Use Astra Trident

Astra Trident

NetApp
June 27, 2022

Table of Contents

Use Astra Trident

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 will automatically offer up storage pools from backends that together match the requirements defined by a storage class. Learn more about configuring the backend based on the type of storage system you have.

- 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

Configure an Azure NetApp Files backend

Learn about how to configure Azure NetApp Files (ANF) as the backend for your Astra Trident installation using the sample configurations provided.

> The Azure NetApp Files service does not support volumes less than 100 GB. Astra Trident automatically creates 100-GB volumes if a smaller volume is requested.

What you'll need

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

- `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 the `Owner` or `Contributor` role that is predefined by Azure.

  > To learn more about Azure built-in roles, see the Azure documentation.

- 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.
- If you are using Azure NetApp Files for the first time or in a new location, some initial configuration is required. See the quickstart guide.

About this task

Based on the backend configuration (subnet, virtual network, service level, and location), Trident creates ANF volumes on capacity pools that are available in the requested location and match the requested service level and subnet.

> NOTE: Astra Trident does not support Manual QoS capacity pools.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	"azure-netapp-files"
`backendName`	Custom name or the storage backend	Driver name + "_" + random characters
`subscriptionID`	The subscription ID from your Azure subscription	
`tenantID`	The tenant ID from an App Registration	
`clientID`	The client ID from an App Registration	
`clientSecret`	The client secret from an App Registration	
`serviceLevel`	One of `Standard`, `Premium`, or `Ultra`	"" (random)
`location`	Name of the Azure location where the new volumes will be created	
`resourceGroups`	List of resource groups for filtering discovered resources	"[]" (no filter)
`netappAccounts`	List of NetApp accounts for filtering discovered resources	"[]" (no filter)
`capacityPools`	List of capacity pools for filtering discovered resources	"[]" (no filter, random)
`virtualNetwork`	Name of a virtual network with a delegated subnet	""
`subnet`	Name of a subnet delegated to `Microsoft.Netapp/volumes`	""
`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)
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `\{"api": false, "method": true, "discovery": true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null

If you encounter a “No capacity pools found” error when attempting to create a PVC, it is likely your app registration doesn’t have the required permissions and resources (subnet, virtual network, capacity pool) associated. Astra Trident will log the Azure resources it discovered when the backend is created when debug is enabled. Be sure to check if an appropriate role is being used.

If you want to mount the volumes by using NFS version 4.1, you can include `nfsvers=4` in the comma-delimited mount options list to choose NFS v4.1. Any mount options set in a storage class override the mount options set in a backend configuration file.

The values for `resourceGroups`, `netappAccounts`, `capacityPools`, `virtualNetwork`, and `subnet` may be specified using short or fully-qualified names. Short names may match multiple resources with the same name, so using fully-qualified names is recommended in most situations. The `resourceGroups`, `netappAccounts`, and `capacityPools` values are filters which restrict the set of discovered resources to those available to this storage backend and may be specified in any combination. The fully-qualified names are of the following 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>

You can control how each volume is provisioned by default by specifying the following options in a special section of the configuration file. See the configuration examples below.

Parameter	Description	Default
`exportRule`	The export rule(s) for new volumes	"0.0.0.0/0"
`snapshotDir`	Controls visibility of the .snapshot directory	"false"
`size`	The default size of new volumes	"100G"
`unixPermissions`	The unix permissions of new volumes (4 octal digits)	"" (preview feature, requires whitelisting in subscription)

The `exportRule` value must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation.

For all the volumes created on an ANF backend, Astra Trident copies all the labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all the volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

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 ANF in the configured location, and places new volumes on one of those pools and subnets randomly.

This configuration is ideal when you are just getting started with ANF 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 ANF 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",
        "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 storage 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.

```
{
    "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"],
    "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
    "labels": {
        "cloud": "azure"
    },
    "location": "eastus",

    "storage": [
        {
            "labels": {
                "performance": "gold"
            },
            "serviceLevel": "Ultra",
            "capacityPools": ["ultra-1", "ultra-2"]
        },
        {
            "labels": {
                "performance": "silver"
            },
            "serviceLevel": "Premium",
            "capacityPools": ["premium-1"]
        },
        {
            "labels": {
                "performance": "bronze"
            },
            "serviceLevel": "Standard",
            "capacityPools": ["standard-1", "standard-2"]
        }
    ]
}
```

The following `StorageClass` definitions refer to the storage pools above. By using the `parameters.selector` field, you can specify for each `StorageClass` the visrtual 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
```

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 an Astra Data Store backend

Learn how to configure an Astra Data Store (ADS) backend for your Astra Trident installation using the sample configurations provided.

What you'll need

To configure and use the Astra Data Store backend, you need the following:

- A supported ADS storage system. See Astra Data Store preview documentation for details.
- Credentials for the Kubernetes cluster hosting ADS. The Kubernetes cluster hosting ADS must have a namespace dedicated to the volume, snapshot, and export policy resources that this Astra Trident backend will create and manage. A `kubeconfig` must be available for the Kubernetes cluster hosting ADS that supports:
  - Reading all the objects in the `astrads-system` namespace
  - Reading/writing objects in the namespace created for this Astra Trident backend's use
  - Listing all cluster namespaces

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	"astrads-nas"
`backendName`	Custom name or the storage backend	ADS cluster name
`cluster`	The name of the AstraDSCluster resource	
`namespace`	The namespace where Astra Trident will create all ADS custom resources	
`kubeconfig`	Credentials for the ADS Kubernetes cluster (Base64 compact JSON)	`nfsMountOptions`
Fine-grained control of NFS mount options	"vers=4.1"	`autoExportPolicy`
Enable automatic export policy creation and updating [Boolean]	false	`autoExportCIDRs`
List of CIDRs to filter Kubernetes' node IPs against when `autoExportPolicy` is enabled	["0.0.0.0/0", "::/0"]	`limitVolumeSize`
Fail provisioning if requested volume size is above this value	"" (not enforced by default)	`debugTraceFlags`
Debug flags to use when troubleshooting. Example: {"api":false, "method":true}	null	`labels`

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

You should convert the `kubeconfig` value from YAML to compact JSON format, and then to Base64 format before including it in the backend configuration.

Each backend provisions volumes in a single namespace on the Kubernetes cluster that hosts ADS. To create volumes in other namespaces, you can define additional backends. ADS volumes can be attached to any namespace in the hosting cluster, any other Kubernetes cluster, or anywhere else that can mount NFS shares.

You can control default provisioning for each volume using these options in a special section of the configuration file.

See the configuration examples below.

Parameter	Description	Default
`exportPolicy`	Export policy to use	"default"
`unixPermissions`	Mode for new volumes, must be octal and begin with "0"	"0777"
`snapshotReserve`	Percentage of volume reserved for snapshots	"5"
`snapshotDir`	Controls visibility of the `.snapshot` directory	"false"
`qosPolicy`	QoS policy to assign for volumes created	""

For all volumes created on an ADS backend, Astra Trident will copy all the labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all volumes created per storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration file.

Example 1: Minimal backend configuration

This is the absolute minimum backend configuration.

Example

```
{
    "version": 1,
    "storageDriverName": "astrads-nas",
    "cluster": "astrads-sti-c6220-09-10-11-12",
    "namespace": "test",
    "kubeconfig":
"eyJjdXJyZW50LWNvbnRleHQiOiJmZWRlcmFsLWNvbnRleHQiLCJhcGlWZXJzaW9uIjoidj
EiLCJjbHVzdGVycyI6W3siY2x1c3RlciI6eyJhcGktdmVyc2lvbiI6InYxIiwic2VydmVyI
joiaHR0cDovL2Nvdy5vcmc6ODA4MCJ9LCJuYW1lIjoiY293LWNsdXN0ZXIifSx7ImNsdXN0
ZXIiOnsiY2VydGlmaWNhdGUtYXV0aG9yaXR5IjoicGF0aC90by9teS9jYWZpbGUiLCJzZXJ
2ZXIiOiJodHRwczovL2hvcnNlLm9yZzo0NDQzIn0sIm5hbWUiOiJob3JzZS1jbHVzdGVyIn
0seyJjbHVzdGVyIjp7Imluc2VjdXJlLXNraXAtdGxzLXZlcmlmeSI6dHJ1ZSwic2VydmVyI
joiaHR0cHM6Ly9waWcub3JnOjQ0MyJ9LCJuYW1lIjoicGlnLWNsdXN0ZXIifV0sImNvbnRl
eHRzIjpbeyJjb250ZXh0Ijp7ImNsdXN0ZXIiOiJob3JzZS1jbHVzdGVyIiwibmFtZXNwYWN
lIjoiY2hpc2VsLW5zIiwidXNlciI6ImdyZWVuLXVzZXIifSwibmFtZSI6ImZlZGVyYWwtY2
9udGV4dCJ9LHsiY29udGV4dCI6eyJjbHVzdGVyIjoicGlnLWNsdXN0ZXIiLCJuYW1lc3BhY
2UiOiJzYXctbnMiLCJ1c2VyIjoiYmxhY2stdXNlciJ9LCJuYW1lIjoicXVlZW4tYW5uZS1j
b250ZXh0In1dLCJraW5kIjoiQ29uZmlnIiwicHJlZmVyZW5jZXMiOnsiY29sb3JzIjp0cnV
lfSwidXNlcnMiOlt7Im5hbWUiOiJibHVlLXVzZXIiLCJ1c2VyIjp7InRva2VuIjoiYmx1ZS
10b2tlbiJ9fSx7Im5hbWUiOiJncmVlbi11c2VyIiwidXNlciI6eyJjbGllbnQtY2VydGlma
WNhdGUiOiJwYXRoL3RvL215L2NsaWVudC9jZXJ0IiwiY2xpZW50LWtleSI6InBhdGgvdG8v
bXkvY2xpZW50L2tleSJ9fV19"
}
```

Example 2: Single service level configuration

This example shows a backend file that applies the same aspects to all Astra Trident-created storage.

Example

```
{
    "version": 1,
        "storageDriverName": "astrads-nas",
        "cluster": "astrads-sti-c6220-09-10-11-12",
        "namespace": "test",
        "kubeconfig":
"eyJjdXJyZW50LWNvbnRleHQiOiJmZWRlcmFsLWNvbnRleHQiLCJhcGlWZXJzaW9uIjoidj
EiLCJjbHVzdGVycyI6W3siY2x1c3RlciI6eyJhcGktdmVyc2lvbiI6InYxIiwic2VydmVyI
joiaHR0cDovL2Nvdy5vcmc6ODA4MCJ9LCJuYW1lIjoiY293LWNsdXN0ZXIifSx7ImNsdXN0
ZXIiOnsiY2VydGlmaWNhdGUtYXV0aG9yaXR5IjoicGF0aC90by9teS9jYWZpbGUiLCJzZXJ
2ZXIiOiJodHRwczovL2hvcnNlLm9yZzo0NDQzIn0sIm5hbWUiOiJob3JzZS1jbHVzdGVyIn
0seyJjbHVzdGVyIjp7Imluc2VjdXJlLXNraXAtdGxzLXZlcmlmeSI6dHJ1ZSwic2VydmVyI
joiaHR0cHM6Ly9waWcub3JnOjQ0MyJ9LCJuYW1lIjoicGlnLWNsdXN0ZXIifV0sImNvbnRl
eHRzIjpbeyJjb250ZXh0Ijp7ImNsdXN0ZXIiOiJob3JzZS1jbHVzdGVyIiwibmFtZXNwYWN
lIjoiY2hpc2VsLW5zIiwidXNlciI6ImdyZWVuLXVzZXIifSwibmFtZSI6ImZlZGVyYWwtY2
9udGV4dCJ9LHsiY29udGV4dCI6eyJjbHVzdGVyIjoicGlnLWNsdXN0ZXIiLCJuYW1lc3BhY
2UiOiJzYXctbnMiLCJ1c2VyIjoiYmxhY2stdXNlciJ9LCJuYW1lIjoicXVlZW4tYW5uZS1j
b250ZXh0In1dLCJraW5kIjoiQ29uZmlnIiwicHJlZmVyZW5jZXMiOnsiY29sb3JzIjp0cnV
lfSwidXNlcnMiOlt7Im5hbWUiOiJibHVlLXVzZXIiLCJ1c2VyIjp7InRva2VuIjoiYmx1ZS
10b2tlbiJ9fSx7Im5hbWUiOiJncmVlbi11c2VyIiwidXNlciI6eyJjbGllbnQtY2VydGlma
WNhdGUiOiJwYXRoL3RvL215L2NsaWVudC9jZXJ0IiwiY2xpZW50LWtleSI6InBhdGgvdG8v
bXkvY2xpZW50L2tleSJ9fV19",
        "defaults": {
            "exportPolicy": "myexportpolicy1",
            "qosPolicy": "bronze",
            "snapshotReserve": "10"
        },
        "labels": {"cloud": "on-prem", "creator": "ads-cluster-1",
"performance": "bronze"}
}
```

Example 3: Virtual storage pool configuration

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

Example

```
{
    "version": 1,
    "storageDriverName": "astrads-nas",
    "cluster": "astrads-sti-c6220-09-10-11-12",
    "namespace": "test",
    "kubeconfig":
"eyJjdXJyZW50LWNvbnRleHQiOiJmZWRlcmFsLWNvbnRleHQiLCJhcGlWZXJzaW9uIjoidj
EiLCJjbHVzdGVycyI6W3siY2x1c3RlciI6eyJhcGktdmVyc2lvbiI6InYxIiwic2VydmVyI
joiaHR0cDovL2Nvdy5vcmc6ODA4MCJ9LCJuYW1lIjoiY293LWNsdXN0ZXIifSx7ImNsdXN0
ZXIiOnsiY2VydGlmaWNhdGUtYXV0aG9yaXR5IjoicGF0aC90by9teS9jYWZpbGUiLCJzZXJ
2ZXIiOiJodHRwczovL2hvcnNlLm9yZzo0NDQzIn0sIm5hbWUiOiJob3JzZS1jbHVzdGVyIn
0seyJjbHVzdGVyIjp7Imluc2VjdXJlLXNraXAtdGxzLXZlcmlmeSI6dHJ1ZSwic2VydmVyI
joiaHR0cHM6Ly9waWcub3JnOjQ0MyJ9LCJuYW1lIjoicGlnLWNsdXN0ZXIifV0sImNvbnRl
eHRzIjpbeyJjb250ZXh0Ijp7ImNsdXN0ZXIiOiJob3JzZS1jbHVzdGVyIiwibmFtZXNwYWN
lIjoiY2hpc2VsLW5zIiwidXNlciI6ImdyZWVuLXVzZXIifSwibmFtZSI6ImZlZGVyYWwtY2
9udGV4dCJ9LHsiY29udGV4dCI6eyJjbHVzdGVyIjoicGlnLWNsdXN0ZXIiLCJuYW1lc3BhY
2UiOiJzYXctbnMiLCJ1c2VyIjoiYmxhY2stdXNlciJ9LCJuYW1lIjoicXVlZW4tYW5uZS1j
b250ZXh0In1dLCJraW5kIjoiQ29uZmlnIiwicHJlZmVyZW5jZXMiOnsiY29sb3JzIjp0cnV
lfSwidXNlcnMiOlt7Im5hbWUiOiJibHVlLXVzZXIiLCJ1c2VyIjp7InRva2VuIjoiYmx1ZS
10b2tlbiJ9fSx7Im5hbWUiOiJncmVlbi11c2VyIiwidXNlciI6eyJjbGllbnQtY2VydGlma
WNhdGUiOiJwYXRoL3RvL215L2NsaWVudC9jZXJ0IiwiY2xpZW50LWtleSI6InBhdGgvdG8v
bXkvY2xpZW50L2tleSJ9fV19",

    "autoExportPolicy": true,
    "autoExportCIDRs": ["10.211.55.0/24"],

    "labels": {"cloud": "on-prem", "creator": "ads-cluster-1"},
    "defaults": {"snapshotReserve": "5"},

    "storage": [
        {
            "labels": {"performance": "gold", "cost": "3"},
            "defaults": {
                "qosPolicy": "gold",
                "snapshotReserve": "10"
            }
         },
         {
            "labels": {"performance": "silver", "cost": "2"},
            "defaults": {"qosPolicy": "silver"}
         },
         {
            "labels": {"performance": "bronze", "cost": "1"},
            "defaults": {"qosPolicy": "bronze"}
```

```
            }
        ]
}
```

The following StorageClass definitions refer to the storage pools above. By using the `parameters.selector` field, 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 virtual pool.

Example

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ads
provisioner: csi.trident.netapp.io
parameters:
  backendType: astrads-nas
allowVolumeExpansion: true

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

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

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

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 CVS for GCP backend

Learn about how to configure NetApp Cloud Volumes Service (CVS) for Google Cloud Platform (GCP) as the backend for your Astra Trident installation using the sample configurations provided.

> NetApp Cloud Volumes Service for Google Cloud does not support CVS-Performance volumes less than 100 GiB in size, or CVS volumes less than 300 GiB in size. Astra Trident automatically creates volumes of the minimum size if a the volume requested is smaller than the minimum size.

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 CVS
- Project number of your Google Cloud account
- Google Cloud service account with the `netappcloudvolumes.admin` role
- API key file for your CVS service account

Astra Trident now includes support for smaller volumes with the default CVS service type on GCP. For backends created with `storageClass=software`, volumes will now have a minimum provisioning size of 300 GiB. CVS currently provides this feature under Controlled Availability and does not provide technical support. Users must sign up for access to sub-1TiB volumes here. NetApp recommends customers consume sub-1TiB volumes for **non-production** workloads.

> When deploying backends using the default CVS service type (`storageClass=software`), users must obtain access to the sub-1TiB volumes feature on GCP for the Project Number(s) and Project ID(s) in question. This is necessary for Astra Trident to provision sub-1TiB volumes. If not, volume creations will fail for PVCs that are lesser than 600 GiB. Obtain access to sub-1TiB volumes using this form.

Volumes created by Astra Trident for the default CVS service level will be provisioned as follows:

- PVCs that are smaller than 300 GiB will result in Astra Trident creating a 300 GiB CVS volume.
- PVCs that are between 300 GiB to 600 GiB will result in Astra Trident creating a CVS volume of the requested size.

* PVCs that are between 600 GiB and 1 TiB will result in Astra Trident creating a 1TiB CVS volume.
* PVCs that are greater than 1 TiB will result in Astra Trident creating a CVS volume of the requested size.

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	"gcp-cvs"
`backendName`	Custom name or the storage backend	Driver name + "_" + part of API key
`storageClass`	Type of storage. Choose from `hardware` (performance optimized) or `software` (CVS service type)	
`projectNumber`	Google Cloud account project number. The value is found on the Google Cloud portal’s Home page.	
`apiRegion`	CVS account region. It is the region where the backend will provision the volumes.	
`apiKey`	API key for the Google Cloud service account with the `netappcloudvolumes.admin` role. It includes the JSON-formatted contents of a Google Cloud service account’s private key file (copied verbatim into the backend configuration file).	
`proxyURL`	Proxy URL if proxy server required to connect to CVS Account. The proxy server can either be an HTTP proxy or an HTTPS proxy. For an HTTPS proxy, certificate validation is skipped to allow the usage of self-signed certificates in the proxy server. Proxy servers with authentication enabled are not supported.	
`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)

Parameter	Description	Default
`serviceLevel`	The CVS service level for new volumes. The values are "standard", "premium", and "extreme".	"standard"
`network`	GCP network used for CVS volumes	“default”
`debugTraceFlags`	Debug flags to use when troubleshooting. Example, `\{"api":false, "method":true}`. Do not use this unless you are troubleshooting and require a detailed log dump.	null

If using a shared VPC network, both `projectNumber` and `hostProjectNumber` must be specified. In that case, `projectNumber` is the service project, and `hostProjectNumber` is the host project.

The `apiRegion` represents the GCP region where Astra Trident creates CVS volumes. When creating cross-region Kubernetes clusters, CVS volumes created in an `apiRegion` can be used in workloads scheduled on nodes across multiple GCP regions. Be aware that cross-region traffic incurs an additional cost.

- To enable cross-region access, your StorageClass definition for `allowedTopologies` must include all regions. For example:

  ```
  - key: topology.kubernetes.io/region
    values:
    - us-east1
    - europe-west1
  ```

- `storageClass` is an optional parameter that you can use to select the desired CVS service type. You can choose from the base CVS service type (`storageClass=software`) or the CVS-Performance service type (`storageClass=hardware`), which Trident uses by default. Make sure you specify an `apiRegion` that provides the respective CVS `storageClass` in your backend definition.

Astra Trident’s integration with the base CVS service type on Google Cloud is a **beta feature**, not meant for production workloads. Trident is **fully supported** with the CVS-Performance service type and uses it by default.

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

You can control how each volume is provisioned by default by specifying the following options in a special section of the configuration file. See the configuration examples below.

Parameter	Description	Default
`exportRule`	The export rule(s) for new volumes	"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	"100Gi"

The `exportRule` value must be a comma-separated list of any combination of IPv4 addresses or IPv4 subnets in CIDR notation.

> For all the volumes created on a CVS Google Cloud backend, Trident copies all the labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all the volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Example 1: Minimal configuration

This is the absolute minimum backend configuration.

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znn
```

```
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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"
    }
}
```

Example 2: Base CVS service type configuration

This example shows a backend definition that uses the base CVS service type, which is meant for general-purpose workloads and provides light/moderate performance, coupled with high zonal availability.

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "storageClass": "software",
    "apiRegion": "us-east4",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
```

```
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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"
    }
}
```

Example 3: Single service level configuration

This example shows a backend file that applies the same aspects to all Astra Trident-created storage in the Google Cloud us-west2 region. This example also shows the usage of `proxyURL` in the backend configuration file.

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
```

```
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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"
    },
    "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"
    }
}
```

Example 4: Virtual storage pool configuration

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

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the `snapshotReserve` at 5% and the `exportRule` to 0.0.0.0/0. The virtual storage pools are defined in the `storage` section. In this example, each individual storage pool sets its own `serviceLevel`, and some pools overwrite the default values.

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\zn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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"
        }
    ]
}
```

The following StorageClass definitions refer to the storage pools above. By using the `parameters.selector` field, 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.

The first StorageClass (`cvs-extreme-extra-protection`) maps to the first virtual storage 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 storage pool is selected and ensures that the snapshot reserve requirement is met.

```
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
```

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

What you'll need

- 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.

What you need to know

The `solidfire-san` storage driver supports both volume modes: file and block. 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	No Filesystem. Raw block device.
`solidfire-san`	iSCSI	Block	RWO,ROX,RWX	No Filesystem. Raw block device.
`solidfire-san`	iSCSI	Filesystem	RWO,ROX	`xfs`, `ext3`, `ext4`
`solidfire-san`	iSCSI	Filesystem	RWO,ROX	`xfs`, `ext3`, `ext4`

Astra Trident uses CHAP when functioning as an enhanced CSI Provisioner. If you're using CHAP (which is the default for CSI), no further preparation is required. It is recommended to explicitly set the `UseCHAP` option to use CHAP with non-CSI Trident. Otherwise, see here.

Volume access groups are only supported by the conventional, non-CSI framework for Astra Trident. When configured to work in CSI mode, Astra Trident uses CHAP.

If neither `AccessGroups` or `UseCHAP` are set, one of the following rules applies:

- If the default `trident` access group is detected, access groups are used.
- If no access group is detected and Kubernetes version is 1.7 or later, then CHAP is used.

Backend configuration options

See the following table for the backend configuration options:

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

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

For all volumes created, Astra Trident will copy all labels present on a storage pool to the backing storage LUN at the time it is provisioned. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

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 storage pools

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

In the sample backend definition file shown below, specific defaults are set for all storage pools, which set the `type` at Silver. The virtual storage pools are defined in the `storage` section. In this example, some of the storage pool sets their own type, and some pools overwrite 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 storage 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 storage 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 storage pool is selected and will ensure the storage requirement is met.

```yaml
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

Configure a backend with ONTAP or Cloud Volumes ONTAP SAN drivers

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

- Preparation
- Configuration and examples

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.

Preparation

Learn about how to prepare to configure an ONTAP backend with ONTAP SAN drivers. 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 of your Kubernetes worker nodes must have the appropriate iSCSI tools installed. See here for more details.

Authentication

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).

Users can also choose to update existing backends, opting to move from credential-based to certificate-based, and vice-versa. If **both credentials and certificates are provided**, Astra Trident will default to using certificates while issuing a warning to remove the credentials from the backend definition.

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:

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret",
}
```

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=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",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"trustedCACertificate": "QNFinfO...SiqOyN",
"storagePrefix": "myPrefix_"
}

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

Update authentication methods or rotate credentials

You can update an existing backend to make use of 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, use an 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",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "secret",
"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.

Specify igroups

Astra Trident uses igroups to control access to the volumes (LUNs) that it provisions. Administrators have two options when it comes to specifying igroups for backends:

- Astra Trident can automatically create and manage an igroup per backend. If `igroupName` is not included in the backend definition, Astra Trident creates an igroup named `trident-<backend-UUID>` on the SVM. This will ensure each backend has a dedicated igroup and handle the automated addition/deletion of Kubernetes node IQNs.
- Alternatively, pre-created igroups can also be provided in a backend definition. This can be done using the `igroupName` config parameter. Astra Trident will add/delete Kubernetes node IQNs to the pre-existing

igroup.

For backends that have `igroupName` defined, the `igroupName` can be deleted with a `tridentctl backend update` to have Astra Trident auto-handle igroups. This will not disrupt access to volumes that are already attached to workloads. Future connections will be handled using the igroup Astra Trident created.

> Dedicating an igroup for each unique instance of Astra Trident is a best practice that is beneficial for the Kubernetes admin as well as the storage admin. CSI Trident automates the addition and removal of cluster node IQNs to the igroup, greatly simplifying its management. When using the same SVM across Kubernetes environments (and Astra Trident installations), using a dedicated igroup ensures that changes made to one Kubernetes cluster don't influence igroups associated with another. In addition, it is also important to ensure each node in the Kubernetes cluster has a unique IQN. As mentioned above, Astra Trident automatically handles the addition and removal of IQNs. Reusing IQNs across hosts can lead to undesirable scenarios where hosts get mistaken for one another and access to LUNs is denied.

If Astra Trident is configured to function as a CSI Provisioner, Kubernetes node IQNs are automatically added to/removed from the igroup. When nodes are added to a Kubernetes cluster, `trident-csi` DaemonSet deploys a pod (`trident-csi-xxxxx`) on the newly added nodes and registers the new nodes it can attach volumes to. Node IQNs are also added to the backend's igroup. A similar set of steps handle the removal of IQNs when node(s) are cordoned, drained, and deleted from Kubernetes.

If Astra Trident does not run as a CSI Provisioner, the igroup must be manually updated to contain the iSCSI IQNs from every worker node in the Kubernetes cluster. IQNs of nodes that join the Kubernetes cluster will need to be added to the igroup. Similarly, IQNs of nodes that are removed from the Kubernetes cluster must be removed from the igroup.

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": "FaKePaSsWoRd",
    "igroupName": "trident",
    "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).
- Managing the addition of inititators to the `igroupName` given in the backend. If unspecified, this defaults to `trident`.

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": "FaKePaSsWoRd",
    "igroupName": "trident",
    "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.

Configuration options and examples

Learn about how to create and use ONTAP SAN drivers with your Astra Trident installation. This section provides backend configuration examples and details about how to map 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 or the storage backend	Driver name + “_” + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF	“10.0.0.1”, “[2001:1234:abcd::fefe]”
`dataLIF`	IP address of protocol LIF. Use square brackets for IPv6. Cannot be updated after you set it	Derived by the SVM unless specified
`useCHAP`	Use CHAP to authenticate iSCSI for ONTAP SAN drivers [Boolean]	false
`chapInitiatorSecret`	CHAP initiator secret. Required if `useCHAP=true`	“”
`labels`	Set of arbitrary JSON-formatted labels to apply on volumes	“”
`chapTargetInitiatorSecret`	CHAP target initiator secret. Required if `useCHAP=true`	“”
`chapUsername`	Inbound username. Required if `useCHAP=true`	“”
`chapTargetUsername`	Target username. Required if `useCHAP=true`	“”
`clientCertificate`	Base64-encoded value of client certificate. Used for certificate-based auth	“”
`clientPrivateKey`	Base64-encoded value of client private key. Used for certificate-based auth	“”
`trustedCACertificate`	Base64-encoded value of trusted CA certificate. Optional. Used for certificate-based 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	“”
`svm`	Storage virtual machine to use	Derived if an SVM `managementLIF` is specified
`igroupName`	Name of the igroup for SAN volumes to use	“trident-<backend-UUID>”

Parameter	Description	Default
`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 for the economy driver.	"" (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}	null
`useREST`	Boolean parameter to use ONTAP REST APIs. **Tech preview**	false

`useREST` is provided as a **tech preview** that is recommended for test environments and not for production workloads. When set to `true`, Astra Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.9 and later. In addition, the ONTAP login role used must have access to the `ontap` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles.

To communicate with the ONTAP cluster, you should provide the authentication parameters. This could be the username/password to a security login or an installed certificate.

If you are using an Amazon FSx for NetApp ONTAP backend, do not specify the `limitAggregateUsage` parameter. The `fsxadmin` and `vsadmin` roles provided by Amazon FSx for NetApp ONTAP do not contain the required access permissions to retrieve aggregate usage and limit it through Astra Trident.

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

For the `ontap-san` drivers, the default is to use all data LIF IPs from the SVM and to use iSCSI multipath. Specifying an IP address for the dataLIF for the `ontap-san` drivers forces them to disable multipath and use only the specified address.

When creating a backend, remember that `dataLIF` and `storagePrefix` cannot be modified after creation. To update these parameters, you will need to create a new backend.

`igroupName` can be set to an igroup that is already created on the ONTAP cluster. If unspecified, Astra Trident automatically creates an igroup named trident-<backend-UUID>. If providing a pre-defined igroupName, NetApp recommends using an igroup per Kubernetes cluster, if the SVM is to be shared between environments. This is necessary for Astra Trident to maintain IQN additions/deletions automatically.

Backends can also have igroups updated after creation:

- igroupName can be updated to point to a new igroup that is created and managed on the SVM outside of Astra Trident.
- igroupName can be omitted. In this case, Astra Trident will create and manage a trident-<backend-UUID> igroup automatically.

In both cases, volume attachments will continue to be accessible. Future volume attachments will use the updated igroup. This update does not disrupt access to volumes present on the backend.

A fully-qualified domain name (FQDN) can be specified for the `managementLIF` option.

`managementLIF` for all ONTAP drivers can also be set to IPv6 addresses. Make sure to install Trident with the `--use-ipv6` flag. Care must be taken to define `managementLIF` IPv6 address within square brackets.

> When using IPv6 addresses, make sure `managementLIF` and `dataLIF` (if included in your backend definition) are defined within square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]. If `dataLIF` is not provided, Astra Trident will fetch the IPv6 data LIFs from the SVM.

To enable the ontap-san drivers to use CHAP, set the `useCHAP` parameter to `true` in your backend definition. Astra Trident will then configure and use bidirectional CHAP as the default authentication for the SVM given in the backend. See here to learn about how it works.

For the `ontap-san-economy` driver, the `limitVolumeSize` option will also restrict the maximum size of the volumes it manages for qtrees and LUNs.

> Astra Trident sets provisioning labels in the "Comments" field of all volumes created using the `ontap-san` driver. For each volume created, the "Comments" field on the FlexVol will be populated with all labels present on the storage pool it is placed in. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Backend configuration options for provisioning volumes

You can control how each volume is provisioned by default using these options in a special 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`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""

Parameter	Description	Default
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	“”
`snapshotReserve`	Percentage of volume reserved for snapshots “0”	If `snapshotPolicy` is “none”, else “”
`splitOnClone`	Split a clone from its parent upon creation	“false”
`splitOnClone`	Split a clone from its parent upon creation	“false”
`encryption`	Enable NetApp volume encryption	“false”
`securityStyle`	Security style for new volumes	“unix”
`tieringPolicy`	Tiering policy to use “none”	“snapshot-only” for pre-ONTAP 9.5 SVM-DR configuration

> 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.

Here’s an example with defaults defined:

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "managementLIF": "10.0.0.1",
 "dataLIF": "10.0.0.2",
 "svm": "trident_svm",
 "username": "admin",
 "password": "password",
 "labels": {"k8scluster": "dev2", "backend": "dev2-sanbackend"},
 "storagePrefix": "alternate-trident",
 "igroupName": "custom",
 "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:

```
Total volume size = [(PVC requested size) / (1 - (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, the recommendation is to specify DNS names for LIFs instead of IP addresses.

`ontap-san` driver with certificate-based authentication

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,
    "storageDriverName": "ontap-san",
    "backendName": "DefaultSANBackend",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
    "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
    "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

`ontap-san` **driver with bidirectional CHAP**

This is a minimal backend configuration example. This basic configuration creates an `ontap-san` backend with `useCHAP` set to `true`.

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "labels": {"k8scluster": "test-cluster-1", "backend": "testcluster1-
sanbackend"},
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

`ontap-san-economy` **driver**

```
{
    "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",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

Examples of backends with virtual storage pools

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

In this example, some of the storage pool sets their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools overwrite the default values set above.

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret",

    "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"
            }
        }
    ]
}
```

Here is an iSCSI example for the `ontap-san-economy` driver:

```
{
    "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",
    "igroupName": "trident",
```

```
    "username": "vsadmin",
    "password": "secret",

    "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"
            }
        }
    ]
}
```

Map backends to StorageClasses

The following StorageClass definitions refer to the above virtual storage 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 (`protection-gold`) will map to the first, second virtual storage pool in the `ontap-nas-flexgroup` backend and the first virtual storage pool in the `ontap-san` backend. These are the only pool offering gold level protection.
- The second StorageClass (`protection-not-gold`) will map to the third, fourth virtual storage pool in

`ontap-nas-flexgroup` backend and the second, third virtual storage pool in `ontap-san` backend. These are the only pools offering protection level other than gold.

- The third StorageClass (`app-mysqldb`) will map to the fourth virtual storage pool in `ontap-nas` backend and the third virtual storage pool in `ontap-san-economy` backend. These are the only pools offering storage pool configuration for mysqldb type app.
- The fourth StorageClass (`protection-silver-creditpoints-20k`) will map to the third virtual storage pool in `ontap-nas-flexgroup` backend and the second virtual storage pool in `ontap-san` backend. These are the only pools offering gold-level protection at 20000 creditpoints.
- The fifth StorageClass (`creditpoints-5k`) will map to the second virtual storage pool in `ontap-nas-economy` backend and the third virtual storage pool in `ontap-san` backend. These are the only pool offerings at 5000 creditpoints.

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
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"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

Configure a backend with ONTAP NAS drivers

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

- Preparation
- Configuration and examples

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.

Preparation

Learn about how to prepare to configure an ONTAP backend with ONTAP NAS drivers. For all ONTAP backends, Astra Trident requires at least one aggregate assigned to the SVM.

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 Gold class that uses the `ontap-nas` driver and a Bronze class that uses the `ontap-nas-economy` one.

All of your Kubernetes worker nodes must have the appropriate NFS tools installed. See here for more details.

Authentication

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).

Users can also choose to update existing backends, opting to move from credential-based to certificate-based, and vice-versa. If **both credentials and certificates are provided**, Astra Trident will default to using certificates while issuing a warning to remove the credentials from the backend definition.

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:

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "secret"
}
```

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": "Faaaakkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"storagePrefix": "myPrefix_"
}

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

Update authentication methods or rotate credentials

You can update an existing backend to make use of 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, use an updated `backend.json` file containing the required parameters to execute `tridentctl backend update`.

```
$ 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": "secret",
"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

The 20.04 release of CSI 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 finegrained and automated managment.

> The dynamic management of export policies is only available for CSI Trident. It is important to ensure that the worker nodes are not being NATed.

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": "FaKePaSsWoRd",
    "autoExportCIDRs": ["192.168.0.0/24"],
    "autoExportPolicy": true
}
```

> When using this feature, you must ensure that the root junction in your SVM has a precreated export policy with an export rule that permits the node CIDR block (such as the default export policy). Always follow NetApp's recommended best practice of dedicating a 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", "::/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>
      fileSystemType: 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.

Configuration options and examples

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

Backend configuration options

See the following table for the backend configuration options:

Parameter	Description	Default
`version`		Always 1
`storageDriverName`	Name of the storage driver	“ontap-nas”, “ontap-nas-economy”, “ontap-nas-flexgroup”, “ontap-san”, “ontap-san-economy”
`backendName`	Custom name or the storage backend	Driver name + “_” + dataLIF
`managementLIF`	IP address of a cluster or SVM management LIF	“10.0.0.1”, “[2001:1234:abcd::fefe]”
`dataLIF`	IP address of protocol LIF. Use square brackets for IPv6. Cannot be updated after you set it	Derived by the SVM unless specified
`autoExportPolicy`	Enable automatic export policy creation and updating [Boolean]	false
`autoExportCIDRs`	List of CIDRs to filter Kubernetes’ node IPs against when `autoExportPolicy` is enabled	[“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	

Parameter	Description	Default
`svm`	Storage virtual machine to use	Derived if an SVM `managementLIF` is specified
`igroupName`	Name of the igroup for SAN volumes to use	“trident-<backend-UUID>”
`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 for the economy driver.	“” (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}	null
`nfsMountOptions`	Comma-separated list of NFS mount options	“”
`qtreesPerFlexvol`	Maximum Qtrees per FlexVol, must be in range [50, 300]	“200”
`useREST`	Boolean parameter to use ONTAP REST APIs. **Tech preview**	false

`useREST` is provided as a **tech preview** that is recommended for test environments and not for production workloads. When set to `true`, Astra Trident will use ONTAP REST APIs to communicate with the backend. This feature requires ONTAP 9.9 and later. In addition, the ONTAP login role used must have access to the `ontap` application. This is satisfied by the pre-defined `vsadmin` and `cluster-admin` roles.

To communicate with the ONTAP cluster, you should provide the authentication parameters. This could be the username/password to a security login or an installed certificate.

If you are using an Amazon FSx for NetApp ONTAP backend, do not specify the `limitAggregateUsage` parameter. The `fsxadmin` and `vsadmin` roles provided by Amazon FSx for NetApp ONTAP do not contain the required access permissions to retrieve aggregate usage and limit it through Astra Trident.

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

When creating a backend, remember that the `dataLIF` and `storagePrefix` cannot be modified after creation. To update these parameters, you will need to create a new backend.

A fully-qualified domain name (FQDN) can be specified for the `managementLIF` option. A FQDN may also be specified for the `dataLIF` option, in which case the FQDN will be used for the NFS mount operations. This way you can create a round-robin DNS to load-balance across multiple data LIFs.

`managementLIF` for all ONTAP drivers can also be set to IPv6 addresses. Make sure to install Astra Trident with the `--use-ipv6` flag. Care must be taken to define the `managementLIF` IPv6 address within square brackets.

> When using IPv6 addresses, make sure `managementLIF` and `dataLIF` (if included in your backend definition) are defined within square brackets, such as [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]. If `dataLIF` is not provided, Astra Trident will fetch the IPv6 data LIFs from the SVM.

Using the `autoExportPolicy` and `autoExportCIDRs` options, CSI Trident can manage export policies automatically. This is supported for all ontap-nas-* drivers.

For the `ontap-nas-economy` driver, the `limitVolumeSize` option will also restrict 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.

The `nfsMountOptions` parameter can be used to specify mount options. The mount options for Kubernetes persistent volumes are normally specified in storage classes, but if no mount options are specified in a storage class, Astra Trident will fall back to using the mount options specified in the storage backend's configuration file. If no mount options are specified in either the storage class or the configuration file, then Astra Trident will not set any mount options on an associated persistent volume.

> Astra Trident sets provisioning labels in the "Comments" field of all volumes created using `ontap-nas` and `ontap-nas-flexgroup`. Based on the driver used, the comments are set on the FlexVol (`ontap-nas`) or FlexGroup (`ontap-nas-flexgroup`). Astra Trident will copy all labels present on a storage pool to the storage volume at the time it is provisioned. Storage administrators can define labels per storage pool and group all volumes created in a storage pool. This provides a convenient way of differentiating volumes based on a set of customizable labels that are provided in the backend configuration.

Backend configuration options for provisioning volumes

You can control how each volume is provisioned by default using these options in a special 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`	QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend	""

Parameter	Description	Default
`adaptiveQosPolicy`	Adaptive QoS policy group to assign for volumes created. Choose one of qosPolicy or adaptiveQosPolicy per storage pool/backend. Not supported by ontap-nas-economy.	“”
`snapshotReserve`	Percentage of volume reserved for snapshots “0”	If `snapshotPolicy` is “none”, else “”
`splitOnClone`	Split a clone from its parent upon creation	“false”
`encryption`	Enable NetApp volume encryption	“false”
`securityStyle`	Security style for new volumes	“unix”
`tieringPolicy`	Tiering policy to use “none”	“snapshot-only” for pre-ONTAP 9.5 SVM-DR configuration
unixPermissions	Mode for new volumes	“777”
snapshotDir	Controls visibility of the `.snapshot` directory	“false”
exportPolicy	Export policy to use	“default”
securityStyle	Security style for new volumes	“unix”

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.

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 = 10GiB 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:

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` driver with certificate-based authentication

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_"
}
```

`ontap-nas` driver with auto export policy

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": "secret",
    "nfsMountOptions": "nfsvers=4",
}
```

`ontap-nas-flexgroup` **driver**

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "labels": {"k8scluster": "test-cluster-east-1b", "backend": "test1-ontap-cluster"},
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
}
```

`ontap-nas` **driver with IPv6**

```
{
 "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": "netapp123"
}
```

`ontap-nas-economy` **driver**

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret"
}
```

Examples of backends with virtual storage pools

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

In this example, some of the storage pool sets their own `spaceReserve`, `spaceAllocation`, and `encryption` values, and some pools overwrite the default values set above.

`ontap-nas` **driver**

```
{
    {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "admin",
    "password": "secret",
    "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":{"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` **driver**

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

```
    "password": "secret",

    "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` **driver**

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

    "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":{"department":"legal", "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 the above virtual storage 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 (`protection-gold`) will map to the first, second virtual storage pool in the `ontap-nas-flexgroup` backend and the first virtual storage pool in the `ontap-san` backend. These are the only pool offering gold level protection.
- The second StorageClass (`protection-not-gold`) will map to the third, fourth virtual storage pool in `ontap-nas-flexgroup` backend and the second, third virtual storage pool in `ontap-san` backend. These are the only pools offering protection level other than gold.
- The third StorageClass (`app-mysqldb`) will map to the fourth virtual storage pool in `ontap-nas` backend and the third virtual storage pool in `ontap-san-economy` backend. These are the only pools offering storage pool configuration for mysqldb type app.
- The fourth StorageClass (`protection-silver-creditpoints-20k`) will map to the third virtual storage pool in `ontap-nas-flexgroup` backend and the second virtual storage pool in `ontap-san` backend. These are the only pools offering gold-level protection at 20000 creditpoints.
- The fifth StorageClass (`creditpoints-5k`) will map to the second virtual storage pool in `ontap-nas-economy` backend and the third virtual storage pool in `ontap-san` backend. These are the only pool offerings at 5000 creditpoints.

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

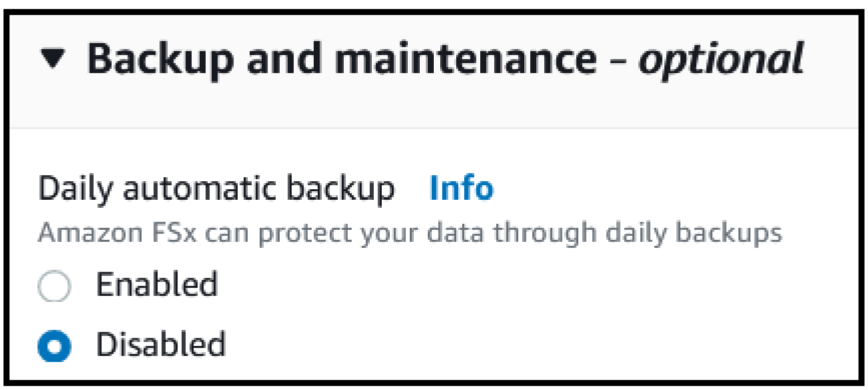
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
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"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

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 NetApp's ONTAP storage operating system. Amazon FSx for NetApp 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 supports many of ONTAP's file system features and administration APIs.

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.

> **Creating your Amazon FSx for ONTAP file system**
>
> Volumes created on Amazon FSx filesystems 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.
>
> 
>

Learn about Astra Trident

If you are new to Astra Trident, familiarize yourself by using the links provided below:

- FAQs
- Requirements for using Astra Trident
- Deploy Astra Trident
- Best practices for configuring ONTAP, Cloud Volumes ONTAP, and Amazon FSx for NetApp ONTAP
- Integrate Astra Trident
- ONTAP SAN backend configuration

* ONTAP NAS backend configuration

Learn more about driver capabilities here.

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.

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. It is not recommended to use the `fsxadmin` user, with Trident, as a `vsadmin` SVM user has access to more Astra Trident capabilities.

Drivers

You can integrate Astra Trident with Amazon FSx for NetApp ONTAP by 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.

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.

> We strongly recommend using the `vsadmin` user instead of the `fsxadmin` to configure your backend. Astra Trident will communicate with the FSx file system using this username and password.

To learn more about authentication, see these links:

* ONTAP NAS
* ONTAP SAN

Deploy and configure Astra Trident on EKS with Amazon FSx for NetApp ONTAP

What you'll 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 and/or iSCSI.

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

For other Astra Trident requirements, see here.

Steps

1. Deploy Astra Trident using one of the ../trident-get-started/kubernetes-deploy.html[deployment methods^].
2. Configure Astra Trident as follows:
   a. Collect your SVM's management LIF DNS name. For example, by 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 authentication. If you are using an `ontap-san` backend, see here. If you are using an `ontap-nas` backend, see here.

> 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:

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "customBackendName",
  "managementLIF": "svm-XXXXXXXXXXXXXXXXX.fs-XXXXXXXXXXXXXXXXX.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

Do not specify `dataLIF` for the `ontap-san` and `ontap-san-economy` drivers to allow Astra Trident to use multipath.

The `limitAggregateUsage` parameter will not work with the `vsadmin` and `fsxadmin` user accounts. The configuration operation will fail if you specify this parameter.

After deployment, perform the steps to create a storage class, provision a volume, and mount the volume in a pod.

Find more information

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

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:

```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
```

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

Storage platform Secret Fields description	Secret	Fields description
ONTAP	username	Username to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	password	Password to connect to the cluster/SVM. Used for credential-based authentication
ONTAP	clientPrivateKey	Base64-encoded value of client private key. Used for certificate-based authentication
ONTAP	chapUsername	Inbound username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetUsername	Target username. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP target initiator secret. Required if useCHAP=true. For `ontap-san` and `ontap-san-economy`

The Secret created in this step will be referenced in the `spec.credentials` field of the `TridentBackendConfig` object that is created in the next step.

Step 2: Create the `TridentBackendConfig` CR

You are now ready to create your `TridentBackendConfig` CR. In this example, a backend that uses the `ontap-san` driver is created by using the `TridentBackendConfig` object shown below:

```
$ kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

Step 3: Verify the status of the `TridentBackendConfig` CR

Now that you created the `TridentBackendConfig` CR, you can verify the status. See the following example:

```
$ kubectl -n trident get tbc backend-tbc-ontap-san
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

A backend was successfully created and bound to the `TridentBackendConfig` CR.

Phase can take one of the following values:

- `Bound`: The `TridentBackendConfig` CR is associated with a backend, and that backend contains `configRef` set to the `TridentBackendConfig` CR's uid.
- `Unbound`: Represented using `""`. The `TridentBackendConfig` object is not bound to a backend. All newly created `TridentBackendConfig` CRs are in this phase by default. After the phase changes, it cannot revert to Unbound again.
- `Deleting`: The `TridentBackendConfig` CR's `deletionPolicy` was set to delete. When the `TridentBackendConfig` CR is deleted, it transitions to the Deleting state.
    - If no persistent volume claims (PVCs) exist on the backend, deleting the `TridentBackendConfig` will result in 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 the user's intervention.

At this stage, a backend is successfully created! There are several operations that can additionally be handled, such as backend updates and backend deletions.

(Optional) Step 4: Get more details

You can run the following command to get more information about your backend:

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

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

In addition, you can also obtain a YAML/JSON dump of `TridentBackendConfig`.

```
$ kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```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.

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.
  In this case, `TridentBackendConfig` objects can be updated directly through Kubernetes.

```
$ kubectl apply -f <updated-backend-file.yaml>
```

Alternatively, make changes to the existing `TridentBackendConfig` CR by running 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. 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's 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 Storage Pools (if present) must retain the same order as in the original backend.
- Credentials are provided through a Kubernetes Secret and not in plain text.

In this case, the `TridentBackendConfig` will look like this:

```
$ cat backend-tbc-ontap-nas.yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'

$ kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

Step 3: Verify the status of the `TridentBackendConfig` CR

After the `TridentBackendConfig` has been created, its phase must be `Bound`. It should also reflect the same backend name and UUID as that of the existing backend.

```
$ kubectl -n trident 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.

Manage storage classes

Find information about creating a storage class, deleting a storage class, and viewing existing storage classes.

Design a storage class

See here for more information on what storage classes are and how you configure them.

Create a storage class

After you have a storage class file, run the following command:

```
kubectl create -f <storage-class-file>
```

`<storage-class-file>` should be replaced with your storage class file name.

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 esixting StorageClasses and re-create them with `parameters.fsType` specified.

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.

> You should only have one default storage class in your cluster at any given 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}]'
```

Perform volume operations

Learn about the features Astra Trident provides for managing your volumes.

* Use CSI Topology
* Work with snapshots
* Expand volumes
* Import volumes

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. 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 1.17 or later.

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- Nodes in the cluster should have labels that introduce topology awareness (`topology.kubernetes.io/region` and `topology.kubernetes.io/zone`). These labels **should be present on nodes in the cluster** before 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","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

Step 1: Create a topology-aware backend

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 what an example backend definition looks like:

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "xxxxxxxxxxxx",
 "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-central1",
"managementLIF": "172.16.238.5",
"svm": "nfs_svm",
"username": "admin",
"password": "Netapp123",
"supportedTopologies": [
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"},
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
    ]
"storage": [
   {
       "labels": {"workload":"production"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
         "region": "Iowa-DC",
         "zone": "Iowa-DC-B",
         "supportedTopologies": [
             {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
         ]
     }
]
}
```

In this example, the `region` and `zone` labels stand for the location of the storage pool. `topology.kubernetes.io/region` and `topology.kubernetes.io/zone` dictate where the storage pools can be consumed from.

Step 2: Define StorageClasses that are topology aware

Based on the topology labels that are provided to the nodes in the cluster, StorageClasses can be defined to contain topology information. This will determine the storage pools that serve as candidates for PVC requests made, and the subset of nodes that can make use of the volumes provisioned by Trident.

See the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
- matchLabelExpressions:
- key: topology.kubernetes.io/zone
  values:
  - us-east1-a
  - us-east1-b
- key: topology.kubernetes.io/region
  values:
  - us-east1
parameters:
  fsType: "ext4"
```

In the StorageClass definition provided above, `volumeBindingMode` is set to `WaitForFirstConsumer`. PVCs that are requested with this StorageClass will not be acted upon until they are referenced in a pod. And, `allowedTopologies` provides the zones and region to be used. The `netapp-san-us-east1` StorageClass 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:

```
$ kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
$ kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                     netapp-san-us-east1
2s
$ kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

For Trident to create a volume and bind it to the PVC, use the PVC in a pod. See the following example:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

This podSpec instructs Kubernetes to schedule the pod on nodes that are present in the `us-east1` region, and choose from any node that is present in the `us-east1-a` or `us-east1-b` zones.

See the following output:

```
$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
$ kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

Update backends to include `supportedTopologies`

Pre-existing backends can be updated to include a list of `supportedTopologies` using `tridentctl backend update`. This will not affect volumes that have already been provisioned, and will only be used for subsequent PVCs.

Find more information

- Manage resources for containers
- nodeSelector
- Affinity and anti-affinity
- Taints and Tolerations

Work with snapshots

Beginning with the 20.01 release of Astra Trident, you can create snapshots of PVs at the Kubernetes layer. You can use these snapshots to maintain point-in-time copies of volumes that have been created by Astra Trident and schedule the creation of additional volumes (clones). Volume snapshot is supported by the `ontap-nas`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `gcp-cvs`, and `azure-netapp-files` drivers.

> ⓘ This feature is available from Kubernetes 1.17 (beta) and is GA from 1.20. To understand the changes involved in moving from beta to GA, see the release blog. With the graduation to GA, the `v1` API version is introduced and is backward compatible with `v1beta1` snapshots.

What you'll need

- Creating volume snapshots requires creating an external snapshot controller and Custom Resource Definitions (CRDs). This is the responsibility of the Kubernetes orchestrator being used (for example: Kubeadm, GKE, OpenShift).

If your Kubernetes distribution does not include the snapshot controller and CRDs, you can deploy them as follows.

1. Create volume snapshot CRDs.

   For Kubernetes 1.20 and above, use v1 snapshot CRDs with snapshot components of v5.0 or above. For Kubernetes versions 1.17-1.19, use v1beta1 with v3.0.3 snapshot components.

v5.0 components

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.
yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents
.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
5.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

v3.0.3 components

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-
snapshotter/v3.0.3/client/config/crd/snapshot.storage.k8s.io_volumes
napshots.yaml
```

2. Create the snapshot controller in the desired namespace. Edit the YAML manifests below to modify namespace.

   For Kubernetes 1.20 and above use v5.0 or above. For Kubernetes versions 1.17-1.19 use v3.0.3

   > Don’t create a snapshot controller if setting up on-demand volume snapshots in a GKE environment. GKE uses a built-in, hidden snapshot-controller.

v5.0 controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-5.0/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

v3.0.3 controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/v3.0.3/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

CSI Snapshotter provides a validating webhook to help users validate existing v1beta1 snapshots and confirm they are valid resource objects. The validating webhook automatically labels invalid snapshot objects and prevents the creation of future invalid objects. The validating webhook is deployed by the Kubernetes orchestrator. See the instructions to deploy the validating webhook manually here. Find examples of invalid snapshot manifests here.

The example detailed below explains the constructs required for working with snapshots and shows how snapshots can be created and used.

Step 1: Set up a `VolumeSnapshotClass`

Before creating a volume snapshot, set up a `VolumeSnapshotClass`.

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

The `driver` points to Astra Trident's 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.

Step 2: Create a snapshot of an existing PVC

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

The snapshot is being created for a PVC named `pvc1`, and the name of the snapshot is set to `pvc1-snap`.

```
$ kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

$ kubectl get volumesnapshots
NAME                   AGE
pvc1-snap              50s
```

This created a `VolumeSnapshot` object. A VolumeSnapshot is analogous to a PVC and is associated with a `VolumeSnapshotContent` object that represents the actual snapshot.

It is possible to identify the `VolumeSnapshotContent` object for the `pvc1-snap` VolumeSnapshot by describing it.

```
$ 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
.
.
```

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.

Step 3: Create PVCs from VolumeSnapshots

See the following example for creating a PVC using a snapshot:

```
$ cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

`dataSource` shows that the PVC must be created using a VolumeSnapshot named `pvc1-snap` as the source of the data. This instructs Astra Trident to create a PVC from the snapshot. After the PVC is created, it can be attached to a pod and used just like any other PVC.

> When deleting a Persistent Volume with associated snapshots, the corresponding Trident volume is updated to a "Deleting state". For the Astra Trident volume to be deleted, the snapshots of the volume should be removed.

Find more information

- Volume snapshots
- `VolumeSnapshotClass`

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.

Overview

Expanding an iSCSI PV includes the following steps:

- Editing the StorageClass definition to set the `allowVolumeExpansion` field to `true`.
- Editing the PVC definition and updating the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.
- Attaching the PV must be attached 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.

The example below shows how expanding iSCSI PVs work.

Step 1: Configure the StorageClass to support volume expansion

```
$ 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

```
$ 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).

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

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               10s
```

Step 3: Define a pod that attaches the PVC

In this example, a pod is created that uses the `san-pvc`.

```
$  kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-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:     centos-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:

```
$ kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
$ tridentctl get volumes -n trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

Expand an NFS volume

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 `kubectl edit storageclass` to allow volume expansion.

Step 2: Create a PVC with the StorageClass you created

```
$ cat pvc-ontapnas.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

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:

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

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

$ tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n
trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

Import volumes

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

Drivers that support volume import

This table depicts the drivers that support importing volumes and the release they were introduced in.

Driver	Release
`ontap-nas`	19.04
`ontap-nas-flexgroup`	19.04
`solidfire-san`	19.04
`azure-netapp-files`	19.04

Driver	Release
`gcp-cvs`	19.04
`ontap-san`	19.04

Why should I import volumes?

There are several use cases for importing a volume into Trident:

- Containerizing an application and reusing its existing data set
- Using a clone of a data set for an ephemeral application
- Rebuilding a failed Kubernetes cluster
- Migrating application data during disaster recovery

How does the import work?

The Persistent Volume Claim (PVC) file is used by the volume import process to create the PVC. At a minimum, the PVC file should include the name, namespace, accessModes, and storageClassName fields as shown in the following example.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class
```

The `tridentctl` client is used to import an existing storage volume. Trident imports the volume by persisting volume metadata and creating the PVC and PV.

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

To import a storage volume, specify the name of the Astra Trident backend containing the volume, as well as the name that uniquely identifies the volume on the storage (for example: ONTAP FlexVol, Element Volume, CVS Volume path). The storage volume must allow read/write access and be accessible by the specified Astra Trident backend. The `-f` string argument is required and specifies the path to the YAML or JSON PVC file.

When Astra Trident receives the import volume request, 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.

When a volume is imported with the `--no-manage` argument, Trident does not perform any additional operations on the PVC or PV for the lifecycle of the objects. Because Trident ignores PV and PVC events for `--no-manage` objects, the storage volume is not deleted when the PV is deleted. 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.

Trident 19.07 and later handle the attachment of PVs and mounts the volume as part of importing it. For imports using earlier versions of Astra Trident, there will not be any operations in the data path and the volume import will not verify if the volume can be mounted. If a mistake is made with volume import (for example, the StorageClass is incorrect), you can recover by changing the reclaim policy on the PV to `retain`, deleting the PVC and PV, and retrying the volume import command.

`ontap-nas` and `ontap-nas-flexgroup` imports

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.

> An ONTAP volume must be of type rw to be imported by Trident. If a volume is of type dp, it is a SnapMirror destination volume; you should break the mirror relationship before importing the volume into Trident.

> The `ontap-nas` driver cannot import and manage qtrees. The `ontap-nas` and `ontap-nas-flexgroup` drivers do not allow duplicate volume names.

For example, to import a volume named `managed_volume` on a backend named `ontap_nas`, use the following command:

```
$ tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

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

To import a volume named `unmanaged_volume` (on the `ontap_nas backend`), which Trident will not manage, use the following command:

```
$ tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-file>
--no-manage

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

When using the `--no-manage` argument, Trident does not rename the volume or validate if the volume was mounted. The volume import operation fails if the volume was not mounted manually.

> A previously existing bug with importing volumes with custom UnixPermissions has been fixed. You can specify unixPermissions in your PVC definition or backend configuration, and instruct Astra Trident to import the volume accordingly.

`ontap-san` import

Astra Trident can also 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. You can use the `tridentctl import` command in the same way as in other cases:

- Include the name of the `ontap-san` backend.
- Provide the name of the FlexVol that needs to be imported. Remember, this FlexVol contains only one LUN that must be imported.
- Provide the path of the PVC definition that must be used with the `-f` flag.
- Choose between having the PVC managed or unmanaged. By default, Trident will manage the PVC and rename the FlexVol and LUN on the backend. To import as an unmanaged volume, pass the `--no-manage` flag.

> When importing an unmanaged `ontap-san` volume, you should make sure that the LUN in the FlexVol is named `lun0` and is mapped to an igroup with the desired initiators. Astra Trident automatically handles this for a managed import.

Astra Trident will then import the FlexVol and associate it with the PVC definition. Astra Trident also renames the FlexVol to the `pvc-<uuid>` format and the LUN within the FlexVol to `lun0`.

> It is recommended to import volumes that do not have existing active connections. If you are looking to import an actively used volume, clone the volume first and then do the import.

Example

To import the `ontap-san-managed` FlexVol that is present on the `ontap_san_default` backend, run the `tridentctl import` command as:

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

An ONTAP volume must be of type rw to be imported by Astra Trident. If a volume is of type dp, it is a SnapMirror destination volume; you should break the mirror relationship before importing the volume into Astra Trident.

`element` import

You can import NetApp Element software/NetApp HCI volumes to your Kubernetes cluster with Trident. You need the name of your Astra Trident backend, and the unique name of the volume and the PVC file as the arguments for the `tridentctl import` command.

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

The Element driver supports duplicate volume names. If there are duplicate volume names, Trident's volume import process returns an error. As a workaround, clone the volume and provide a unique volume name. Then import the cloned volume.

`gcp-cvs` **import**

To import a volume backed by the NetApp Cloud Volumes Service in GCP, identify the volume by its volume path instead of its name.

To import an `gcp-cvs` volume on the backend called `gcpcvs_YEppr` with the volume path of `adroit-jolly-swift`, use the following command:

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

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

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`.

`azure-netapp-files` **import**

To import an `azure-netapp-files` volume on the backend called `azurenetappfiles_40517` with the volume path `importvol1`, run the following command:

```
$ tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

The volume path for the ANF volume is present in the mount path after the :/. For example, if the mount path is `10.0.0.2:/importvol1`, the volume path is `importvol1`.

Prepare the worker node

All of the worker nodes in the Kubernetes cluster need to be able to mount the volumes that you have provisioned for your pods. If you are using the `ontap-nas`, `ontap-nas-economy`, or `ontap-nas-flexgroup` driver for one of your backends, your worker nodes need the NFS tools. Otherwise they require the iSCSI tools.

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

You should always reboot your worker nodes after installing the NFS or iSCSI tools, or else attaching volumes to containers might fail.

NFS volumes

Protocol	Operating system	Commands
NFS	RHEL/CentOS	`sudo yum install -y nfs-utils`
NFS	Ubuntu/Debian	`sudo apt-get install -y nfs-common`

You should ensure that the NFS service is started up during boot time.

iSCSI volumes

Consider the following when using iSCSI volumes:

- 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 RHEL or CentOS version 8.2 or later with the `solidfire-san` driver, ensure that the CHAP authentication algorithm is set to MD5 in `/etc/iscsi/iscsid.conf`.

```
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, make sure to specify the `discard` mountOption in the StorageClass to perform inline space reclamation. See RedHat's documentation.

<table>
<tr><th>Protocol</th><th>Operating system</th><th>Commands</th></tr>
<tr><td>iSCSI</td><td>RHEL/CentOS</td><td>
1. Install the following system packages:

<code>sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-mapper-multipath</code>

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

<code>rpm -q iscsi-initiator-utils</code>

3. Set scanning to manual:

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

4. Enable multipathing:

<code>sudo mpathconf --enable --with_multipathd y --find_multipaths n</code>

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

5. Ensure that <code>iscsid</code> and <code>multipathd</code> are running:

<code>sudo systemctl enable --now iscsid multipathd</code>

6. Enable and start <code>iscsi</code>:

<code>sudo systemctl enable --now iscsi</code>
</td></tr>
</table>

Protocol	Operating system	Commands

iSCSI	Ubuntu/Debian	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`

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. `sudo systemctl status open-iscsi`

If you want to learn more about automatic worker node preparation, which is a beta feature, see here.

Automatic worker node preparation

Astra Trident can automatically install the required `NFS` and `iSCSI` tools on the nodes present in the Kubernetes cluster. This is a **beta feature** and is **not meant for** production clusters. Today, the feature is available for nodes that run **CentOS, RHEL, and Ubuntu**.

For this feature, Astra Trident includes a new install flag: `--enable-node-prep` for installations deployed with `tridentctl`. For deployments with the Trident operator, use the Boolean option `enableNodePrep`.

The `--enable-node-prep` installation option tells Astra Trident to install and ensure that NFS and iSCSI packages and/or services are running when a volume is mounted on a worker node. This is a **beta feature** meant to be used in dev/test environments that is **not qualified** for production use.

When the `--enable-node-prep` flag is included for Astra Trident installations deployed with `tridentctl`, here is what happens:

1. As part of the installation, Astra Trident registers the nodes it runs on.
2. When a Persistent Volume Claim (PVC) request is made, Astra Trident creates a PV from one of the backends it manages.
3. Using the PVC in a pod would require Astra Trident to mount the volume on the node the pod runs on. Astra Trident attempts to install the required NFS/iSCSI client utilities and to ensure that the required services are active. This is done before the volume is mounted.

The preparation of a worker node is done only once as part of the first attempt made to mount a volume. All subsequent volume mounts should succeed as long as no changes outside Astra Trident touch the `NFS` and `iSCSI` utilities.

In this manner, Astra Trident can ensure that all the nodes in a Kubernetes cluster have the required utilities needed to mount and attach volumes. For NFS volumes, the export policy should also permit the volume to be mounted. Trident can automatically manage export policies per backend; alternatively, users can manage export policies out-of-band.

Monitor Astra Trident

Astra Trident provides a set of Prometheus metrics endpoints that you can use to monitor Astra Trident's performance.

The metrics provided by Astra Trident enable you to do the following:

- Keep tabs on Astra Trident's health and configuration. You can examine how successful operations are and if it can communicate with the backends as expected.

- Examine backend usage information and understand how many volumes are provisioned on a backend and the amount of space consumed, and so on.
- Maintain a mapping of the amount of volumes provisioned on available backends.
- Track performance. You can take a look at how long it takes for Astra Trident to communicate to backends and perform operations.

> By default, Trident's metrics are exposed on the target port `8001` at the `/metrics` endpoint. These metrics are **enabled by default** when Trident is installed.

What you'll need

- A Kubernetes cluster with Astra Trident installed.
- A Prometheus instance. This can be a containerized Prometheus deployment or you can choose to run Prometheus as a native application.

Step 1: Define a Prometheus target

You should define a Prometheus target to gather the metrics and obtain information about the backends Astra Trident manages, the volumes it creates, and so on. This blog explains how you can use Prometheus and Grafana with Astra Trident to retrieve metrics. The blog explains how you can run Prometheus as an operator in your Kubernetes cluster and the creation of a ServiceMonitor to obtain Astra Trident's metrics.

Step 2: Create a Prometheus ServiceMonitor

To consume the Trident metrics, you should create a Prometheus ServiceMonitor that watches the `trident-csi` service and listens on the `metrics` port. A sample ServiceMonitor looks like this:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
      release: prom-operator
  spec:
    jobLabel: trident
    selector:
      matchLabels:
        app: controller.csi.trident.netapp.io
    namespaceSelector:
      matchNames:
      - trident
    endpoints:
    - port: metrics
      interval: 15s
```

This ServiceMonitor definition retrieves metrics returned by the `trident-csi` service and specifically looks for the `metrics` endpoint of the service. As a result, Prometheus is now configured to understand Astra

Trident’s
metrics.

In addition to metrics available directly from Astra Trident, kubelet exposes many `kubelet_volume_*` metrics via it’s own metrics endpoint. Kubelet can provide information about the volumes that are attached, and pods and other internal operations it handles. See here.

Step 3: Query Trident metrics with PromQL

PromQL is good for creating expressions that return time-series or tabular data.

Here are some PromQL queries that you can use:

Get Trident health information

- **Percentage of HTTP 2XX responses from Astra Trident**

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Percentage of REST responses from Astra Trident via status code**

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Average duration in ms of operations performed by Astra Trident**

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Get Astra Trident usage information

- **Average volume size**

```
trident_volume_allocated_bytes/trident_volume_count
```

- **Total volume space provisioned by each backend**

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

Get individual volume usage

> ⓘ This is enabled only if kubelet metrics are also gathered.

- **Percentage of used space for each volume**

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Learn about Astra Trident AutoSupport telemetry

By default, Astra Trident sends Prometheus metrics and basic backend information to NetApp on a daily cadence.

- To stop Astra Trident from sending Prometheus metrics and basic backend information to NetApp, pass the `--silence-autosupport` flag during Astra Trident installation.
- Astra Trident can also send container logs to NetApp Support on-demand via `tridentctl send autosupport`. You will need to trigger Astra Trident to upload it's logs. Before you submit logs, you should accept NetApp's
privacy policy.
- Unless specified, Astra Trident fetches the logs from the past 24 hours.
- You can specify the log retention timeframe with the `--since` flag. For example: `tridentctl send autosupport --since=1h`. This information is collected and sent via a `trident-autosupport` container
that is installed alongside Astra Trident. You can obtain the container image at Trident AutoSupport.
- Trident AutoSupport does not gather or transmit Personally Identifiable Information (PII) or Personal Information. It comes with a EULA that is not applicable to the Trident container image itself. You can learn more about NetApp's commitment to data security and trust here.

An example payload sent by Astra Trident looks like this:

```
{
  "items": [
    {
      "backendUUID": "ff3852e1-18a5-4df4-b2d3-f59f829627ed",
      "protocol": "file",
      "config": {
        "version": 1,
        "storageDriverName": "ontap-nas",
        "debug": false,
        "debugTraceFlags": null,
        "disableDelete": false,
        "serialNumbers": [
          "nwkvzfanek_SN"
        ],
        "limitVolumeSize": ""
      },
      "state": "online",
      "online": true
    }
  ]
}
```


- The AutoSupport messages are sent to NetApp's AutoSupport endpoint. If you are using a private registry to store container images, you can use the `--image-registry` flag.
- You can also configure proxy URLs by generating the installation YAML files. This can be done by using `tridentctl install --generate-custom-yaml` to create the YAML files and adding the `--proxy-url` argument for the `trident-autosupport` container in `trident-deployment.yaml`.

Disable Astra Trident metrics

To **disable** metrics from being reported, you should generate custom YAMLs (using the `--generate-custom-yaml` flag) and edit them to remove the `--metrics` flag from being invoked for the `trident-main` container.