



Use Astra Trident

Astra Trident

NetApp
October 22, 2024

Table of Contents

- Use Astra Trident 1
 - Prepare the worker node 1
 - Configure and manage backends 5
 - Create and manage storage classes 115
 - Provision and manage volumes 120

Use Astra 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 or iSCSI tools based on your driver selection.

Selecting the right tools

If you are using a combination of drivers, you should install NFS and iSCSI tools.

NFS tools

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

iSCSI tools

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



Recent versions of RedHat CoreOS have NFS and iSCSI installed by default.

Node service discovery

Astra 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

Astra 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

Astra 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

Astra 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, Astra 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. Astra 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 Astra Trident installation.

Astra 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, Astra 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

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/RedHat CoreOS with iSCSI PVs, specify the `discard` `mountOption` in the `StorageClass` to perform inline space reclamation. See [RedHat documentation](#).

RHEL 8+

1. Install the following system packages:

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_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.



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

Configure and manage backends

Configure backends

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

Astra 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 Cloud Volumes Service for Google Cloud Platform 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 Astra Trident with Amazon FSx for NetApp ONTAP](#)

Azure NetApp Files

Configure an Azure NetApp Files backend

You can configure Azure NetApp Files as the backend for Astra Trident. You can attach NFS and SMB volumes using an Azure NetApp Files backend.

Azure NetApp Files driver details

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

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

Considerations

- The Azure NetApp Files service does not support volumes smaller than 100 GB. Astra Trident automatically creates 100-GB volumes if a smaller volume is requested.
- Astra Trident supports SMB volumes mounted to pods running on Windows nodes only.

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.

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:

- 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 Astra Trident requires. After creating the custom role, [assign the role using the Azure portal](#).

```
{
  "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/MountTargets/read",
          "Microsoft.Network/virtualNetworks/read",
```

```

        "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/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.

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 2019. Astra Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Astra 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

Astra 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.



Astra Trident does not support Manual QoS capacity pools.

Azure NetApp Files backends provide these configuration options.

| Parameter | Description | Default |
|--------------------------------|--|---------------------------------------|
| <code>version</code> | | Always 1 |
| <code>storageDriverName</code> | Name of the storage driver | "azure-netapp-files" |
| <code>backendName</code> | Custom name of the storage backend | Driver name + "_" + random characters |
| <code>subscriptionID</code> | The subscription ID from your Azure subscription | |
| <code>tenantID</code> | The tenant ID from an App Registration | |
| <code>clientID</code> | The client ID from an App Registration | |
| <code>clientSecret</code> | The client secret from an App Registration | |
| <code>serviceLevel</code> | One of Standard, Premium, or Ultra | "" (random) |
| <code>location</code> | Name of the Azure location where the new volumes will be created | |
| <code>resourceGroups</code> | List of resource groups for filtering discovered resources | [] (no filter) |
| <code>netappAccounts</code> | List of NetApp accounts for filtering discovered resources | [] (no filter) |
| <code>capacityPools</code> | List of capacity pools for filtering discovered resources | [] (no filter, random) |
| <code>virtualNetwork</code> | Name of a virtual network with a delegated subnet | "" |
| <code>subnet</code> | Name of a subnet delegated to Microsoft.Netapp/volumes | "" |

| Parameter | Description | Default |
|------------------------------|---|------------------------------|
| <code>networkFeatures</code> | <p>Set of VNet features for a volume, may be <code>Basic</code> or <code>Standard</code>.</p> <p>Network Features is not available in all regions and might have to be enabled in a subscription. Specifying <code>networkFeatures</code> when the functionality is not enabled causes volume provisioning to fail.</p> | "" |
| <code>nfsMountOptions</code> | <p>Fine-grained control of NFS mount options.</p> <p>Ignored for SMB volumes.</p> <p>To mount volumes using NFS version 4.1, include <code>nfsvers=4</code> in the comma-delimited mount options list to choose NFS v4.1.</p> <p>Mount options set in a storage class definition override mount options set in backend configuration.</p> | "nfsvers=3" |
| <code>limitVolumeSize</code> | Fail provisioning if the requested volume size is above this value | "" (not enforced by default) |
| <code>debugTraceFlags</code> | <p>Debug flags to use when troubleshooting. Example, <code>\{"api": false, "method": true, "discovery": true\}</code>. Do not use this unless you are troubleshooting and require a detailed log dump.</p> | null |
| <code>nasType</code> | <p>Configure NFS or SMB volumes creation.</p> <p>Options are <code>nfs</code>, <code>smb</code> or <code>null</code>. Setting to <code>null</code> defaults to NFS volumes.</p> | <code>nfs</code> |



For more information on Network Features, refer to [Configure network features for an Azure NetApp Files volume](#).

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, Astra Trident will log 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.

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 |
|------------------------------|--|---|
| <code>exportRule</code> | Export rules for new volumes. <code>exportRule</code> 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" |
| <code>snapshotDir</code> | Controls visibility of the <code>.snapshot</code> directory | "false" |
| <code>size</code> | The default size of new volumes | "100G" |
| <code>unixPermissions</code> | The unix permissions of new volumes (4 octal digits). Ignored for SMB volumes. | "" (preview feature, requires whitelisting in subscription) |

Example configurations

Example 1: Minimal configuration

This is the absolute minimum backend configuration. With this configuration, Astra 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.

```
---
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
```

Example 2: Specific service level configuration with capacity pool filters

This backend configuration places volumes in Azure's `eastus` location in an `Ultra` capacity pool. Astra 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
```

Example 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: my-virtual-network
subnet: 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'
```

Example 4: 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:
  - ultra-1
  - ultra-2
  networkFeatures: Standard
- labels:
  performance: silver
  serviceLevel: Premium
  capacityPools:
  - premium-1
- labels:
  performance: bronze
  serviceLevel: Standard
  capacityPools:
  - standard-1
  - standard-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.

Example 1: 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"
```

Example 2: 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}
```

Example 3: 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 the backend creation fails, something is wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs
```

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

Configure a Cloud Volumes Service for Google Cloud backend

Learn how to configure NetApp Cloud Volumes Service for Google Cloud as the backend for your Astra Trident installation using the sample configurations provided.

Google Cloud driver details

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

| Driver | Protocol | volumeMode | Access modes supported | File systems supported |
|----------------------|----------|------------|------------------------|------------------------|
| <code>gcp-cvs</code> | NFS | Filesystem | RWO, ROX, RWX, RWOP | <code>nfs</code> |

Learn about Astra Trident support for Cloud Volumes Service for Google Cloud

Astra Trident can create Cloud Volumes Service volumes in one of two [service types](#):

- **CVS-Performance:** The default Astra Trident service type. This performance-optimized service type is best suited for production workloads that value performance. The CVS-Performance service type is a hardware option supporting volumes with a minimum 100 GiB size. You can choose one of [three service levels](#):
 - `standard`
 - `premium`
 - `extreme`
- **CVS:** The CVS service type provides high zonal availability with limited to moderate performance levels. The CVS service type is a software option that uses storage pools to support volumes as small as 1 GiB. The storage pool can contain up to 50 volumes where all volumes share the capacity and performance of the pool. You can choose one of [two service levels](#):
 - `standardsw`

What you'll need

To configure and use the [Cloud Volumes Service for Google Cloud](#) backend, you need the following:

- A Google Cloud account configured with NetApp Cloud Volumes Service
- Project number of your Google Cloud account
- Google Cloud service account with the `netappcloudvolumes.admin` role
- API key file for your Cloud Volumes Service account

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 |
|--------------------------------|--|-------------------------------------|
| <code>version</code> | | Always 1 |
| <code>storageDriverName</code> | Name of the storage driver | "gcp-cvs" |
| <code>backendName</code> | Custom name or the storage backend | Driver name + "_" + part of API key |
| <code>storageClass</code> | Optional parameter used to specify the CVS service type. Use <code>software</code> to select the CVS service type. Otherwise, Astra Trident assumes CVS-Performance service type (<code>hardware</code>). | |
| <code>storagePools</code> | CVS service type only. Optional parameter used to specify storage pools for volume creation. | |
| <code>projectNumber</code> | Google Cloud account project number. The value is found on the Google Cloud portal home page. | |
| <code>hostProjectNumber</code> | Required if using a shared VPC network. In this scenario, <code>projectNumber</code> is the service project, and <code>hostProjectNumber</code> is the host project. | |
| <code>apiRegion</code> | The Google Cloud region where Astra Trident creates Cloud Volumes Service volumes. When creating cross-region Kubernetes clusters, volumes created in an <code>apiRegion</code> can be used in workloads scheduled on nodes across multiple Google Cloud regions. Cross-region traffic incurs an additional cost. | |
| <code>apiKey</code> | API key for the Google Cloud service account with the <code>netappcloudvolumes.admin</code> role. It includes the JSON-formatted contents of a Google Cloud service account's private key file (copied verbatim into the backend configuration file). | |

| Parameter | Description | Default |
|-------------------|--|---|
| proxyURL | <p>Proxy URL if proxy server required to connect to CVS account. The proxy server can either be an HTTP proxy or an HTTPS proxy.</p> <p>For an HTTPS proxy, certificate validation is skipped to allow the usage of self-signed certificates in the proxy server.</p> <p>Proxy servers with authentication enabled are not supported.</p> | |
| 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 | <p>The CVS-Performance or CVS service level for new volumes.</p> <p>CVS-Performance values are <code>standard</code>, <code>premium</code>, or <code>extreme</code>.</p> <p>CVS values are <code>standardsw</code> or <code>zoneredundantstandardsw</code>.</p> | <p>CVS-Performance default is "standard".</p> <p>CVS default is "standardsw".</p> |
| network | Google Cloud network used for Cloud Volumes Service volumes. | "default" |
| debugTraceFlags | <p>Debug flags to use when troubleshooting. Example, <code>\{"api":false, "method":true\}</code>.</p> <p>Do not use this unless you are troubleshooting and require a detailed log dump.</p> | null |
| allowedTopologies | <p>To enable cross-region access, your StorageClass definition for <code>allowedTopologies</code> must include all regions.</p> <p>For example:</p> <ul style="list-style-type: none"> - key: <code>topology.kubernetes.io/region</code> values: - <code>us-east1</code> - <code>eu-west1</code> | |

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 or IPv4 subnets in CIDR notation. | "0.0.0.0/0" |

| Parameter | Description | Default |
|-----------------|--|---|
| snapshotDir | Access to the .snapshot directory | "false" |
| snapshotReserve | Percentage of volume reserved for snapshots | "" (accept CVS default of 0) |
| size | The size of new volumes. CVS-Performance minimum is 100 GiB. CVS minimum is 1 GiB. | CVS-Performance service type defaults to "100GiB". CVS service type does not set a default but requires a 1 GiB minimum. |

CVS-Performance service type examples

The following examples provide sample configurations for the CVS-Performance service type.

Example 1: Minimal configuration

This is the minimum backend configuration using default CVS-Performance service type with the default "standard" service level.

```
---  
version: 1  
storageDriverName: gcp-cvs  
projectNumber: '012345678901'  
apiRegion: us-west2  
apiKey:  
  type: service_account  
  project_id: my-gcp-project  
  private_key_id: "<id_value>"  
  private_key: |  
    -----BEGIN PRIVATE KEY-----  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m  
    XsYg6gyxy4zq7OlwWgLwGa==  
    -----END PRIVATE KEY-----  
  client_email: cloudvolumes-admin-sa@my-gcp-  
project.iam.gserviceaccount.com  
  client_id: '123456789012345678901'
```

```
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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```



```
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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```



```

-----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
  nfsMountOptions: vers=3,proto=tcp,timeo=600
  defaults:
    snapshotReserve: '5'
    exportRule: 0.0.0.0/0
  labels:
    cloud: gcp
  region: us-west2
  storage:
- labels:
  performance: extreme
  protection: extra
  serviceLevel: extreme
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
- labels:
  performance: extreme
  protection: standard
  serviceLevel: extreme
- labels:
  performance: premium
  protection: extra
  serviceLevel: premium
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
- labels:
  performance: premium
  protection: standard
  serviceLevel: premium
- labels:
  performance: standard
  serviceLevel: standard

```

Storage class definitions

The following StorageClass definitions apply to the virtual pool configuration example. Using `parameters.selector`, you can specify for each StorageClass the virtual pool used to host a volume. The volume will have the aspects defined in the chosen pool.

Storage class example

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=extreme; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-standard-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-standard
provisioner: netapp.io/trident
parameters:
  selector: "performance=standard"
allowVolumeExpansion: true
```

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "protection=extra"
allowVolumeExpansion: true
```

- The first StorageClass (`cvs-extreme-extra-protection`) maps to the first virtual pool. This is the only pool offering extreme performance with a snapshot reserve of 10%.
- The last StorageClass (`cvs-extra-protection`) calls out any storage pool which provides a snapshot reserve of 10%. Astra Trident decides which virtual pool is selected and ensures that the snapshot reserve requirement is met.

CVS service type examples

The following examples provide sample configurations for the CVS service type.


```
client_id: '123456789012345678901'  
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/cloudvolumes-admin-  
sa%40my-gcp-project.iam.gserviceaccount.com  
serviceLevel: standardsw
```

Example 2: Storage pool configuration

This sample backend configuration uses `storagePools` to configure a storage pool.

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    MIIEvAIBADANBgkqhkiG9w0BAQEFAASCbKwggSiAgEAAoIBAQDaT+Oui9FBAw19
    L1AGEkrYU5xd9K5NlO5jMkIFND5wCD+Nv+jd1GvtFRLaLK5RvXyF5wzvztmODNS+
    qtScpQ+5cFpQkuGtv9U9+N6qtuVYYO3b504Kp5CtqVPJCgMJaK2j8pZTIqUiMum/
    5/Y9oTbZrjAHSMsgJm2nHzFq2X0rqVmaHghI6ATm4DOuWx8XGWKGTGIPlc0qPqJlqS
    LLaWOH4VIZQZCAyW5IUp9CAmwqHgdG0uhFNfCgMmED6PBUvVLsLvcq86X+QSWR9k
    ETqElj/sGCenPF7ti1DhGBFafd9hPnxg9PZY29ArEZwY9G/ZjZQX7WPgs0VvxiNR
    DxZRC3GXAgMBAAECggEACn5c59bG/qnVEVI1CwMAalM5M2z09JFh1L1ljKwntNPj
    Vilw2eTW2+UE7HbJru/S7KQgA5Dnn9kvCraEahPRuddUMrD0vG4kTl/IODV6uFuk
    Y0sZfbqd4jMUQ21smvGsqFzwloYWS5qzO1W83ivXH/HW/iqkmY2eW+EPRS/hwSSu
    SscR+SojI7PB0BWSJhlV4yqYf3vcd/D95el2CVHfRCkL85DKumeZ+yHEnpiXGZAE
    t8xSs4a500Pm6NHhevCw2a/UQ95/foXNUR450HtbjieJo5o+FF6EYZQGfU2ZHZO8
    37FBKuaJkdGW5xqaI9TL7aqkGkFMF4F2qvOZM+vy8QKBgQD4oVuOkJD1hkTHP86W
    esFlw1kpWyJR9ZA7LI0g/rVpslnX+XdDq0WQf4umdLNau5hYEH9LU6ZSGs1Xk3/B
    NHwR6OXFuqEKNiu83d0zSlHhTy7PZpOZdj5a/vVvQfPDMz7OvsqLRd7YCAbdzuQ0
    +Ahq0Ztwvg0HQ64hdW0ukpYRRwKBgQDgyHj98oqsw0YuIa+pP1yS0pPwLmjwKyNm
    /HayzCp+Qjiiyy7Tzg8AUqlH1Ou83XbV428jvg7kDh07PCCKFq+mMmfqHmTpb0Maq
    KpKnZg4ipsqPlyHNNEoRmcailXbwIhCLewMqMrggUiLOmCw4PscL5nK+4GKu2XE1
    jLqjWAZFMQKBgFHkQ9XXRAJ1kR3XpGHoGN890pZ0kCVSrqju6aUef/5KY1FCt8ew
    F/+aIxM2iQsvmWQYOvVCnhuY/F2GfAQ7d0om3decuwI0CX/xy7PjHmLXa2uaZs4
    WR17sLduj62RqXRLX0c0QkwBiNFyHbRcpdkZJQujbyMhBa+7j7SxT4BtAoGAWMWT
    UucocRXZm/pdvz9wteNH3YDwnJLMxm1KC06qMXbBoYrliY4sm3ywJWMC+iCd/H8A
    Gecxd/xVu5mA2L2N3KMq18Zhz8Th0G5DwKyDRJgOQ0Q46yuNXOoYEjlo4Wjyk8Me
    +tlQ8iK98E0UmZnhTgfSpSNElzbz2AqnzQ3MN9uECgYAqdvdpVnKGFvdtZ2DjyMoJ
    E89UIC41WjjJGmHsd8W65+3X0RwMzKMT6aZc5tK9J5dHvmWIETnbM+1TImdbBFga
    NWOC6f3r2xbGXHhaWS1+nobpTuvlo56ZRJVvV71FMsiddzMuHH8pxfgNjemwA4P
    ThDHcejv035NNV6Kyo00tA==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
  data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
```

```
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/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

What's next?

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

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

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

```
tridentctl logs
```

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

Configure a NetApp HCI or SolidFire backend

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

Element driver details

Astra 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, Astra 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. See [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 of 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. Astra 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) |

| Parameter | Description | 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.

Astra 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. Astra Trident will decide which virtual pool is selected and will ensure 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

Astra 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).



If you are using Astra Control for protection, recovery, and mobility, read [Astra Control driver compatibility](#).

| Driver | Protocol | volumeMode | Access modes supported | File systems supported |
|-------------------|----------|------------|---|---------------------------------|
| ontap-san | iSCSI | Block | RWO, ROX, RWX, RWOP | No filesystem; raw block device |
| ontap-san | iSCSI | Filesystem | RWO, RWOP ROX and RWX are not available in Filesystem volume mode. | xf ^s , 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. | xf ^s , ext3, ext4 |

Astra Control driver compatibility

Astra Control provides seamless protection, disaster recovery, and mobility (moving volumes between Kubernetes clusters) for volumes created with the `ontap-nas`, `ontap-nas-flexgroup`, and `ontap-san` drivers. See [Astra Control replication prerequisites](#) for details.



- 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 `ontap-nas-economy` if you anticipate the need for data protection, disaster recovery, or mobility.

User permissions

Astra 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, Astra 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 Astra 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 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, Astra Trident requires at least one aggregate assigned to the SVM.

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

Astra 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:** Astra 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

Astra 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 Astra Trident releases. A custom security login role can be created and used with Astra Trident, but is not recommended.

A sample backend definition will look like this:

YAML

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

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
```

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...Vaaalllluuueeee",
  "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 Astra Trident can communicate with the ONTAP backend and handle future volume operations.

Authenticate connections with bidirectional CHAP

Astra 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`, Astra 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 Astra 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, Astra 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, Astra 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, Astra Trident creates a corresponding `tridentbackend` CRD and stores the CHAP secrets and usernames as Kubernetes secrets. All PVs that are created by Astra 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 through the CLI/ONTAP UI as Astra 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 Astra Trident on the SVM. New connections will 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 Astra Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses.

Backend configuration options

See the following table for the backend configuration options:

| Parameter | Description | Default |
|-----------|-------------|----------|
| version | | Always 1 |

| Parameter | Description | Default |
|---------------------------|--|---|
| storageDriverName | Name of the storage driver | ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy |
| backendName | Custom name of the storage backend | Driver name + "_" + dataLIF |
| managementLIF | <p>IP address of a cluster or SVM management LIF.</p> <p>A fully-qualified domain name (FQDN) can be specified.</p> <p>Can be set to use IPv6 addresses if Astra 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].</p> <p>For seamless MetroCluster switchover, see the MetroCluster example.</p> | "10.0.0.1", "[2001:1234:abcd::fefe]" |
| dataLIF | <p>IP address of protocol LIF.</p> <p>Do not specify for iSCSI. Astra 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.</p> <p>Omit for Metrocluster. See the MetroCluster example.</p> | Derived by the SVM |
| svm | <p>Storage virtual machine to use</p> <p>Omit for Metrocluster. See the MetroCluster example.</p> | Derived if an SVM managementLIF is specified |
| useCHAP | <p>Use CHAP to authenticate iSCSI for ONTAP SAN drivers [Boolean].</p> <p>Set to true for Astra 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.</p> | 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 | "" |

| Parameter | Description | Default |
|----------------------|--|--|
| 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. | "" |
| username | Username needed to communicate with the ONTAP cluster. Used for credential-based authentication. | "" |
| password | Password needed to communicate with the ONTAP cluster. 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 later. To update this parameter, you will need to create a new backend. | trident |
| 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 Astra Trident. | "" (not enforced by default) |
| 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. | "" (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 |

| Parameter | Description | Default |
|-----------|---|---------|
| useREST | <p>Boolean parameter to use ONTAP REST APIs. Tech preview</p> <p>useREST is provided as a tech preview that is recommended for test environments and not for production workloads. When set to <code>true</code>, Astra 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 <code>ontap</code> application. This is satisfied by the pre-defined <code>vsadmin</code> and <code>cluster-admin</code> roles.</p> <p>useREST is not supported with MetroCluster.</p> | false |

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" |
| qosPolicy | <p>QoS policy group to assign for volumes created. Choose one of <code>qosPolicy</code> or <code>adaptiveQosPolicy</code> per storage pool/backend.</p> <p>Using QoS policy groups with Astra Trident requires ONTAP 9.8 or later. We recommend using a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group will enforce the ceiling for the total throughput of all workloads.</p> | "" |
| adaptiveQosPolicy | Adaptive QoS policy group to assign for volumes created. Choose one of <code>qosPolicy</code> or <code>adaptiveQosPolicy</code> per storage pool/backend | "" |
| snapshotReserve | Percentage of volume reserved for snapshots | "0" if <code>snapshotPolicy</code> is "none", otherwise "" |

| Parameter | Description | Default |
|----------------|--|--|
| splitOnClone | Split a clone from its parent upon creation | "false" |
| encryption | <p>Enable NetApp Volume Encryption (NVE) on the new volume; defaults to <code>false</code>. NVE must be licensed and enabled on the cluster to use this option.</p> <p>If NAE is enabled on the backend, any volume provisioned in Astra Trident will be NAE enabled.</p> <p>For more information, refer to: How Astra Trident works with NVE and NAE.</p> | "false" |
| luksEncryption | Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS) . | "" |
| securityStyle | Security style for new volumes | unix |
| tieringPolicy | Tiering policy to use "none" | "snapshot-only" for pre-ONTAP 9.5 SVM-DR configuration |

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, Astra 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. Astra 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`, Astra Trident calculates the size of volumes as follows:

$$\text{Total volume size} = [(\text{PVC requested size}) / (1 - (\text{snapshotReserve percentage} / 100))] * 1.1$$

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

| Vserver | Volume | Aggregate | State | Type | Size | Available | Used% |
|---------|--------|---|--------|------|--------|-----------|-------|
| | | _pvc_89f1c156_3801_4de4_9f9d_034d54c395f4 | online | RW | 10GB | 5.00GB | 0% |
| | | _pvc_e42ec6fe_3baa_4af6_996d_134adbbb8e6d | online | RW | 5.79GB | 5.50GB | 0% |
| | | _pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba | online | RW | 1GB | 511.8MB | 0% |

3 entries were displayed.

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 Astra Trident, we recommend 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>
```

ONTAP SAN economy example

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
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-san
managementLIF: 192.168.1.66
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: c19qxIm36DKyawxy
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>
```

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.

Astra Trident sets provisioning labels in the "Comments" field. Comments are set on the FlexVol. Astra 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'
```

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: netapp.io/trident
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: netapp.io/trident
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: netapp.io/trident
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: netapp.io/trident
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: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

Astra Trident will decide which virtual pool is selected and will ensure 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

Astra 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).



If you are using Astra Control for protection, recovery, and mobility, read [Astra Control driver compatibility](#).

| 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 |
| ontap-nas-flexgroup | NFS SMB | Filesystem | RWO, ROX, RWX, RWOP | "", nfs, smb |

Astra Control driver compatibility

Astra Control provides seamless protection, disaster recovery, and mobility (moving volumes between Kubernetes clusters) for volumes created with the `ontap-nas`, `ontap-nas-flexgroup`, and `ontap-san` drivers. See [Astra Control replication prerequisites](#) for details.



- 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 `ontap-nas-economy` if you anticipate the need for data protection, disaster recovery, or mobility.

User permissions

Astra 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, Astra 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 Astra 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.

Requirements

- For all ONTAP backends, Astra Trident requires at least one aggregate 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. See [here](#) for more details.
- Astra 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

Astra 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: Astra 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

Astra 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 Astra Trident releases. A custom security login role can be created and used with Astra Trident, but is not recommended.

A sample backend definition will look like this:

YAML

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

JSON

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "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/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...Vaaalllluuueeee",
  "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 Astra Trident can communicate with the ONTAP backend and handle future volume operations.

Manage NFS export policies

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

Astra Trident provides two options when working with export policies:

- Astra 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. Astra Trident adds node IPs that fall in these ranges to the export policy automatically. Alternatively, when no CIDRs are specified, any global-scoped unicast IP found on the nodes will be added to the export policy.
- Storage administrators can create an export policy and add rules manually. Astra Trident uses the default

export policy unless a different export policy name is specified in the configuration.

Dynamically manage export policies

Astra 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 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
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 Astra Trident.

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

- `autoExportPolicy` is set to `true`. This indicates that Astra Trident will create an export policy for the `svm1` SVM and handle the addition and deletion of rules using `autoExportCIDRs` address blocks. For example, a backend with UUID `403b5326-8482-40db-96d0-d83fb3f4daec` and `autoExportPolicy` set to `true` creates an export policy named `trident-403b5326-8482-40db-96d0-d83fb3f4daec` on the SVM.
- `autoExportCIDRs` contains a list of address blocks. This field is optional and it defaults to `["0.0.0.0", "::/0"]`. If not defined, Astra Trident adds all globally-scoped unicast addresses found on the worker nodes.

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 will be added to the export policy that Astra Trident creates. When Astra 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`. After filtering the IPs, Astra Trident creates export policy rules for the client IPs it discovers, with one rule for each node it identifies.

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 Astra 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>
      fileType: ext4
```

As nodes are added to a Kubernetes cluster and registered with the Astra Trident controller, export policies of existing backends are updated (provided they fall in the address range specified in `autoExportCIDRs` for the backend).

When a node is removed, Astra Trident checks all backends that are online to remove the access rule for the node. By removing this node IP from the export policies of managed backends, Astra 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` will ensure that Astra Trident manages the export policies automatically. This will create a new export policy named after the backend's UUID and volumes that are present on the backend will use the newly created export policy when they are 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 Astra Trident pod on the node. Astra 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. Failure to configure either of these protocols will cause SMB volume creation to fail.

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 2019. Astra Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Astra 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 Astra 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 Astra 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 <code>nfs</code> . | smb |
| securityStyle | Security style for new volumes. Must be set to <code>ntfs</code> or <code>mixed</code> for SMB volumes. | <code>ntfs</code> or <code>mixed</code> for SMB volumes |
| unixPermissions | Mode for new volumes. Must be left empty for SMB volumes. | "" |

ONTAP NAS configuration options and examples

Learn to create and use ONTAP NAS drivers with your Astra Trident installation. This section provides backend configuration examples and details for mapping backends to StorageClasses.

Backend configuration options

See the following table for the backend configuration options:

| Parameter | Description | Default |
|-----------|-------------|----------|
| version | | Always 1 |

| Parameter | Description | Default |
|-------------------|--|---|
| storageDriverName | Name of the storage driver | "ontap-nas", "ontap-nas-economy", "ontap-nas-flexgroup", "ontap-san", "ontap-san-economy" |
| backendName | Custom name of the storage backend | Driver name + "_" + dataLIF |
| managementLIF | <p>IP address of a cluster or SVM management LIF</p> <p>A fully-qualified domain name (FQDN) can be specified.</p> <p>Can be set to use IPv6 addresses if Astra 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].</p> <p>For seamless MetroCluster switchover, see the MetroCluster example.</p> | "10.0.0.1", "[2001:1234:abcd::fefe]" |
| dataLIF | <p>IP address of protocol LIF.</p> <p>We recommend specifying dataLIF. If not provided, Astra Trident fetches data LIFs 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 data LIFs.</p> <p>Can be changed after initial setting. Refer to Update dataLIF after initial configuration.</p> <p>Can be set to use IPv6 addresses if Astra 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].</p> <p>Omit for Metrocluster. See the MetroCluster example.</p> | Specified address or derived from SVM, if not specified (not recommended) |
| svm | <p>Storage virtual machine to use</p> <p>Omit for Metrocluster. See the MetroCluster example.</p> | Derived if an SVM managementLIF is specified |
| autoExportPolicy | <p>Enable automatic export policy creation and updating [Boolean].</p> <p>Using the autoExportPolicy and autoExportCIDRs options, Astra Trident can manage export policies automatically.</p> | false |

| Parameter | Description | Default |
|----------------------|--|------------------------------|
| autoExportCIDsRs | List of CIDRs to filter Kubernetes' node IPs against when autoExportPolicy is enabled. Using the autoExportPolicy and autoExportCIDsRs options, Astra 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 | "" |
| 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 | |
| password | Password to connect to the cluster/SVM. Used for credential-based auth | |
| storagePrefix | Prefix used when provisioning new volumes in the SVM. Cannot be updated after you set it | "trident" |
| limitAggregateUsage | Fail provisioning if usage is above this percentage. Does not apply to Amazon FSx for ONTAP | "" (not enforced by default) |
| 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. | "" (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 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. | nfs |

| Parameter | Description | Default |
|------------------|---|-----------|
| nfsMountOptions | <p>Comma-separated list of NFS mount options.</p> <p>The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Astra Trident will fall back to using the mount options specified in the storage backend's configuration file.</p> <p>If no mount options are specified in the storage class or the configuration file, Astra Trident will not set any mount options on an associated persistent volume.</p> | "" |
| qtreesPerFlexvol | Maximum Qtrees per FlexVol, must be in range [50, 300] | "200" |
| smbShare | <p>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 Astra Trident to create the SMB share; or you can leave the parameter blank to prevent common share access to volumes.</p> <p>This parameter is optional for on-premises ONTAP.</p> <p>This parameter is required for Amazon FSx for ONTAP backends and cannot be blank.</p> | smb-share |
| useREST | <p>Boolean parameter to use ONTAP REST APIs. Tech preview</p> <p>useREST is provided as a tech preview that is recommended for test environments and not for production workloads. When set to <code>true</code>, Astra 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 <code>ontap</code> application. This is satisfied by the pre-defined <code>vsadmin</code> and <code>cluster-admin</code> roles.</p> <p>useREST is not supported with MetroCluster.</p> | false |

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" |

| Parameter | Description | Default |
|-------------------|---|--|
| 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 | "" |
| 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 <code>false</code> . NVE must be licensed and enabled on the cluster to use this option. If NAE is enabled on the backend, any volume provisioned in Astra Trident will be NAE enabled. For more information, refer to: How Astra Trident works with NVE and NAE . | "false" |
| tieringPolicy | Tiering policy to use "none" | "snapshot-only" for pre-ONTAP 9.5 SVM-DR configuration |
| unixPermissions | Mode for new volumes | "777" for NFS volumes; empty (not applicable) for SMB volumes |
| snapshotDir | Controls access to the <code>.snapshot</code> directory | "false" |
| exportPolicy | Export policy to use | "default" |
| securityStyle | Security style for new volumes. NFS supports <code>mixed</code> and <code>unix</code> security styles. SMB supports <code>mixed</code> and <code>ntfs</code> security styles. | NFS default is <code>unix</code> . SMB default is <code>ntfs</code> . |



Using QoS policy groups with Astra Trident requires ONTAP 9.8 or later. It is recommended to use a non-shared QoS policy group and ensure the policy group is applied to each constituent individually. A shared QoS policy group will enforce 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`, Astra 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, Astra 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 lesser space than what they requested. Before v21.07, when the user requests a PVC (for example, 5GiB), with the `snapshotReserve` to 50 percent, they get only 2.5GiB 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 Astra 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 = 5GiB`, the total volume size is $2/.5 = 10\text{GiB}$ and the available size is 5GiB, which is what the user requested in the PVC request. The `volume show` command should show results similar to this example:

| Vserver | Volume | Aggregate | State | Type | Size | Available | Used% |
|---------|---|-----------|--------|------|------|-----------|-------|
| | _pvc_89f1c156_3801_4de4_9f9d_034d54c395f4 | | online | RW | 10GB | 5.00GB | 0% |
| | _pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba | | online | RW | 1GB | 511.8MB | 0% |

2 entries were displayed.

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

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.

Astra Trident sets provisioning labels in the "Comments" field. Comments are set on FlexVol for `ontap-nas` or FlexGroup for `ontap-nas-flexgroup`. Astra 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: netapp.io/trident  
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: netapp.io/trident  
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: netapp.io/trident
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: netapp.io/trident
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: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

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

Update dataLIF after initial configuration

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

```
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 for the new data LIF to take effect.

Amazon FSx for NetApp ONTAP

Use Astra Trident with Amazon FSx for NetApp ONTAP

[Amazon FSx for NetApp ONTAP](#) is a fully managed AWS service that enables customers to launch and run file systems powered by the NetApp ONTAP storage operating system. FSx for ONTAP enables you to leverage NetApp features, performance, and administrative capabilities you are familiar with, while taking advantage of the simplicity, agility, security, and scalability of storing data on AWS. FSx for ONTAP supports ONTAP file system features and administration APIs.

Overview

A file system is the primary resource in Amazon FSx, analogous to an ONTAP cluster on premises. Within each SVM you can create one or multiple volumes, which are data containers that store the files and folders in your file system. With Amazon FSx for NetApp ONTAP, Data ONTAP will be provided as a managed file system in the cloud. The new file system type is called **NetApp ONTAP**.

Using Astra Trident with Amazon FSx for NetApp ONTAP, you can ensure Kubernetes clusters running in Amazon Elastic Kubernetes Service (EKS) can provision block and file persistent volumes backed by ONTAP.

Amazon FSx for NetApp ONTAP uses [FabricPool](#) to manage storage tiers. It enables you to store data in a tier, based on whether the data is frequently accessed.

Considerations

- SMB volumes:
 - SMB volumes are supported using the `ontap-nas` driver only.
 - Astra Trident supports SMB volumes mounted to pods running on Windows nodes only.
- Volumes created on Amazon FSx file systems that have automatic backups enabled cannot be deleted by Trident. To delete PVCs, you need to manually delete the PV and the FSx for ONTAP volume. To prevent this issue:
 - Do not use **Quick create** to create the FSx for ONTAP file system. The quick create workflow enables automatic backups and does not provide an opt-out option.
 - When using **Standard create**, disable automatic backup. Disabling automatic backups allows Trident to successfully delete a volume without further manual intervention.

▼ **Backup and maintenance - *optional***

Daily automatic backup [Info](#)

Amazon FSx can protect your data through daily backups

Enabled

Disabled

FSx for ONTAP driver details

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

- `ontap-san`: Each PV provisioned is a LUN within its own Amazon FSx for NetApp ONTAP volume.
- `ontap-san-economy`: Each PV provisioned is a LUN with a configurable number of LUNs per Amazon FSx for NetApp ONTAP volume.
- `ontap-nas`: Each PV provisioned is a full 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, see [NAS drivers](#) and [SAN drivers](#).

Authentication

Astra Trident offers two modes of authentication.

- **Certificate-based**: Astra Trident will communicate with the SVM on your FSx file system using a certificate installed on your SVM.
- **Credential-based**: You can use the `fsxadmin` user for your file system or the `vsadmin` user configured for your SVM.



Astra 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 Astra Trident.

You can update backends to move between credential-based and certificate-based methods. However, if you attempt to provide **credentials and certificates**, backend creation will fail. To switch to a different authentication method, you must remove the existing method from the backend configuration.

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

- [ONTAP NAS authentication](#)
- [ONTAP SAN authentication](#)

Find more information

- [Amazon FSx for NetApp ONTAP documentation](#)
- [Blog post on Amazon FSx for NetApp ONTAP](#)

Integrate Amazon FSx for NetApp ONTAP

You can integrate your Amazon FSx for NetApp ONTAP file system with Astra Trident to ensure Kubernetes clusters running in Amazon Elastic Kubernetes Service (EKS) can provision block and file persistent volumes backed by ONTAP.

Requirements

In addition to [Astra Trident requirements](#), to integrate FSx for ONTAP with Astra 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.

- Astra Trident supports SMB volumes mounted to pods running on Windows nodes only. Refer to [Prepare to provision SMB volumes](#) for details.

ONTAP SAN and NAS driver integration



If you are configuring for SMB volumes, you must read [Prepare to provision SMB volumes](#) before creating the backend.

Steps

1. Deploy Astra Trident using one of the [deployment methods](#).
2. Collect your SVM management LIF DNS name. For example, using the AWS CLI, find the `DNSName` entry under `Endpoints` → `Management` after running the following command:

```
aws fsx describe-storage-virtual-machines --region <file system region>
```

3. Create and install certificates for [NAS backend authentication](#) or [SAN backend authentication](#).



You can log in to your file system (for example to install certificates) using SSH from anywhere that can reach your file system. Use the `fsxadmin` user, the password you configured when you created your file system, and the management DNS name from `aws fsx describe-file-systems`.

4. Create a backend file using your certificates and the DNS name of your management LIF, as shown in the sample below:

YAML

```
---
version: 1
storageDriverName: ontap-san
backendName: customBackendName
managementLIF: svm-XXXXXXXXXXXXXXXXXXXX.fs-XXXXXXXXXXXXXXXXXXXX.fsx.us-
east-2.aws.internal
svm: svm01
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

JSON

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "customBackendName",
  "managementLIF": "svm-XXXXXXXXXXXXXXXXXXXX.fs-
XXXXXXXXXXXXXXXXXXXX.fsx.us-east-2.aws.internal",
  "svm": "svm01",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

For information about creating backends, see these links:

- [Configure a backend with ONTAP NAS drivers](#)
- [Configure a backend with ONTAP SAN drivers](#)

Prepare to provision SMB volumes

You can provision SMB volumes using the `ontap-nas` driver. Before you complete [ONTAP SAN and NAS driver integration](#) complete the following steps.

Before you begin

Before you can provision SMB volumes using the `ontap-nas` driver, you must have the following.

- A Kubernetes cluster with a Linux controller node and at least one Windows worker node running Windows Server 2019. Astra Trident supports SMB volumes mounted to pods running on Windows nodes only.
- At least one Astra 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. Create SMB shares. 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 |
|-----------------------|---|------------------------|
| <code>smbShare</code> | 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 Astra Trident to create the SMB share. This parameter is required for Amazon FSx for ONTAP backends. | <code>smb-share</code> |
| <code>nasType</code> | Must set to <code>smb</code>. If null, defaults to <code>nfs</code> . | <code>smb</code> |

| Parameter | Description | Example |
|-----------------|---|-------------------------------|
| 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. | "" |

FSx for ONTAP configuration options and examples

Learn about backend configuration options for Amazon FSx for ONTAP. This section provides backend configuration examples.

Backend configuration options

See the following table for the backend configuration options:

| Parameter | Description | Example |
|-------------------|---|---|
| version | | Always 1 |
| storageDriverName | Name of the storage driver | ontap-nas, ontap-nas-economy, ontap-nas-flexgroup, ontap-san, ontap-san-economy |
| backendName | Custom name of 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 Astra 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]. | “10.0.0.1”, “[2001:1234:abcd::fefe]” |

| Parameter | Description | Example |
|------------------|--|-------------------------|
| dataLIF | <p>IP address of protocol LIF.</p> <p>ONTAP NAS drivers: We recommend specifying dataLIF. If not provided, Astra Trident fetches data LIFs 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 data LIFs. Can be changed after initial setting. Refer to Update dataLIF after initial configuration.</p> <p>ONTAP SAN drivers: Do not specify for iSCSI. Astra 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.</p> <p>Can be set to use IPv6 addresses if Astra 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].</p> | |
| autoExportPolicy | <p>Enable automatic export policy creation and updating [Boolean].</p> <p>Using the autoExportPolicy and autoExportCIDRs options, Astra Trident can manage export policies automatically.</p> | false |
| autoExportCIDRs | <p>List of CIDRs to filter Kubernetes' node IPs against when autoExportPolicy is enabled.</p> <p>Using the autoExportPolicy and autoExportCIDRs options, Astra Trident can manage export policies automatically.</p> | "["0.0.0.0/0", "::/0"]" |
| labels | Set of arbitrary JSON-formatted labels to apply on volumes | "" |

| Parameter | Description | Example |
|----------------------|--|---|
| 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. | "" |
| 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 Astra 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. | "" (not enforced by default) |
| lunsPerFlexvol | Maximum LUNs per Flexvol, must be in range [50, 200]. SAN only. | 100 |

| Parameter | Description | Example |
|------------------|--|------------------------|
| debugTraceFlags | <p>Debug flags to use when troubleshooting. Example, {"api":false, "method":true}</p> <p>Do not use debugTraceFlags unless you are troubleshooting and require a detailed log dump.</p> | null |
| nfsMountOptions | <p>Comma-separated list of NFS mount options.</p> <p>The mount options for Kubernetes-persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Astra Trident will fall back to using the mount options specified in the storage backend's configuration file.</p> <p>If no mount options are specified in the storage class or the configuration file, Astra Trident will not set any mount options on an associated persistent volume.</p> | "" |
| nasType | <p>Configure NFS or SMB volumes creation.</p> <p>Options are <code>nfs</code>, <code>smb</code>, or <code>null</code>.</p> <p>Must set to <code>smb</code> for SMB volumes. Setting to <code>null</code> defaults to NFS volumes.</p> | <code>nfs</code> |
| qtreesPerFlexvol | Maximum Qtrees per FlexVol, must be in range [50, 300] | 200 |
| smbShare | <p>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 Astra Trident to create the SMB share.</p> <p>This parameter is required for Amazon FSx for ONTAP backends.</p> | <code>smb-share</code> |

| Parameter | Description | Example |
|-----------|---|--------------------|
| useREST | <p>Boolean parameter to use ONTAP REST APIs. Tech preview</p> <p>useREST is provided as a tech preview that is recommended for test environments and not for production workloads. When set to <code>true</code>, Astra Trident will use ONTAP REST APIs to communicate with the backend.</p> <p>This feature requires ONTAP 9.11.1 and later. In addition, the ONTAP login role used must have access to the <code>ontap</code> application. This is satisfied by the pre-defined <code>vsadmin</code> and <code>cluster-admin</code> roles.</p> | <code>false</code> |

Update dataLIF after initial configuration

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

```
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 for the new data LIF to take effect.

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 |
|------------------------------|---|-------------------|
| <code>spaceAllocation</code> | Space-allocation for LUNs | <code>true</code> |
| <code>spaceReserve</code> | Space reservation mode; "none" (thin) or "volume" (thick) | <code>none</code> |
| <code>snapshotPolicy</code> | Snapshot policy to use | <code>none</code> |

| Parameter | Description | Default |
|-------------------|--|--|
| qosPolicy | <p>QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend.</p> <p>Using QoS policy groups with Astra Trident requires ONTAP 9.8 or later.</p> <p>We recommend using a non-shared QoS policy group and ensuring the policy group is applied to each constituent individually. A shared QoS policy group will enforce the ceiling for the total throughput of all workloads.</p> | "" |
| adaptiveQosPolicy | <p>Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool or backend.</p> <p>Not supported by ontap-nas-economy.</p> | "" |
| snapshotReserve | Percentage of volume reserved for snapshots "0" | If snapshotPolicy is none, else "" |
| splitOnClone | Split a clone from its parent upon creation | false |
| encryption | <p>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.</p> <p>If NAE is enabled on the backend, any volume provisioned in Astra Trident will be NAE enabled.</p> <p>For more information, refer to: How Astra Trident works with NVE and NAE.</p> | false |
| luksEncryption | <p>Enable LUKS encryption. Refer to Use Linux Unified Key Setup (LUKS).</p> <p>SAN only.</p> | "" |
| tieringPolicy | Tiering policy to use none | snapshot-only for pre-ONTAP 9.5 SVM-DR configuration |

| Parameter | Description | Default |
|-----------------|---|--|
| unixPermissions | Mode for new volumes. Leave empty for SMB volumes. | "" |
| securityStyle | Security style for new volumes. NFS supports <code>mixed</code> and <code>unix</code> security styles. SMB supports <code>mixed</code> and <code>ntfs</code> security styles. | NFS default is <code>unix</code> . SMB default is <code>ntfs</code> . |

Example

Using `nasType`, `node-stage-secret-name`, and `node-stage-secret-namespace`, you can specify an SMB volume and provide the required Active Directory credentials. SMB volumes are supported using the `ontap-nas` driver only.

```

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

```

Create backends with kubectl

A backend defines the relationship between Astra Trident and a storage system. It tells Astra Trident how to communicate with that storage system and how Astra Trident should provision volumes from it. After Astra 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 Astra 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 Astra 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 Astra 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 Astra 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, see 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). Astra 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 Astra 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: t@Ax@7q(>
```

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 |
| Cloud Volumes Service for GCP | private_key_id | ID of the private key. Part of API key for GCP Service Account with CVS admin role |
| Cloud Volumes Service for GCP | private_key | Private key. Part of API key for GCP Service Account with CVS admin role |
| 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 <code>ontap-san</code> and <code>ontap-san-economy</code> |
| ONTAP | chapInitiatorSecret | CHAP initiator secret. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code> |
| ONTAP | chapTargetUsername | Target username. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code> |
| ONTAP | chapTargetInitiatorSecret | CHAP target initiator secret. Required if useCHAP=true. For <code>ontap-san</code> and <code>ontap-san-economy</code> |

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 |
|-----------------------|-------------------|--------------------------------------|
| 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 Astra 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:** Astra 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 | PHASE | STATUS | STORAGE DRIVER | BACKEND NAME | DELETION POLICY | BACKEND UUID |
|-----------------------|-------|--------|----------------|-------------------|-----------------|--------------------------------------|
| backend-tbc-ontap-san | | Bound | Success | ontap-san-backend | ontap-san | 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8 |

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 Astra Trident (for example, during Astra 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 Astra 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 will ensure the backend is still present and can be managed by using `tridentctl`.

Run the following command:

```
kubectl delete tbc <tbc-name> -n trident
```

Astra 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. Astra 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 Astra 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 Astra 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 and Trident will continue to manage those volumes and snapshots until they are deleted.

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 Astra 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 Astra 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 Astra 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.

```

kubect1 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 will ensure 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, Astra 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 Astra Trident how to provision volumes.

Configure a Kubernetes `StorageClass` object

The [Kubernetes `StorageClass` object](#) identifies Astra Trident as the provisioner that is used for that class instructs Astra Trident how to provision a volume. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
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 Astra 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 Astra Trident, and Astra 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

Astra Trident provides [simple storage class definitions for specific backends](#).

Alternatively, you can edit `sample-input/storage-class-csi.yaml.template` 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 Astra Trident's synchronized storage classes, run the following command:

```
tridentctl get storageclass
```

- To view Astra 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 Astra 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 Astra Trident will continue to manage them.



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

Overview

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

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 PV and PVC, you can mount the volume in a pod.

Sample manifests

PersistentVolume sample manifest

This sample manifest shows a basic PV of 10Gi that is associated with StorageClass `basic-csi`.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaim sample manifest

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
```

Pod manifest sample

```
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
```

Create the PV and PVC

Steps

1. Create the PV.

```
kubectl create -f pv.yaml
```

2. Verify the PV status.

```
kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM
STORAGECLASS  REASON    AGE
pv-storage    4Gi       RWO           Retain          Available
7s
```

3. Create the PVC.

```
kubectl create -f pvc.yaml
```

4. Verify the PVC status.

```
kubectl get pvc
NAME          STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
pvc-storage  Bound  pv-name         2Gi       RWO
5m
```

5. Mount the volume in a pod.

```
kubectl create -f pv-pod.yaml
```



You can monitor the progress using `kubectl get pod --watch`.

6. Verify that the volume is mounted on `/my/mount/path`.

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

7. You can now delete the Pod. The Pod application will no longer exist, but the volume will remain.

```
kubectl delete pod task-pv-pod
```

Refer to [Kubernetes and Trident objects](#) for details on how storage classes interact with the `PersistentVolumeClaim` and parameters for controlling how Astra Trident provisions volumes.

Expand volumes

Astra Trident provides Kubernetes users the ability to expand their volumes after they are created. Find information about the configurations required to expand iSCSI and NFS 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

```

Astra Trident creates a Persistent Volume (PV) and associates it with this Persistent Volume Claim (PVC).

```

kubect1 get pvc
NAME          STATUS      VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
san-pvc      Bound       pvc-8a814d62-bd58-4253-b0d1-82f2885db671  1Gi
RW0          ontap-san    8s

kubect1 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, Astra Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Astra 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 Astra Trident volume:


```

kubect1 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
kubect1 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

Astra Trident supports volume expansion for NFS PVs provisioned on `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `gcp-cvs`, 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 `kubect1 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
```

Astra Trident should create a 20MiB 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 20MiB PV to 1GiB, edit the PVC and set `spec.resources.requests.storage` to 1GB:

```
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 Astra Trident volume:

```

kubect1 get pvc ontapnas20mb
NAME                STATUS      VOLUME
CAPACITY    ACCESS MODES   STORAGECLASS   AGE
ontapnas20mb  Bound        pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi
RWO                ontapnas                4m44s

kubect1 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      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

Import volumes

You can import existing storage volumes as a Kubernetes PV using `tridentctl import`.

Overview and considerations

You might import a volume into Astra 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.

- Astra 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 Astra 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, Astra 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, Astra Trident manages the PVC and renames the FlexVol and LUN on the backend. You can pass the `--no-manage` flag to import an unmanaged volume. If you use `--no-manage`, Astra 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.



This option is 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 use `tridentctl import` to import a volume.

Steps

1. Create the Persistent Volume Claim (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

```



Don't include additional parameters such as PV name or volume size. This can cause the import command to fail.

2. Use the `tridentctl import volume` command to specify the name of the Astra Trident backend containing the volume and the name that uniquely identifies the volume on the storage (for example: ONTAP FlexVol, Element Volume, Cloud Volumes Service path). The `-f` argument is required to specify the path to the PVC file.

```

tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-
file>

```

Examples

Review the following volume import examples for supported drivers.

ONTAP NAS and ONTAP NAS FlexGroup

Astra Trident supports volume import using the `ontap-nas` and `ontap-nas-flexgroup` drivers.



- The `ontap-nas-economy` driver cannot import and manage qtrees.
- 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 on the ONTAP cluster. Importing FlexVols with the `ontap-nas` driver works the same. A FlexVol 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

The following show an example of a managed volume and an unmanaged volume import.

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>
```

| PROTOCOL | NAME | BACKEND UUID | SIZE | STATE | STORAGE CLASS | MANAGED |
|----------|--|--------------------------------------|---------|--------|---------------|---------|
| file | pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | 1.0 GiB | online | standard | true |

Unmanaged volume

When using the `--no-manage` argument, Astra 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
```

| PROTOCOL | NAME | BACKEND UUID | SIZE | STATE | STORAGE CLASS | MANAGED |
|----------|--|--------------------------------------|---------|--------|---------------|---------|
| file | pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | 1.0 GiB | online | standard | false |

ONTAP SAN

Astra Trident supports volume import using the `ontap-san` driver. Volume import is not supported using the `ontap-san-economy` driver.

Astra Trident can import ONTAP SAN FlexVols that contain a single LUN. This is consistent with the `ontap-san` driver, which creates a FlexVol for each PVC and a LUN within the FlexVol. Astra Trident imports the FlexVol and associates it with the PVC definition.

ONTAP SAN examples

The following show an example of a managed volume and an unmanaged volume import.

Managed volume

For managed volumes, Astra Trident renames the FlexVol to the `pvc-<uuid>` format and the LUN within the FlexVol to `lun0`.

The following example imports the `ontap-san-managed` FlexVol 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
  
```

Element

Astra Trident supports NetApp Element software and NetApp HCI volume import using the `solidfire-san` driver.



The Element driver supports duplicate volume names. However, Astra 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.

Element example

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   |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
  
```

Google Cloud Platform

Astra Trident supports volume import using the `gcp-cvs` driver.



To import a volume backed by the NetApp Cloud Volumes Service in Google Cloud Platform, 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 export path is `10.0.0.1:/adroit-jolly-swift`, the volume path is `adroit-jolly-swift`.

Google Cloud Platform example

The following example imports a `gcp-cvs` volume on backend `gcpcvs_YEppr` with the volume path of `adroit-jolly-swift`.

```
tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-
file> -n trident
```

| PROTOCOL | NAME | BACKEND UUID | SIZE | STATE | STORAGE CLASS | MANAGED |
|----------|--|--------------------------------------|--------|--------|---------------|---------|
| | pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | e1a6e65b-299e-4568-ad05-4f0a105c888f | 93 GiB | online | gcp-storage | true |
| | | | | | file | |

Azure NetApp Files

Astra 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`.

Azure NetApp Files example

The following example imports an `azure-netapp-files` volume on backend `azurenetafiles_40517` with the volume path `importvol1`.

```
tridentctl import volume azurenetappfiles_40517 importvoll1 -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      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

Share an NFS volume across namespaces

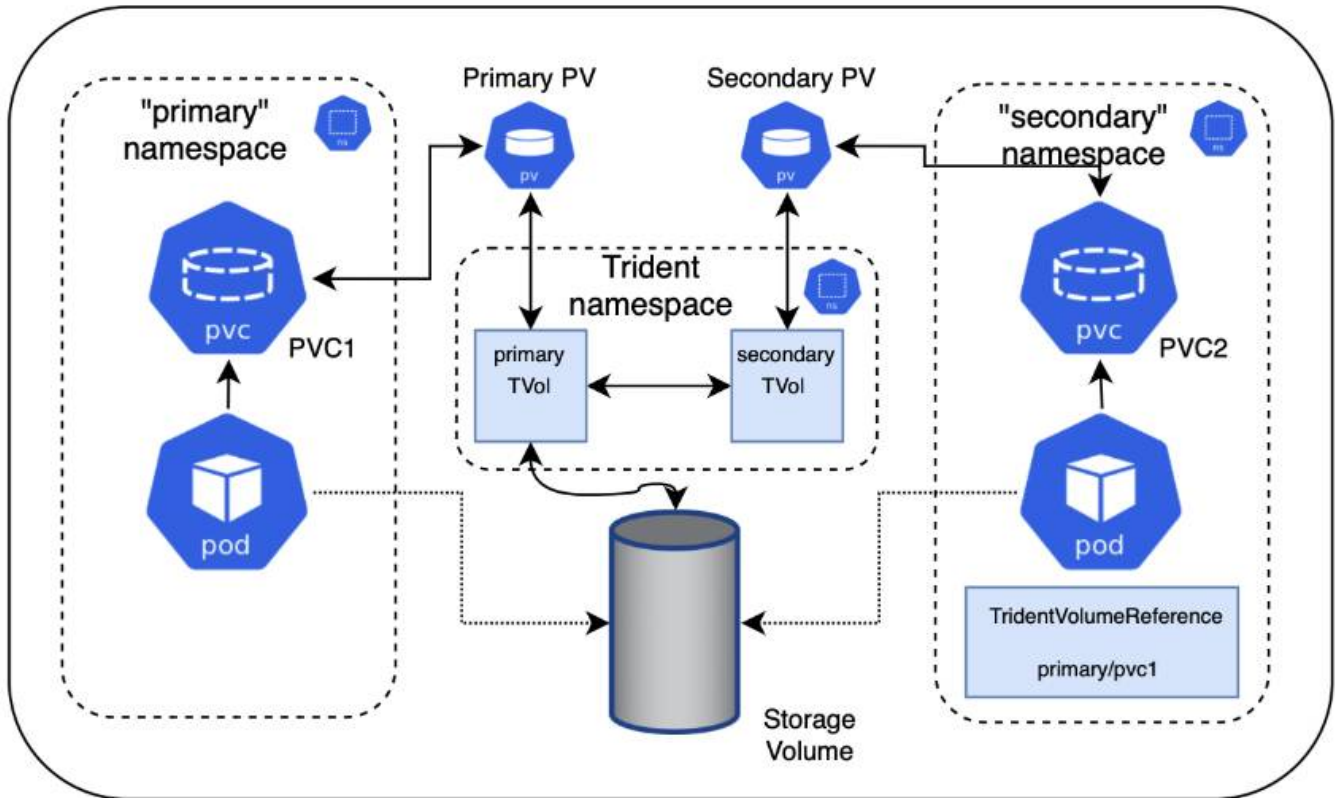
Using Astra Trident, you can create a volume in a primary namespace and share it in one or more secondary namespaces.

Features

The Astra 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
```

Astra 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:** Create the custom role and kubeconfig 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

Astra 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. Astra 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, Astra 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

- Astra 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:

- Visit [Sharing volumes between namespaces: Say hello to cross-namespace volume access](#).
- Watch the demo on [NetAppTV](#).

Use CSI Topology

Astra 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, Astra Trident uses CSI Topology.



Learn more about the CSI Topology feature [here](#).

Kubernetes provides two unique volume binding modes:

- With `VolumeBindingMode` set to `Immediate`, Astra 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:"1e11e4a2108024935ecfcb2912226cedeadfd99df",
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:"1e11e4a2108024935ecfcb2912226cedeadfd99df",
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 Astra Trident is installed for Astra 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","kubernetes.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","kubernetes.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","kubernetes.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

Astra 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 must be 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

```
---
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

```
{
  "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, Astra Trident will create 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-centrall
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-a
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-b
storage:
- labels:
    workload: production
    region: Iowa-DC
    zone: Iowa-DC-A
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-a
- labels:
    workload: dev
    region: Iowa-DC
    zone: Iowa-DC-B
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-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 will create 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:
  name: pvc-san
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 300Mi
  storageClassName: netapp-san-us-east1
```

Creating a PVC using this manifest would result in the following:

```

kubect1 create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubect1 get pvc
NAME          STATUS      VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san      Pending
2s
kubect1 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
  ----      -
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:

```

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: voll
        persistentVolumeClaim:
          claimName: pvc-san
    containers:
      - name: sec-ctx-demo
        image: busybox
        command: [ "sh", "-c", "sleep 1h" ]
        volumeMounts:
          - name: voll
            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 Astra Trident, import a snapshot created outside of Astra 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`, `gcp-cvs`, and `azure-netapp-files` 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 Astra 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: pvcl-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io

```

Import a volume snapshot

Astra Trident supports the [Kubernetes pre-provisioned snapshot process](#) to enable the cluster administrator to create a `VolumeSnapshotContent` object and import snapshots created outside of Astra Trident.

Before you begin

Astra 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 Astra 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 Astra 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>

```

- 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

```

- Internal processing (no action required):** The external snapshotter recognizes the newly created `VolumeSnapshotContent` and runs the `ListSnapshots` call. Astra Trident creates the `TridentSnapshot`.
 - The external snapshotter sets the `VolumeSnapshotContent` to `readyToUse` and the `VolumeSnapshot` to `true`.
 - Trident returns `readyToUse=true`.
- 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 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.

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 Astra 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](#)

Copyright information

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

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

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

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

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

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

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

Trademark information

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