Provision and manage volumes

Astra Trident

NetApp
January 14, 2026

This PDF was generated from https://docs.netapp.com/us-en/trident-2402/trident-use/vol-provision.html on January 14, 2026. Always check docs.netapp.com for the latest.

Table of Contents

Provision and manage volumes . . . 1
- Provision a volume . . . 1
  - Overview . . . 1
  - Create the PV and PVC . . . 4
- Expand volumes . . . 5
  - Expand an iSCSI volume . . . 5
  - Expand an NFS volume . . . 9
- Import volumes . . . 12
  - Overview and considerations . . . 12
  - Import a volume . . . 13
  - Examples . . . 14
- Share an NFS volume across namespaces . . . 20
  - Features . . . 20
  - Quick start . . . 21
  - Configure the source and destination namespaces . . . 22
  - Delete a shared volume . . . 23
  - Use `tridentctl get` to query subordinate volumes . . . 23
  - Limitations . . . 24
  - For more information . . . 24
- Replicate volumes using SnapMirror . . . 24
  - Replication prerequisites . . . 24
  - Create a mirrored PVC . . . 25
  - Volume Replication States . . . 28
  - Promote secondary PVC during an unplanned failover . . . 28
  - Promote secondary PVC during a planned failover . . . 29
  - Restore a mirror relationship after a failover . . . 29
  - Additional operations . . . 29
  - Update mirror relationships when ONTAP is online . . . 30
  - Update mirror relationships when ONTAP is offline . . . 30
  - Enable Astra Control Provisioner . . . 31
- Use CSI Topology . . . 40
  - Overview . . . 40
  - Step 1: Create a topology-aware backend . . . 42
  - Step 2: Define StorageClasses that are topology aware . . . 44
  - Step 3: Create and use a PVC . . . 45
  - Update backends to include `supportedTopologies` . . . 48
  - Find more information . . . 48
- Work with snapshots . . . 48
  - Overview . . . 48
  - Create a volume snapshot . . . 49
  - Create a PVC from a volume snapshot . . . 50
  - Import a volume snapshot . . . 51

Recover volume data using snapshots . . . 53
In-place volume restoration from a snapshot . . . 53
Delete a PV with associated snapshots . . . 55
Deploy a volume snapshot controller . . . 55
Related links . . . 56

Provision and manage volumes

Provision a volume

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

Overview

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

The PVC can be configured to request storage of a certain size or access mode. Using the associated StorageClass, the cluster administrator can control more than PersistentVolume size and access mode—such as performance or service level.

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

Sample manifests

PersistentVolume sample manifest

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

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

PersistentVolumeClaim sample manifests

These examples show basic PVC configuration options.

PVC with RWO access

This example shows a basic PVC with RWO access that is associated with a StorageClass named `basic-csi`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

PVC with NVMe/TCP

This example shows a basic PVC for NVMe/TCP with RWO access that is associated with a StorageClass named `protection-gold`.

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Pod manifest samples

These examples show basic configurations to attach the PVC to a pod.

Basic configuration

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

Basic NVMe/TCP configuration

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      resources: {}
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: pvc-san-nvme
```

Create the PV and PVC

Steps

1. Create the PV.

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

2. Verify the PV status.

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

3. Create the PVC.

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

4. Verify the PVC status.

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

5. Mount the volume in a pod.

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

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

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

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

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

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

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

Expand volumes

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

Expand an iSCSI volume

You can expand an iSCSI Persistent Volume (PV) by using the CSI provisioner.

> iSCSI volume expansion is supported by the `ontap-san`, `ontap-san-economy`, `solidfire-san` drivers and requires Kubernetes 1.16 and later.

Step 1: Configure the StorageClass to support volume expansion

Edit the StorageClass definition to set the `allowVolumeExpansion` field to `true`.

```
cat storageclass-ontapsan.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

For an already existing StorageClass, edit it to include the `allowVolumeExpansion` parameter.

Step 2: Create a PVC with the StorageClass you created

Edit the PVC definition and update the `spec.resources.requests.storage` to reflect the newly desired size, which must be greater than the original size.

```
cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

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

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

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

Step 3: Define a pod that attaches the PVC

Attach the PV to a pod for it to be resized. There are two scenarios when resizing an iSCSI PV:

- If the PV is attached to a pod, Astra Trident expands the volume on the storage backend, rescans the device, and resizes the filesystem.
- When attempting to resize an unattached PV, Astra Trident expands the volume on the storage backend. After the PVC is bound to a pod, Trident rescans the device and resizes the filesystem. Kubernetes then updates the PVC size after the expand operation has successfully completed.

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

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

Step 4: Expand the PV

To resize the PV that has been created from 1Gi to 2Gi, edit the PVC definition and update the `spec.resources.requests.storage` to 2Gi.

```
kubectl edit pvc san-pvc
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 ...
```

Step 5: Validate the expansion

You can validate the expansion worked correctly by checking the size of the PVC, PV, and the Astra Trident volume:

```
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 1GiB:

```
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`.

Overview and considerations

You might import a volume into Astra Trident to:

- Containerize an application and reuse its existing data set
- Use a clone of a data set for an ephemeral application
- Rebuild a failed Kubernetes cluster
- Migrate application data during disaster recovery

Considerations

Before importing a volume, review the following considerations.

- Astra Trident can import RW (read-write) type ONTAP volumes only. DP (data protection) type volumes are SnapMirror destination volumes. You should break the mirror relationship before importing the volume into

Astra Trident.

- We suggest importing volumes without active connections. To import an actively-used volume, clone the volume and then perform the import.

This is especially important for block volumes as Kubernetes would be unaware of the previous connection and could easily attach an active volume to a pod. This can result in data corruption.

- Though `StorageClass` must be specified on a PVC, Astra Trident does not use this parameter during import. Storage classes are used during volume creation to select from available pools based on storage characteristics. Because the volume already exists, no pool selection is required during import. Therefore, the import will not fail even if the volume exists on a backend or pool that does not match the storage class specified in the PVC.
- The existing volume size is determined and set in the PVC. After the volume is imported by the storage driver, the PV is created with a ClaimRef to the PVC.
  - The reclaim policy is initially set to `retain` in the PV. After Kubernetes successfully binds the PVC and PV, the reclaim policy is updated to match the reclaim policy of the Storage Class.
  - If the reclaim policy of the Storage Class is `delete`, the storage volume will be deleted when the PV is deleted.
- By default, Astra Trident manages the PVC and renames the FlexVol and LUN on the backend. You can pass the `--no-manage` flag to import an unmanaged volume. If you use `--no-manage`, Astra Trident does not perform any additional operations on the PVC or PV for the lifecycle of the objects. The storage volume is not deleted when the PV is deleted and other operations such as volume clone and volume resize are also ignored.

This option is useful if you want to use Kubernetes for containerized workloads but otherwise want to manage the lifecycle of the storage volume outside of Kubernetes.

- An annotation is added to the PVC and PV that serves a dual purpose of indicating that the volume was imported and if the PVC and PV are managed. This annotation should not be modified or removed.

Import a volume

You can use `tridentctl import` to import a volume.

Steps

1. Create the Persistent Volume Claim (PVC) file (for example, `pvc.yaml`) that will be used to create the PVC. The PVC file should include `name`, `namespace`, `accessModes`, and `storageClassName`. Optionally, you can specify `unixPermissions` in your PVC definition.

   The following is an example of a minimum specification:

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

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

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

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

Examples

Review the following volume import examples for supported drivers.

ONTAP NAS and ONTAP NAS FlexGroup

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

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

Each volume created with the `ontap-nas` driver is a FlexVol on the ONTAP cluster. Importing FlexVols with the `ontap-nas` driver works the same. A FlexVol that already exists on an ONTAP cluster can be imported as a `ontap-nas` PVC. Similarly, FlexGroup vols can be imported as `ontap-nas-flexgroup` PVCs.

ONTAP NAS examples

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

Managed volume

The following example imports a volume named `managed_volume` on a backend named `ontap_nas`:

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

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

Unmanaged volume

When using the `--no-manage` argument, Astra Trident does not rename the volume.

The following example imports `unmanaged_volume` on the `ontap_nas` backend:

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

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

ONTAP SAN

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

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

ONTAP SAN examples

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

Managed volume

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

The following example imports the `ontap-san-managed` FlexVol that is present on the `ontap_san_default` backend:

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
```

Unmanaged volume

The following example imports `unmanaged_example_volume` on the `ontap_san` backend:

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
```

If you have LUNS mapped to igroups that share an IQN with a Kubernetes node IQN, as shown in the following example, you will receive the error: `LUN already mapped to initiator(s) in this group`. You will need to remove the initiator or unmap the LUN to import the volume.

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

Element

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

The Element driver supports duplicate volume names. However, Astra Trident returns an error if there are duplicate volume names. As a workaround, clone the volume, provide a unique volume name, and import the cloned volume.

Element example

The following example imports an `element-managed` volume on backend `element_default`.

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud Platform

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

To import a volume backed by the NetApp Cloud Volumes Service in Google Cloud Platform, identify the volume by its volume path. The volume path is the portion of the volume's export path after the `:/`. For example, if the export path is `10.0.0.1:/adroit-jolly-swift`, the volume path is `adroit-jolly-swift`.

Google Cloud Platform example

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

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

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

Azure NetApp Files

Astra Trident supports volume import using the `azure-netapp-files` driver.

To import an Azure NetApp Files volume, identify the volume by its volume path. The volume path is the portion of the volume's export path after the `:/`. For example, if the mount path is `10.0.0.2:/importvol1`, the volume path is `importvol1`.

Azure NetApp Files example

The following example imports an `azure-netapp-files` volume on backend `azurenetappfiles_40517` with the volume path `importvol1`.

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

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

Share an NFS volume across namespaces

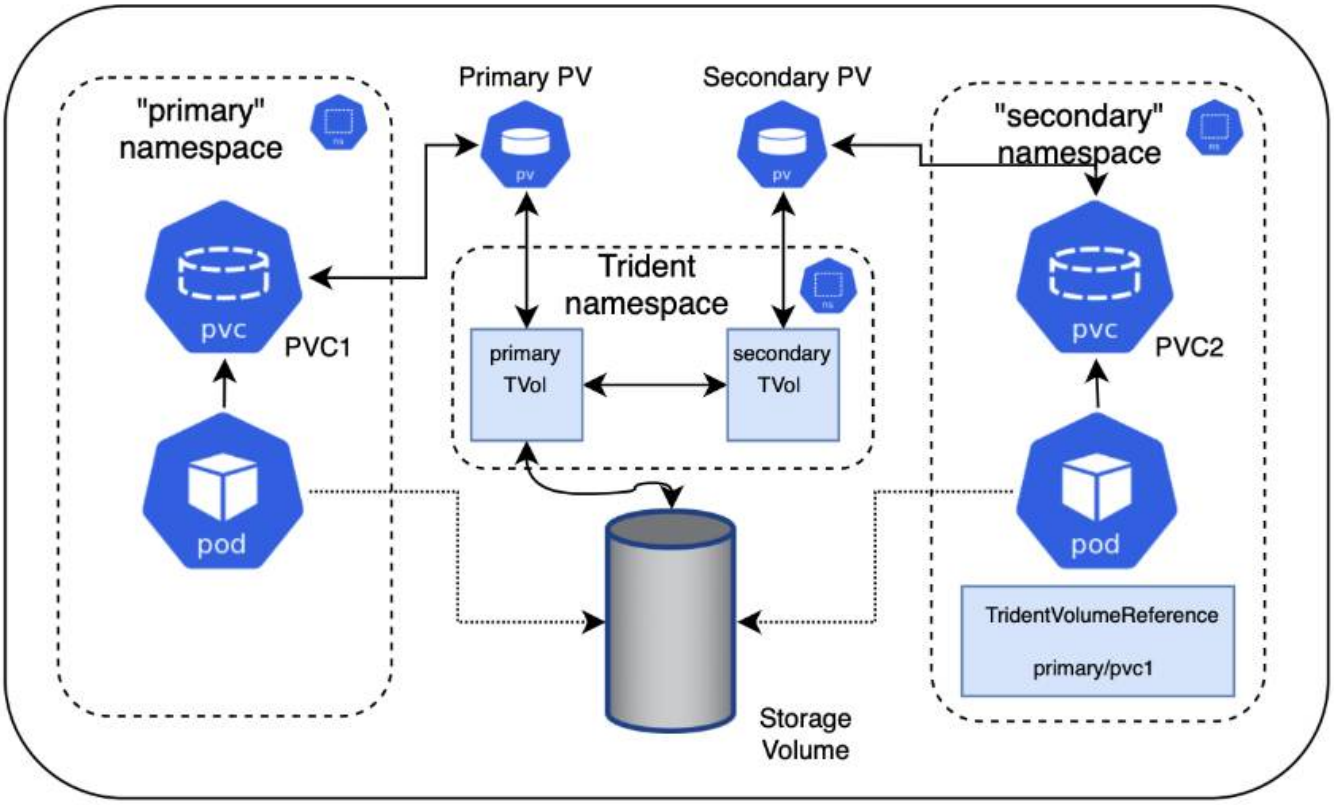
Using Astra Trident, you can create a volume in a primary namespace and share it in one or more secondary namespaces.

Features

The Astra TridentVolumeReference CR allows you to securely share ReadWriteMany (RWX) NFS volumes across one or more Kubernetes namespaces. This Kubernetes-native solution has the following benefits:

- Multiple levels of access control to ensure security
- Works with all Trident NFS volume drivers
- No reliance on tridentctl or any other non-native Kubernetes feature

This diagram illustrates NFS volume sharing across two Kubernetes namespaces.



Quick start

You can set up NFS volume sharing in just a few steps.

1 Configure source PVC to share the volume

The source namespace owner grants permission to access the data in the source PVC.

2 Grant permission to create a CR in the destination namespace

The cluster administrator grants permission to the owner of the destination namespace to create the TridentVolumeReference CR.

3 Create TridentVolumeReference in the destination namespace

The owner of the destination namespace creates the TridentVolumeReference CR to refer to the source PVC.

4 Create the subordinate PVC in the destination namespace

The owner of the destination namespace creates the subordinate PVC to use the data source from the source PVC.

Configure the source and destination namespaces

To ensure security, cross namespace sharing requires collaboration and action by the source namespace owner, cluster administrator, and destination namespace owner. The user role is designated in each step.

Steps

1. **Source namespace owner:** Create the PVC (`pvc1`) in the source namespace that grants permission to share with the destination namespace (`namespace2`) using the `shareToNamespace` annotation.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

   Astra Trident creates the PV and its backend NFS storage volume.

   - You can share the PVC to multiple namespaces using a comma-delimited list. For example, `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`.
   - You can share to all namespaces using `*`. For example, `trident.netapp.io/shareToNamespace: *`
   - You can update the PVC to include the `shareToNamespace` annotation at any time.

2. **Cluster admin:** Create the custom role and kubeconfig to grant permission to the destination namespace owner to create the TridentVolumeReference CR in the destination namespace.
3. **Destination namespace owner:** Create a TridentVolumeReference CR in the destination namespace that refers to the source namespace `pvc1`.

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination namespace owner:** Create a PVC (`pvc2`) in destination namespace (`namespace2`) using the `shareFromPVC` annotation to designate the source PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

The size of the destination PVC must be less than or equal than the source PVC.

Results

Astra Trident reads the `shareFromPVC` annotation on the destination PVC and creates the destination PV as a subordinate volume with no storage resource of its own that points to the source PV and shares the source PV storage resource. The destination PVC and PV appear bound as normal.

Delete a shared volume

You can delete a volume that is shared across multiple namespaces. Astra Trident will remove access to the volume on the source namespace and maintain access for other namespaces that share the volume. When all namespaces that reference the volume are removed, Astra Trident deletes the volume.

Use `tridentctl get` to query subordinate volumes

Using the `tridentctl` utility, you can run the `get` command to get subordinate volumes. For more information, refer to `tridentctl` commands and options.

```
Usage:
  tridentctl get [option]
```

Flags:

- `-h, --help`: Help for volumes.
- `--parentOfSubordinate string`: Limit query to subordinate source volume.
- `--subordinateOf string`: Limit query to subordinates of volume.

Limitations

- Astra Trident cannot prevent destination namespaces from writing to the shared volume. You should use file locking or other processes to prevent overwriting shared volume data.
- You cannot revoke access to the source PVC by removing the `shareToNamespace` or `shareFromNamespace` annotations or deleting the `TridentVolumeReference` CR. To revoke access, you must delete the subordinate PVC.
- Snapshots, clones, and mirroring are not possible on subordinate volumes.

For more information

To learn more about cross-namespace volume access:

- Visit Sharing volumes between namespaces: Say hello to cross-namespace volume access.

Replicate volumes using SnapMirror

Using Astra Control Provisioner, you can create mirror relationships between a source volume on one cluster and the destination volume on the peered cluster for replicating data for disaster recovery. You can use a namespaced Custom Resource Definition (CRD) to perform the following operations:

- Create mirror relationships between volumes (PVCs)
- Remove mirror relationships between volumes
- Break the mirror relationships
- Promote the secondary volume during disaster conditions (failovers)
- Perform lossless transition of applications from cluster to cluster (during planned failovers or migrations)

Replication prerequisites

Ensure that the following prerequisites are met before you begin:

ONTAP clusters

- **Astra Control Provisioner**: Astra Control Provisioner version 23.10 or later must exist on both the source and destination Kubernetes clusters that utilize ONTAP as a backend.
- **Licenses**: ONTAP SnapMirror asynchronous licenses using the Data Protection bundle must be enabled

on both the source and destination ONTAP clusters. Refer to SnapMirror licensing overview in ONTAP for more information.

Peering

- **Cluster and SVM**: The ONTAP storage backends must be peered. Refer to Cluster and SVM peering overview for more information.

  > Ensure that the SVM names used in the replication relationship between two ONTAP clusters are unique.

- **Astra Control Provisioner and SVM**: The peered remote SVMs must be available to Astra Control Provisioner on the destination cluster.

Supported drivers

- Volume replication is supported for the ontap-nas and ontap-san drivers.

Create a mirrored PVC

Follow these steps and use the CRD examples to create mirror relationship between primary and secondary volumes.

Steps

1. Perform the following steps on the primary Kubernetes cluster:
   a. Create a StorageClass object with the `trident.netapp.io/replication: true` parameter.

      Example

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. Create a PVC with previously created StorageClass.

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. Create a MirrorRelationship CR with local information.

Example

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Astra Control Provisioner fetches the internal information for the volume and the volume's current data protection (DP) state, then populates the status field of the MirrorRelationship.

d. Get the TridentMirrorRelationship CR to obtain the internal name and SVM of the PVC.

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. Perform the following steps on the secondary Kubernetes cluster:
   a. Create a StorageClass with the trident.netapp.io/replication: true parameter.

      Example

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

   b. Create a MirrorRelationship CR with destination and source information.

      Example

      ```
      kind: TridentMirrorRelationship
      apiVersion: trident.netapp.io/v1
      metadata:
        name: csi-nas
      spec:
        state: established
        volumeMappings:
        - localPVCName: csi-nas
          remoteVolumeHandle:
      "datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
      ```

Astra Control Provisioner will create a SnapMirror relationship with the configured relationship policy name (or default for ONTAP) and initialize it.

c. Create a PVC with previously created StorageClass to act as the secondary (SnapMirror destination).

Example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Astra Control Provisioner will check for the TridentMirrorRelationship CRD and fail to create the volume if the relationship does not exist. If the relationship exists, Astra Control Provisioner will ensure the new FlexVol volume is placed onto an SVM that is peered with the remote SVM defined in the MirrorRelationship.

Volume Replication States

A Trident Mirror Relationship (TMR) is a CRD that represents one end of a replication relationship between PVCs. The destination TMR has a state, which tells Astra Control Provisioner what the desired state is. The destination TMR has the following states:

- **Established**: the local PVC is the destination volume of a mirror relationship, and this is a new relationship.
- **Promoted**: the local PVC is ReadWrite and mountable, with no mirror relationship currently in effect.
- **Reestablished**: the local PVC is the destination volume of a mirror relationship and was also previously in that mirror relationship.
  - The reestablished state must be used if the destination volume was ever in a relationship with the source volume because it overwrites the destination volume contents.
  - The reestablished state will fail if the volume was not previously in a relationship with the source.

Promote secondary PVC during an unplanned failover

Perform the following step on the secondary Kubernetes cluster:

- Update the *spec.state* field of TridentMirrorRelationship to `promoted`.

Promote secondary PVC during a planned failover

During a planned failover (migration), perform the following steps to promote the secondary PVC:

Steps

1. On the primary Kubernetes cluster, create a snapshot of the PVC and wait until the snapshot is created.
2. On the primary Kubernetes cluster, create the SnapshotInfo CR to obtain internal details.

   Example

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. On secondary Kubernetes cluster, update the *spec.state* field of the *TridentMirrorRelationship* CR to *promoted* and *spec.promotedSnapshotHandle* to be the internalName of the snapshot.
4. On secondary Kubernetes cluster, confirm the status (status.state field) of TridentMirrorRelationship to promoted.

Restore a mirror relationship after a failover

Before restoring a mirror relationship, choose the side that you want to make as the new primary.

Steps

1. On the secondary Kubernetes cluster, ensure that the values for the *spec.remoteVolumeHandle* field on the TridentMirrorRelationship is updated.
2. On secondary Kubernetes cluster, update the *spec.mirror* field of TridentMirrorRelationship to `reestablished`.

Additional operations

Astra Control Provisioner supports the following operations on the primary and secondary volumes:

Replicate primary PVC to a new secondary PVC

Ensure that you already have a primary PVC and a secondary PVC.

Steps

1. Delete the PersistentVolumeClaim and TridentMirrorRelationship CRDs from the established secondary (destination) cluster.
2. Delete the TridentMirrorRelationship CRD from the primary (source) cluster.
3. Create a new TridentMirrorRelationship CRD on the primary (source) cluster for the new secondary (destination) PVC you want to establish.

Resize a mirrored, primary or secondary PVC

The PVC can be resized as normal, ONTAP will automatically expand any destination flevxols if the amount of data exceeds the current size.

Remove replication from a PVC

To remove replication, perform one of the following operations on the current secondary volume:

- Delete the MirrorRelationship on the secondary PVC. This breaks the replication relationship.
- Or, update the spec.state field to *promoted*.

Delete a PVC (that was previously mirrored)

Astra Control Provisioner checks for replicated PVCs, and releases the replication relationship before attempting to delete the volume.

Delete a TMR

Deleting a TMR on one side of a mirrored relationship causes the remaining TMR to transition to *promoted* state before Astra Control Provisioner completes the deletion. If the TMR selected for deletion is already in *promoted* state, there is no existing mirror relationship and the TMR will be removed and Astra Control Provisioner will promote the local PVC to *ReadWrite*. This deletion releases SnapMirror metadata for the local volume in ONTAP. If this volume is used in a mirror relationship in the future, it must use a new TMR with an *established* volume replication state when creating the new mirror relationship.

Update mirror relationships when ONTAP is online

Mirror relationships can be updated any time after they are established. You can use the `state: promoted` or `state: reestablished` fields to update the relationships.
When promoting a destination volume to a regular ReadWrite volume, you can use *promotedSnapshotHandle* to specify a specific snapshot to restore the current volume to.

Update mirror relationships when ONTAP is offline

You can use a CRD to perform a SnapMirror update without Astra Control having direct connectivity to the ONTAP cluster. Refer to the following example format of the TridentActionMirrorUpdate:

Example

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` reflects the state of the TridentActionMirrorUpdate CRD. It can take a value from *Succeeded*, *In Progress*, or *Failed*.

Enable Astra Control Provisioner

Trident versions 23.10 and later include the option to use Astra Control Provisioner, which enables licensed Astra Control users to access advanced storage provisioning functionality. Astra Control Provisioner provides this extended functionality in addition to standard Astra Trident CSI-based functionality. You can use this procedure to enable and install Astra Control Provisioner.

Your Astra Control Service subscription automatically includes the license for Astra Control Provisioner use.

In coming Astra Control updates, Astra Control Provisioner will replace Astra Trident as storage provisioner and orchestrator and be mandatory for Astra Control use. Because of this, it's strongly recommended that Astra Control users enable Astra Control Provisioner. Astra Trident will continue to remain open source and be released, maintained, supported, and updated with new CSI and other features from NetApp.

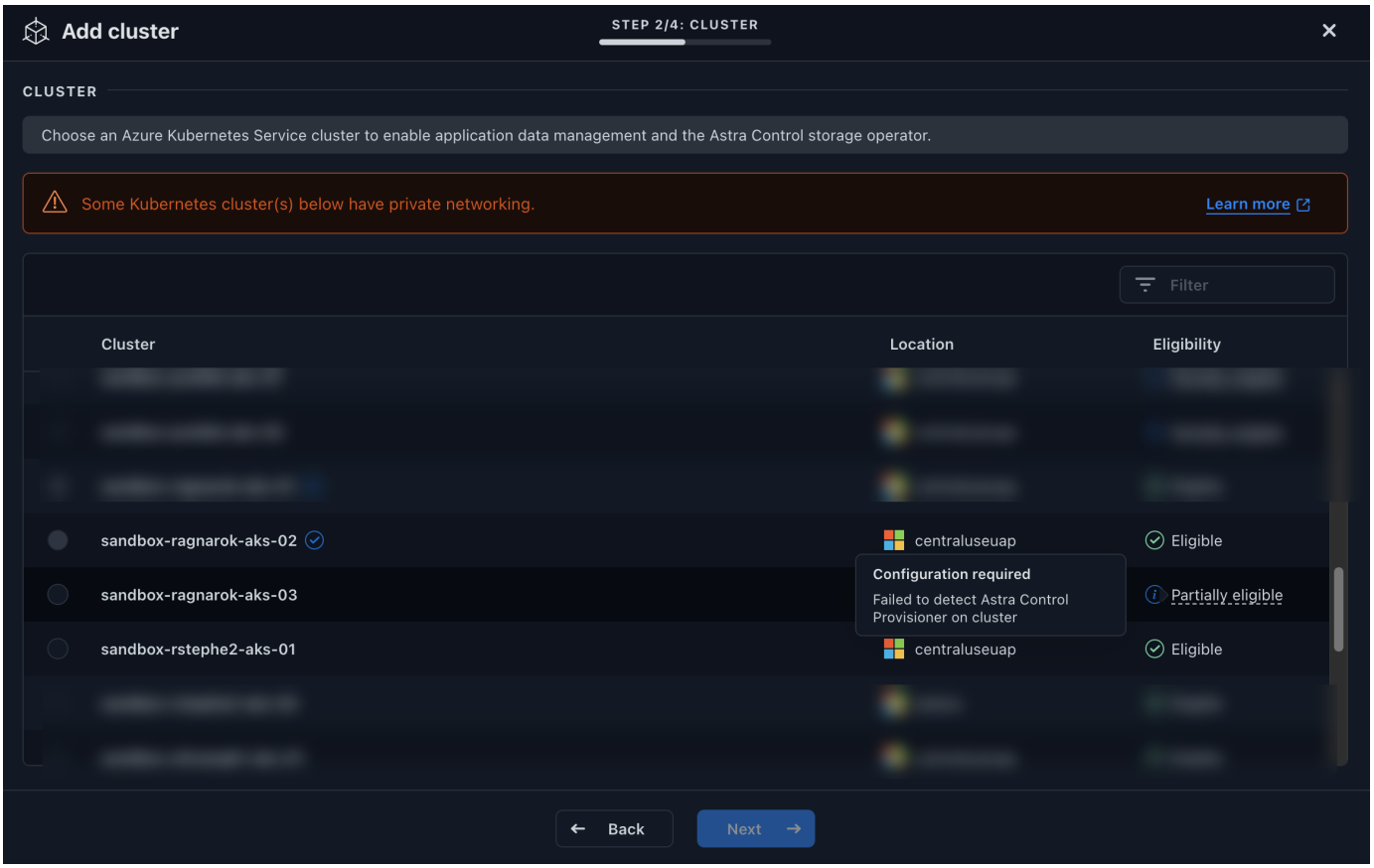
How do I know if I need to enable Astra Control Provisioner?

If you add a cluster to Astra Control Service that does not have Astra Trident previously installed, the cluster will be marked as `Eligible`. After you add the cluster to Astra Control, Astra Control Provisioner will be automatically enabled.

If your cluster is not marked `Eligible`, it will be marked `Partially eligible` because of one of the following:

- It's using an older version of Astra Trident
- It's using an Astra Trident 23.10 that does not yet have the provisioner option enabled
- It's a cluster type that does not allow automatic enablement

For `Partially eligible` cases, use these instructions to manually enable Astra Control Provisioner for your cluster.



Before you enable Astra Control Provisioner

If you have an existing Astra Trident without Astra Control Provisioner and want to enable Astra Control Provisioner, do the following first:

- **If you have Astra Trident installed, confirm that its version is within a four-release window**: You can perform a direct upgrade to Astra Trident 24.02 with Astra Control Provisioner if your Astra Trident is within a four-release window of version 24.02. For example, you can directly upgrade from Astra Trident 23.04 to 24.02.
- **Confirm that your cluster has an AMD64 system architecture**: The Astra Control Provisioner image is provided in both AMD64 and ARM64 CPU architectures, but only AMD64 is supported by Astra Control.

Steps

1. Access the NetApp Astra Control image registry:
   a. Log on to the Astra Control Service UI and record your Astra Control account ID.
      i. Select the figure icon at the top right of the page.
      ii. Select **API access**.
      iii. Write down your account ID.
   b. From the same page, select **Generate API token** and copy the API token string to the clipboard and save it in your editor.
   c. Log into the Astra Control registry using your preferred method:

```
docker login cr.astra.netapp.io -u <account-id> -p <api-token>
```

```
crane auth login cr.astra.netapp.io -u <account-id> -p <api-token>
```

2. (Custom registries only) Follow these steps to move the image to your custom registry. If you aren't using a registry, follow the Trident operator steps in the next section.

> You can use Podman instead of Docker for the following commands. If you are using a Windows environment, PowerShell is recommended.

Docker

1. Pull the Astra Control Provisioner image from the registry:

> The image pulled will not support multiple platforms and will only support the same platform as the host that pulled the image, such as Linux AMD64.

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0
--platform <cluster platform>
```

Example:

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0
--platform linux/amd64
```

2. Tag the image:

```
docker tag cr.astra.netapp.io/astra/trident-acp:24.02.0
<my_custom_registry>/trident-acp:24.02.0
```

3. Push the image to your custom registry:

```
docker push <my_custom_registry>/trident-acp:24.02.0
```

Crane

1. Copy the Astra Control Provisioner manifest to your custom registry:

```
crane copy cr.astra.netapp.io/astra/trident-acp:24.02.0
<my_custom_registry>/trident-acp:24.02.0
```

3. Determine if the original Astra Trident installation method used an operator (either manually or with Helm) or `tridentctl`.

4. Enable Astra Control Provisioner in Astra Trident using the installation method you used originally:

Astra Trident operator

1. Download the Astra Trident installer and extract it.
2. Complete these steps if you have not yet installed Astra Trident or if you removed the operator from your original Astra Trident deployment:
   a. Create the CRD:

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.1
6.yaml
```

   b. Create the trident namespace (`kubectl create namespace trident`) or confirm that the trident namespace still exists (`kubectl get all -n trident`). If the namespace has been removed, create it again.
3. Update Astra Trident to 24.02.0:

> For clusters running Kubernetes 1.24 or earlier, use `bundle_pre_1_25.yaml`.
> For clusters running Kubernetes 1.25 or later, use `bundle_post_1_25.yaml`.

```
kubectl -n trident apply -f trident-installer/deploy/<bundle-
name.yaml>
```

4. Verify Astra Trident is running:

```
kubectl get torc -n trident
```

   Response:

```
NAME      AGE
trident   21m
```

5. If you have a registry that uses secrets, create a secret to use to pull the Astra Control Provisioner image:

```
kubectl create secret docker-registry <secret_name> -n trident
--docker-server=<my_custom_registry> --docker-username=<username>
--docker-password=<token>
```

6. Edit the TridentOrchestrator CR and make the following edits:

```
kubectl edit torc trident -n trident
```

a. Set a custom registry location for the Astra Trident image or pull it from the Astra Control registry (`tridentImage: <my_custom_registry>/trident:24.02.0` or `tridentImage: netapp/trident:24.02.0`).

b. Enable Astra Control Provisioner (`enableACP: true`).

c. Set the custom registry location for the Astra Control Provisioner image or pull it from the Astra Control registry (`acpImage: <my_custom_registry>/trident-acp:24.02.0` or `acpImage: cr.astra.netapp.io/astra/trident-acp:24.02.0`).

d. If you established image pull secrets earlier in this procedure, you can set them here (`imagePullSecrets: - <secret_name>`). Use the same name secret name you established in the previous steps.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  tridentImage: <registry>/trident:24.02.0
  enableACP: true
  acpImage: <registry>/trident-acp:24.02.0
  imagePullSecrets:
  - <secret_name>
```

7. Save and exit the file. The deployment process will begin automatically.
8. Verify the operator, deployment, and replicasets are created.

```
kubectl get all -n trident
```

> There should only be **one instance** of the operator in a Kubernetes cluster. Do not create multiple deployments of the Astra Trident operator.

9. Verify the `trident-acp` container is running and that `acpVersion` is `24.02.0` with a status of `Installed`:

```
kubectl get torc -o yaml
```

Response:

```
status:
  acpVersion: 24.02.0
  currentInstallationParams:
    ...
    acpImage: <registry>/trident-acp:24.02.0
    enableACP: "true"
    ...
  ...
  status: Installed
```

tridentctl

1. Download the Astra Trident installer and extract it.
2. If you have an existing Astra Trident, uninstall it from the cluster that hosts it.
3. Install Astra Trident with Astra Control Provisioner enabled (`--enable-acp=true`):

```
./tridentctl -n trident install --enable-acp=true --acp
-image=mycustomregistry/trident-acp:24.02
```

4. Confirm that Astra Control Provisioner has been enabled:

```
./tridentctl -n trident version
```

Response:

```
+----------------+----------------+-------------+ | SERVER
VERSION | CLIENT VERSION | ACP VERSION | +----------------
+----------------+-------------+ | 24.02.0 | 24.02.0 | 24.02.0. |
+----------------+----------------+-------------+
```

Helm

1. If you have Astra Trident 23.07.1 or earlier installed, uninstall the operator and other components.
2. If your Kubernetes cluster is running 1.24 or earlier, delete psp:

```
kubectl delete psp tridentoperatorpod
```

3. Add the Astra Trident Helm repository:

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

4. Update the Helm chart:

```
helm repo update netapp-trident
```

Response:

```
Hang tight while we grab the latest from your chart
repositories...
...Successfully got an update from the "netapp-trident" chart
repository
Update Complete. □Happy Helming!□
```

5. List the images:

```
./tridentctl images -n trident
```

Response:

```
| v1.28.0            | netapp/trident:24.02.0|
|                    | docker.io/netapp/trident-
autosupport:24.02|
|                    | registry.k8s.io/sig-storage/csi-
provisioner:v4.0.0|
|                    | registry.k8s.io/sig-storage/csi-
attacher:v4.5.0|
|                    | registry.k8s.io/sig-storage/csi-
resizer:v1.9.3|
|                    | registry.k8s.io/sig-storage/csi-
snapshotter:v6.3.3|
|                    | registry.k8s.io/sig-storage/csi-node-
driver-registrar:v2.10.0 |
|                    | netapp/trident-operator:24.02.0 (optional)
```

6. Ensure that trident-operator 24.02.0 is available:

```
helm search repo netapp-trident/trident-operator --versions
```

Response:

```
NAME                              CHART VERSION   APP VERSION
DESCRIPTION
netapp-trident/trident-operator 100.2402.0      24.02.0         A
```

7. Use `helm install` and run one of the following options that include these settings:
   - A name for your deployment location
   - The Astra Trident version
   - The name of the Astra Control Provisioner image
   - The flag to enable the provisioner
   - (Optional) A local registry path. If you are using a local registry, your Trident images can be located in one registry or different registries, but all CSI images must be located in the same registry.
   - The Trident namespace

Options

- Images without a registry

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=cr.astra.netapp.io/astra/trident-
acp:24.02.0 --set enableACP=true --set operatorImage=netapp/trident-
operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

- Images in one or more registries

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=<your-registry>:<acp image> --set
enableACP=true --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=netapp/trident-operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

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

If you have any issues deploying Trident using Helm, run this command to fully uninstall Astra Trident:

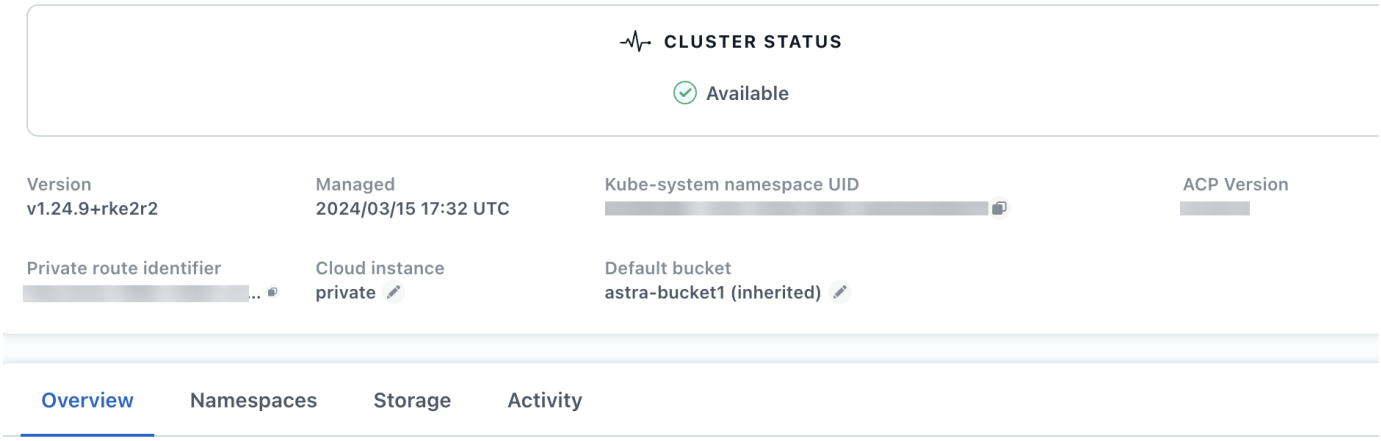
```
./tridentctl uninstall -n trident
```

Do not completely remove Astra Trident CRDs as part of your uninstall before attempting to enable Astra Control Provisioner again.

Result

Astra Control Provisioner functionality is enabled and you can use any features available for the version you are running.

After Astra Control Provisioner is installed, the cluster hosting the provisioner in the Astra Control UI will show an `ACP version` rather than `Trident version` field and current installed version number.



For more information

- Astra Trident upgrades documentation

Use CSI Topology

Astra Trident can selectively create and attach volumes to nodes present in a Kubernetes cluster by making use of the CSI Topology feature.

Overview

Using the CSI Topology feature, access to volumes can be limited to a subset of nodes, based on regions and availability zones. Cloud providers today enable Kubernetes administrators to spawn nodes that are zone based. Nodes can be located in different availability zones within a region, or across various regions. To facilitate the provisioning of volumes for workloads in a multi-zone architecture, Astra Trident uses CSI Topology.

Learn more about the CSI Topology feature here.

Kubernetes provides two unique volume binding modes:

- With `VolumeBindingMode` set to `Immediate`, Astra Trident creates the volume without any topology awareness. Volume binding and dynamic provisioning are handled when the PVC is created. This is the default `VolumeBindingMode` and is suited for clusters that do not enforce topology constraints. Persistent Volumes are created without having any dependency on the requesting pod's scheduling requirements.
- With `VolumeBindingMode` set to `WaitForFirstConsumer`, the creation and binding of a Persistent Volume for a PVC is delayed until a pod that uses the PVC is scheduled and created. This way, volumes are created to meet the scheduling constraints that are enforced by topology requirements.

> The `WaitForFirstConsumer` binding mode does not require topology labels. This can be used independent of the CSI Topology feature.

What you'll need

To make use of CSI Topology, you need the following:

- A Kubernetes cluster running a supported Kubernetes version

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"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 an example backend definition:

YAML

```
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-east1
  topology.kubernetes.io/zone: us-east1-a
- topology.kubernetes.io/region: us-east1
  topology.kubernetes.io/zone: us-east1-b
```

JSON

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "password",
 "supportedTopologies": [
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-a"},
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-b"}
]
}
```

`supportedTopologies` is used to provide a list of regions and zones per backend. These regions and zones represent the list of permissible values that can be provided in a StorageClass. For StorageClasses that contain a subset of the regions and zones provided in a backend, Astra Trident will create a volume on the backend.

You can define `supportedTopologies` per storage pool as well. See the following example:

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-a
- topology.kubernetes.io/region: us-central1
  topology.kubernetes.io/zone: us-central1-b
storage:
- labels:
    workload: production
  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:

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

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

See the following output:

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

Update backends to include `supportedTopologies`

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

Find more information

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

Work with snapshots

Kubernetes volume snapshots of Persistent Volumes (PVs) enable point-in-time copies of volumes. You can create a snapshot of a volume created using Astra Trident, import a snapshot created outside of Astra Trident, create a new volume from an existing snapshot, and recover volume data from snapshots.

Overview

Volume snapshot is supported by `ontap-nas`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `gcp-cvs`, and `azure-netapp-files` drivers.

Before you begin

You must have an external snapshot controller and Custom Resource Definitions (CRDs) to work with snapshots. This is the responsibility of the Kubernetes orchestrator (for example: Kubeadm, GKE, OpenShift).

If your Kubernetes distribution does not include the snapshot controller and CRDs, refer to Deploy a volume snapshot controller.

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

Create a volume snapshot

Steps

1. Create a `VolumeSnapshotClass`. For more information, refer to VolumeSnapshotClass.
   - The `driver` points to the Astra Trident CSI driver.
   - `deletionPolicy` can be `Delete` or `Retain`. When set to `Retain`, the underlying physical snapshot on the storage cluster is retained even when the `VolumeSnapshot` object is deleted.

     Example

     ```
     cat snap-sc.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshotClass
     metadata:
       name: csi-snapclass
     driver: csi.trident.netapp.io
     deletionPolicy: Delete
     ```

2. Create a snapshot of an existing PVC.

   Examples

   - This example creates a snapshot of an existing PVC.

     ```
     cat snap.yaml
     apiVersion: snapshot.storage.k8s.io/v1
     kind: VolumeSnapshot
     metadata:
       name: pvc1-snap
     spec:
       volumeSnapshotClassName: csi-snapclass
       source:
         persistentVolumeClaimName: pvc1
     ```

   - This example creates a volume snapshot object for a PVC named `pvc1` and the name of the snapshot is set to `pvc1-snap`. A VolumeSnapshot is analogous to a PVC and is associated with a `VolumeSnapshotContent` object that represents the actual snapshot.

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

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

   - You can identify the `VolumeSnapshotContent` object for the `pvc1-snap` VolumeSnapshot by

describing it. The `Snapshot Content Name` identifies the VolumeSnapshotContent object which serves this snapshot. The `Ready To Use` parameter indicates that the snapshot can be used to create a new PVC.

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
.
.
.
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
.
.
```

Create a PVC from a volume snapshot

You can use `dataSource` to create a PVC using a VolumeSnapshot named `<pvc-name>` as the source of the data. After the PVC is created, it can be attached to a pod and used just like any other PVC.

The PVC will be created in the same backend as the source volume. Refer to KB: Creating a PVC from a Trident PVC Snapshot cannot be created in an alternate backend.

The following example creates the PVC using `pvc1-snap` as the data source.

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

Import a volume snapshot

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

Before you begin

Astra Trident must have created or imported the snapshot's parent volume.

Steps

1. **Cluster admin:** Create a `VolumeSnapshotContent` object that references the backend snapshot. This initiates the snapshot workflow in Astra Trident.
   - Specify the name of the backend snapshot in `annotations` as `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`.
   - Specify `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` in `snapshotHandle`. This is the only information provided to Astra Trident by the external snapshotter in the `ListSnapshots` call.

     The `<volumeSnapshotContentName>` cannot always match the backend snapshot name due to CR naming constraints.

     Example

     The following example creates a `VolumeSnapshotContent` object that references backend snapshot `snap-01`.

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
```

2. **Cluster admin:** Create the `VolumeSnapshot` CR that references the `VolumeSnapshotContent` object. This requests access to use the `VolumeSnapshot` in a given namespace.

   Example

   The following example creates a `VolumeSnapshot` CR named `import-snap` that references the `VolumeSnapshotContent` named `import-snap-content`.

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: import-snap
   spec:
     # volumeSnapshotClassName: csi-snapclass (not required for pre-
   provisioned or imported snapshots)
     source:
       volumeSnapshotContentName: import-snap-content
   ```

3. **Internal processing (no action required):** The external snapshotter recognizes the newly created `VolumeSnapshotContent` and runs the `ListSnapshots` call. Astra Trident creates the `TridentSnapshot`.
   - The external snapshotter sets the `VolumeSnapshotContent` to `readyToUse` and the `VolumeSnapshot` to `true`.
   - Trident returns `readyToUse=true`.
4. **Any user:** Create a `PersistentVolumeClaim` to reference the new `VolumeSnapshot`, where the `spec.dataSource` (or `spec.dataSourceRef`) name is the `VolumeSnapshot` name.

   Example

   The following example creates a PVC referencing the `VolumeSnapshot` named `import-snap`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

Recover volume data using snapshots

The snapshot directory is hidden by default to facilitate maximum compatibility of volumes provisioned using the `ontap-nas` and `ontap-nas-economy` drivers. Enable the `.snapshot` directory to recover data from snapshots directly.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

The snapshot directory is hidden by default to facilitate maximum compatibility of volumes provisioned using the `ontap-nas` and `ontap-nas-economy` drivers. Enable the `.snapshot` directory to recover data from snapshots directly.

When you restore a snapshot copy, the existing volume configuration is overwritten. Changes made to volume data after the snapshot copy was created are lost.

In-place volume restoration from a snapshot

Astra Control Provisioner provides rapid, in-place volume restoration from a snapshot using the `TridentActionSnapshotRestore` (TASR) CR. This CR functions as an imperative Kubernetes action and does not persist after the operation completes.

Astra Control Provisioner supports snapshot restore on the `ontap-san`, `ontap-san-economy`, `ontap-nas`, `ontap-nas-flexgroup`, `azure-netapp-files`, `gcp-cvs`, and `solidfire-san` drivers.

Before you begin

You must have a bound PVC and available volume snapshot.

- Verify the PVC status is bound.

```
kubectl get pvc
```

- Verify the volume snapshot is ready to use.

```
kubectl get vs
```

Steps

1. Create the TASR CR. This example creates a CR for PVC `pvc1` and volume snapshot `pvc1-snapshot`.

```
cat tasr-pvc1-snapshot.yaml

apiVersion: v1
kind: TridentActionSnapshotRestore
metadata:
  name: this-doesnt-matter
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. Apply the CR to restore from the snapshot. This example restores from snapshot `pvc1`.

```
kubectl create -f tasr-pvc1-snapshot.yaml

tridentactionsnapshotrestore.trident.netapp.io/this-doesnt-matter
created
```

Results

Astra Control Provisioner restores the data from the snapshot. You can verify the snapshot restore status.

```
kubectl get tasr -o yaml

apiVersion: v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: this-doesnt-matter
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- In most cases, Astra Control Provisioner will not automatically retry the operation in case of failure. You will need to perform the operation again.
- Kubernetes users without admin access might have to be granted permission by the admin to create a TASR CR in their application namespace.

Use the volume snapshot restore ONTAP CLI to to restore a volume to a state recorded in a prior snapshot.

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Delete a PV with associated snapshots

When deleting a Persistent Volume with associated snapshots, the corresponding Trident volume is updated to a “Deleting state”. Remove the volume snapshots to delete the Astra Trident volume.

Deploy a volume snapshot controller

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

Steps

1. Create volume snapshot CRDs.

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

2. Create the snapshot controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

If necessary, open `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` and update `namespace` to your namespace.

Related links

- Volume snapshots
- VolumeSnapshotClass

Copyright information

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

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

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

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

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

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

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

Trademark information

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