The PVC can be resized as normal, ONTAP will automatically expand any destination flevxols if the amount of data exceeds the current size.

Remove replication from a PVC

To remove replication, perform one of the following operations on the current secondary volume:

- Delete the MirrorRelationship on the secondary PVC. This breaks the replication relationship.
- Or, update the spec.state field to *promoted*.

Delete a PVC (that was previously mirrored)

Trident checks for replicated PVCs, and releases the replication relationship before attempting to delete the volume.

Delete a TMR

Deleting a TMR on one side of a mirrored relationship causes the remaining TMR to transition to *promoted* state before Trident completes the deletion. If the TMR selected for deletion is already in *promoted* state, there is no existing mirror relationship and the TMR will be removed and Trident will promote the local PVC to *ReadWrite*. This deletion releases SnapMirror metadata for the local volume in ONTAP. If this volume is used in a mirror relationship in the future, it must use a new TMR with an *established* volume replication state when creating the new mirror relationship.

Update mirror relationships when ONTAP is online

Mirror relationships can be updated any time after they are established. You can use the `state: promoted` or `state: reestablished` fields to update the relationships.
When promoting a destination volume to a regular ReadWrite volume, you can use *promotedSnapshotHandle* to specify a specific snapshot to restore the current volume to.

Update mirror relationships when ONTAP is offline

You can use a CRD to perform a SnapMirror update without Trident having direct connectivity to the ONTAP cluster. Refer to the following example format of the TridentActionMirrorUpdate:

Example

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` reflects the state of the TridentActionMirrorUpdate CRD. It can take a value from *Succeeded*, *In Progress*, or *Failed*.

Use CSI Topology

Trident can selectively create and attach volumes to nodes present in a Kubernetes cluster by making use of the CSI Topology feature.

Overview

Using the CSI Topology feature, access to volumes can be limited to a subset of nodes, based on regions and availability zones. Cloud providers today enable Kubernetes administrators to spawn nodes that are zone based. Nodes can be located in different availability zones within a region, or across various regions. To facilitate the provisioning of volumes for workloads in a multi-zone architecture, Trident uses CSI Topology.

Learn more about the CSI Topology feature here.

Kubernetes provides two unique volume binding modes:

- With `VolumeBindingMode` set to `Immediate`, Trident creates the volume without any topology awareness. Volume binding and dynamic provisioning are handled when the PVC is created. This is the default `VolumeBindingMode` and is suited for clusters that do not enforce topology constraints. Persistent Volumes are created without having any dependency on the requesting pod's scheduling requirements.
- With `VolumeBindingMode` set to `WaitForFirstConsumer`, the creation and binding of a Persistent Volume for a PVC is delayed until a pod that uses the PVC is scheduled and created. This way, volumes are created to meet the scheduling constraints that are enforced by topology requirements.

The `WaitForFirstConsumer` binding mode does not require topology labels. This can be used independent of the CSI Topology feature.

What you'll need

To make use of CSI Topology, you need the following:

- A Kubernetes cluster running a supported Kubernetes version

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- Nodes in the cluster should have labels that introduce topology awareness (`topology.kubernetes.io/region` and `topology.kubernetes.io/zone`). These labels **should be present on nodes in the cluster** before Trident is installed for Trident to be topology aware.

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

Step 1: Create a topology-aware backend

Trident storage backends can be designed to selectively provision volumes based on availability zones. Each backend can carry an optional `supportedTopologies` block that represents a list of zones and regions that are supported. For StorageClasses that make use of such a backend, a volume would only be created if requested by an application that is scheduled in a supported region/zone.

Here is an example backend definition:

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

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

`supportedTopologies` is used to provide a list of regions and zones per backend. These regions and zones represent the list of permissible values that can be provided in a StorageClass. For StorageClasses that contain a subset of the regions and zones provided in a backend, Trident creates a volume on the backend.

You can define `supportedTopologies` per storage pool as well. See the following example:

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

In this example, the `region` and `zone` labels stand for the location of the storage pool. `topology.kubernetes.io/region` and `topology.kubernetes.io/zone` dictate where the storage pools can be consumed from.

Step 2: Define StorageClasses that are topology aware

Based on the topology labels that are provided to the nodes in the cluster, StorageClasses can be defined to contain topology information. This will determine the storage pools that serve as candidates for PVC requests made, and the subset of nodes that can make use of the volumes provisioned by Trident.

See the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

In the StorageClass definition provided above, `volumeBindingMode` is set to `WaitForFirstConsumer`. PVCs that are requested with this StorageClass will not be acted upon until they are referenced in a pod. And, `allowedTopologies` provides the zones and region to be used. The `netapp-san-us-east1` StorageClass creates PVCs on the `san-backend-us-east1` backend defined above.

Step 3: Create and use a PVC

With the StorageClass created and mapped to a backend, you can now create PVCs.

See the example `spec` below:

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

Creating a PVC using this manifest would result in the following:

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

For Trident to create a volume and bind it to the PVC, use the PVC in a pod. See the following example:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

This podSpec instructs Kubernetes to schedule the pod on nodes that are present in the `us-east1` region, and choose from any node that is present in the `us-east1-a` or `us-east1-b` zones.

See the following output:

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

Update backends to include `supportedTopologies`

Pre-existing backends can be updated to include a list of `supportedTopologies` using `tridentctl backend update`. This will not affect volumes that have already been provisioned, and will only be used for subsequent PVCs.

Find more information

- Manage resources for containers
- nodeSelector
- Affinity and anti-affinity
- Taints and Tolerations

Work with snapshots

Kubernetes volume snapshots of Persistent Volumes (PVs) enable point-in-time copies of volumes. You can create a snapshot of a volume created using Trident, import a snapshot created outside of Trident, create a new volume from an existing snapshot, and recover volume data from snapshots.

Overview

Volume snapshot is supported by `ontap-nas`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `azure-netapp-files`, and `google-cloud-netapp-volumes` drivers.

Before you begin

You must have an external snapshot controller and Custom Resource Definitions (CRDs) to work with snapshots. This is the responsibility of the Kubernetes orchestrator (for example: Kubeadm, GKE, OpenShift).

If your Kubernetes distribution does not include the snapshot controller and CRDs, refer to Deploy a volume snapshot controller.

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

Create a volume snapshot

Steps

1. Create a `VolumeSnapshotClass`. For more information, refer to VolumeSnapshotClass.
   - The `driver` points to the Trident CSI driver.
   - `deletionPolicy` can be `Delete` or `Retain`. When set to `Retain`, the underlying physical snapshot on the storage cluster is retained even when the `VolumeSnapshot` object is deleted.

     Example

     ```
     cat snap-sc.yaml
     ```

     ```
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshotClass
     metadata:
       name: csi-snapclass
     driver: csi.trident.netapp.io
     deletionPolicy: Delete
     ```

2. Create a snapshot of an existing PVC.

   Examples

   - This example creates a snapshot of an existing PVC.

     ```
     cat snap.yaml
     ```

     ```
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshot
     metadata:
       name: pvc1-snap
     spec:
       volumeSnapshotClassName: csi-snapclass
       source:
         persistentVolumeClaimName: pvc1
     ```

   - This example creates a volume snapshot object for a PVC named `pvc1` and the name of the snapshot is set to `pvc1-snap`. A VolumeSnapshot is analogous to a PVC and is associated with a `VolumeSnapshotContent` object that represents the actual snapshot.

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

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

- You can identify the `VolumeSnapshotContent` object for the `pvc1-snap` VolumeSnapshot by describing it. The `Snapshot Content Name` identifies the VolumeSnapshotContent object which serves this snapshot. The `Ready To Use` parameter indicates that the snapshot can be used to create a new PVC.

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

Create a PVC from a volume snapshot

You can use `dataSource` to create a PVC using a VolumeSnapshot named `<pvc-name>` as the source of the data. After the PVC is created, it can be attached to a pod and used just like any other PVC.

> The PVC will be created in the same backend as the source volume. Refer to KB: Creating a PVC from a Trident PVC Snapshot cannot be created in an alternate backend.

The following example creates the PVC using `pvc1-snap` as the data source.

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

Import a volume snapshot

Trident supports the Kubernetes pre-provisioned snapshot process to enable the cluster administrator to create a `VolumeSnapshotContent` object and import snapshots created outside of Trident.

Before you begin

Trident must have created or imported the snapshot's parent volume.

Steps

1. **Cluster admin:** Create a `VolumeSnapshotContent` object that references the backend snapshot. This initiates the snapshot workflow in Trident.
   - Specify the name of the backend snapshot in `annotations` as `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`.
   - Specify `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` in `snapshotHandle`. This is the only information provided to Trident by the external snapshotter in the `ListSnapshots` call.

     > The `<volumeSnapshotContentName>` cannot always match the backend snapshot name due to CR naming constraints.

     Example

     The following example creates a `VolumeSnapshotContent` object that references backend snapshot `snap-01`.

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. **Cluster admin:** Create the `VolumeSnapshot` CR that references the `VolumeSnapshotContent` object. This requests access to use the `VolumeSnapshot` in a given namespace.

   Example

   The following example creates a `VolumeSnapshot` CR named `import-snap` that references the `VolumeSnapshotContent` named `import-snap-content`.

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: import-snap
   spec:
     # volumeSnapshotClassName: csi-snapclass (not required for pre-
   provisioned or imported snapshots)
     source:
       volumeSnapshotContentName: import-snap-content
   ```

3. **Internal processing (no action required):** The external snapshotter recognizes the newly created `VolumeSnapshotContent` and runs the `ListSnapshots` call. Trident creates the `TridentSnapshot`.
   - The external snapshotter sets the `VolumeSnapshotContent` to `readyToUse` and the `VolumeSnapshot` to `true`.
   - Trident returns `readyToUse=true`.
4. **Any user:** Create a `PersistentVolumeClaim` to reference the new `VolumeSnapshot`, where the `spec.dataSource` (or `spec.dataSourceRef`) name is the `VolumeSnapshot` name.

   Example

The following example creates a PVC referencing the `VolumeSnapshot` named `import-snap`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

Recover volume data using snapshots

The snapshot directory is hidden by default to facilitate maximum compatibility of volumes provisioned using the `ontap-nas` and `ontap-nas-economy` drivers. Enable the `.snapshot` directory to recover data from snapshots directly.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

In-place volume restoration from a snapshot

Trident provides rapid, in-place volume restoration from a snapshot using the `TridentActionSnapshotRestore` (TASR) CR. This CR functions as an imperative Kubernetes action and does not persist after the operation completes.

Trident supports snapshot restore on the `ontap-san`, `ontap-san-economy`, `ontap-nas`, `ontap-nas-flexgroup`, `azure-netapp-files`, `google-cloud-netapp-volumes`, and `solidfire-san` drivers.

Before you begin

You must have a bound PVC and available volume snapshot.

- Verify the PVC status is bound.

```
kubectl get pvc
```

- Verify the volume snapshot is ready to use.

```
kubectl get vs
```

Steps

1. Create the TASR CR. This example creates a CR for PVC `pvc1` and volume snapshot `pvc1-snapshot`.

   The TASR CR must be in a namespace where the PVC & VS exist.

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. Apply the CR to restore from the snapshot. This example restores from snapshot `pvc1`.

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

Results

Trident restores the data from the snapshot. You can verify the snapshot restore status:

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- In most cases, Trident will not automatically retry the operation in case of failure. You will need to perform the operation again.
- Kubernetes users without admin access might have to be granted permission by the admin to create a TASR CR in their application namespace.

Delete a PV with associated snapshots

When deleting a Persistent Volume with associated snapshots, the corresponding Trident volume is updated to a "Deleting state". Remove the volume snapshots to delete the Trident volume.

Deploy a volume snapshot controller

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

Steps

1. Create volume snapshot CRDs.

```
cat snapshot-setup.sh
```

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

2. Create the snapshot controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

If necessary, open `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` and update `namespace` to your namespace.

Related links

- Volume snapshots
- VolumeSnapshotClass

Work with volume group snapshots

Kubernetes volume group snapshots of Persistent Volumes (PVs) NetApp Trident provides the ability to create snapshots of multiple volumes (a group of volume snapshots). This volume group snapshot represents copies from multiple volumes that are taken at the same point-in-time.

VolumeGroupSnapshot is a beta feature in Kubernetes with beta APIs. Kubernetes 1.32 is the minimum version required for VolumeGroupSnapshot.

Create volume group snapshots

Volume group snapshot is supported with the following storage drivers:

- `ontap-san` driver - only for the iSCSI and FC protocols, not for the NVMe/TCP protocol.
- `ontap-san-economy` - only for the iSCSI protocol.
- `ontap-nas`

> Volume group snapshot is not supported for NetApp ASA r2 or AFX storage systems.

Before you begin

- Ensure that your Kubernetes version is K8s 1.32 or higher.
- You must have an external snapshot controller and Custom Resource Definitions (CRDs) to work with snapshots. This is the responsibility of the Kubernetes orchestrator (for example: Kubeadm, GKE, OpenShift).

  If your Kubernetes distribution does not include the external snapshot controller and CRDs, refer to Deploy a volume snapshot controller.

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

- In the snapshot controller YAML, set the `CSIVolumeGroupSnapshot` feature gate to 'true' to ensure that volume group snapshot is enabled.
- Create the required volume group snapshot classes before creating a volume group snapshot.
- Ensure that all PVCs/volumes are on the same SVM to be able to create VolumeGroupSnapshot.

Steps

- Create a VolumeGroupSnapshotClass prior to creating a VolumeGroupSnapshot. For more information, refer to VolumeGroupSnapshotClass.

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- Create PVCs with required labels using existing storage classes, or add these labels to existing PVCs.

  The following example creates the PVC using `pvc1-group-snap` as the data source and label `consistentGroupSnapshot: groupA`. Define the label key and value based on your requirements.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- Create a VolumeGroupSnapshot with the same label (`consistentGroupSnapshot: groupA`) specified in the PVC.

  This example creates a volume group snapshot:

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

Recover volume data using a group snapshot

You can restore individual Persistent Volumes using the individual snapshots which have been created as part of the Volume Group Snapshot. You cannot recover the Volume Group Snapshot as a unit.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

In-place volume restoration from a snapshot

Trident provides rapid, in-place volume restoration from a snapshot using the `TridentActionSnapshotRestore` (TASR) CR. This CR functions as an imperative Kubernetes action and does not persist after the operation completes.

For more information, see In-place volume restoration from a snapshot.

Delete a PV with associated group snapshots

When deleting a group volume snapshot:

- You can delete VolumeGroupSnapshots as a whole, not individual snapshots in the group.
- If PersistentVolumes are deleted while a snapshot exists for that PersistentVolume, Trident will move that volume to a "deleting" state because the snapshot must be removed before the volume can be safely removed.
- If a clone has been created using a grouped snapshot and then the group is to be deleted, a split-on-clone operation will begin and the group cannot be deleted until the split is complete.

Deploy a volume snapshot controller

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

Steps

1. Create volume snapshot CRDs.

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.yaml
```

2. Create the snapshot controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

If necessary, open `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` and update `namespace` to your namespace.

Related links

- VolumeGroupSnapshotClass
- Volume snapshots