Protect applications with Trident protect

Trident

NetApp

February 20, 2025

Table of Contents

Protect applications with Trident protect

Learn about Trident protect

NetApp Trident protect provides advanced application data management capabilities that enhance the functionality and availability of stateful Kubernetes applications backed by NetApp ONTAP storage systems and the NetApp Trident CSI storage provisioner. Trident protect simplifies the management, protection, and movement of containerized workloads across public clouds and on-premises environments. It also offers automation capabilities through its API and CLI.

You can protect applications with Trident protect by creating custom resources (CRs) or by using the Trident protect CLI.

What's next?

You can learn about Trident protect requirements before you install it:

- Trident protect requirements

Install Trident protect

Trident protect requirements

Get started by verifying the readiness of your operational environment, application clusters, applications, and licenses. Ensure that your environment meets these requirements to deploy and operate Trident protect.

Trident protect Kubernetes cluster compatibility

Trident protect is compatible with a wide range of fully managed and self-managed Kubernetes offerings, including:

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes Service (AKS)
- Red Hat OpenShift
- SUSE Rancher
- VMware Tanzu Portfolio
- Upstream Kubernetes

Ensure that the cluster on which you install Trident protect is configured with a running snapshot controller and the related CRDs. To install a snapshot controller, refer to these instructions.

Trident protect storage backend compatibility

Trident protect supports the following storage backends:

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP storage arrays
- Google Cloud NetApp Volumes
- Azure NetApp Files

Ensure that your storage backend meets the following requirements:

- Ensure that NetApp storage connected to the cluster is using Astra Trident 24.02 or newer (Trident 24.10 is recommended).
  - If Astra Trident is older than version 24.06.1 and you plan to use NetApp SnapMirror disaster recovery functionality, you need to manually enable Astra Control Provisioner.
- Ensure that you have the latest Astra Control Provisioner (installed and enabled by default as of Astra Trident 24.06.1).
- Ensure that you have a NetApp ONTAP storage backend.
- Ensure that you have configured an object storage bucket for storing backups.
- Create any application namespaces that you plan to use for applications or application data management operations. Trident protect does not create these namespaces for you; if you specify a nonexistent namespace in a custom resource, the operation will fail.

Requirements for nas-economy volumes

Trident protect supports backup and restore operations to nas-economy volumes. Snapshots, clones, and SnapMirror replication to nas-economy volumes are not currently supported. You need to enable a snapshot directory for each nas-economy volume you plan to use with Trident protect.

Some applications are not compatible with volumes that use a snapshot directory. For these applications, you need to hide the snapshot directory by running the following command on the ONTAP storage system:

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

You can enable the snapshot directory by running the following command for each nas-economy volume, replacing `<volume-UUID>` with the UUID of the volume you want to change:

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

You can enable snapshot directories by default for new volumes by setting the Trident backend configuration option `snapshotDir` to `true`. Existing volumes are not affected.

Protecting data with KubeVirt VMs

Trident protect 24.10 and 24.10.1 and newer have different behavior when you protect applications running on KubeVirt VMs. For both versions, you can enable or disable filesystem freezing and unfreezing during data protection operations.

For all Trident protect versions, to enable or disable automatic freeze functionality in OpenShift environments, you might need to grant the application namespace privileged permissions. For example:

```
oc adm policy add-scc-to-user privileged -z default -n
<application-namespace>
```

Trident protect 24.10

Trident protect 24.10 does not automatically ensure a consistent state for KubeVirt VM filesystems during data protection operations. If you want to protect your KubeVirt VM data using Trident protect 24.10, you need to manually enable the freeze/unfreeze functionality for the filesystems before the data protection operation. This ensures that the filesystems are in a consistent state.

You can configure Trident protect 24.10 to manage the freezing and unfreezing of the VM filesystem during data protection operations by configuring virtualization and then using the following command:

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

Trident protect 24.10.1 and newer

Beginning with Trident protect 24.10.1, Trident protect automatically freezes and unfreezes KubeVirt filesystems during data protection operations. Optionally, you can disable this automatic behavior using the following command:

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

Requirements for SnapMirror replication

NetApp SnapMirror is available for use with Trident protect for the following ONTAP solutions:

- NetApp ASA
- NetApp AFF
- NetApp FAS
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

ONTAP cluster requirements for SnapMirror replication

Ensure your ONTAP cluster meets the following requirements if you plan to use SnapMirror replication:

- **Astra Control Provisioner or Trident**: Astra Control Provisioner or Trident must exist on both the source and destination Kubernetes clusters that utilize ONTAP as a backend. Trident protect supports replication with NetApp SnapMirror technology using storage classes backed by the following drivers:
  - `ontap-nas`
  - `ontap-san`
- **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 considerations for SnapMirror replication

Ensure your environment meets the following requirements if you plan to use storage backend 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 or Trident and SVM**: The peered remote SVMs must be available to Astra Control Provisioner or Trident on the destination cluster.
- **Managed backends**: You need to add and manage ONTAP storage backends in Trident protect to create a replication relationship.
- **NVMe over TCP**: Trident protect does not support NetApp SnapMirror replication for storage backends that are using the NVMe over TCP protocol.

Trident / ONTAP configuration for SnapMirror replication

Trident protect requires that you configure at least one storage backend that supports replication for both the source and destination clusters. If the source and destination clusters are the same, the destination application should use a different storage backend than the source application for the best resiliency.

Install and configure Trident protect

If your environment meets the requirements for Trident protect, you can follow these steps to install Trident protect on your cluster. You can obtain Trident protect from NetApp, or install it from your own private registry. Installing from a private registry is helpful if your cluster cannot access the Internet.

> By default, Trident protect collects support information that helps with any NetApp support cases that you might open, including logs, metrics, and topology information about clusters and managed applications. Trident protect sends these support bundles to NetApp on a daily schedule. You can optionally disable this support bundle collection when you install Trident protect. You can manually generate a support bundle at any time.

Install Trident protect

Install Trident protect from NetApp

Steps

1. Add the Trident Helm repository:

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

2. Install the Trident protect CRDs:

   ```
   helm install trident-protect-crds netapp-trident-protect/trident-
   protect-crds --version 100.2410.1 --create-namespace --namespace
   trident-protect
   ```

3. Use Helm to install Trident protect using one of the following commands. Replace `<name_of_cluster>` with a cluster name, which will be assigned to the cluster and used to identify the cluster's backups and snapshots:

   - Install Trident protect normally:

     ```
     helm install trident-protect netapp-trident-protect/trident-
     protect --set clusterName=<name_of_cluster> --version 100.2410.1
     --create-namespace --namespace trident-protect
     ```

   - Install Trident protect and disable the scheduled daily Trident protect AutoSupport support bundle uploads:

     ```
     helm install trident-protect netapp-trident-protect/trident-
     protect --set autoSupport.enabled=false --set
     clusterName=<name_of_cluster> --version 100.2410.1 --create
     -namespace --namespace trident-protect
     ```

Install Trident protect from a private registry

You can install Trident protect from a private image registry if your Kubernetes cluster is unable to access the Internet. In these examples, replace values in brackets with information from your environment:

Steps

1. Pull the following images to your local machine, update the tags, and then push them to your private registry:

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exechook:24.10.1
netapp/resourcebackup:24.10.1
netapp/resourcerestore:24.10.1
netapp/resourcedelete:24.10.1
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

For example:

```
docker pull netapp/controller:24.10.1
```

```
docker tag netapp/controller:24.10.1 <private-registry-
url>/controller:24.10.1
```

```
docker push <private-registry-url>/controller:24.10.1
```

2. Create the Trident protect system namespace:

```
kubectl create ns trident-protect
```

3. Log in to the registry:

```
helm registry login <private-registry-url> -u <account-id> -p <api-
token>
```

4. Create a pull secret to use for private registry authentication:

```
kubectl create secret docker-registry regcred --docker
-username=<registry-username> --docker-password=<api-token> -n
trident-protect --docker-server=<private-registry-url>
```

5. Add the Trident Helm repository:

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

6. Create a file named `protectValues.yaml`. Ensure that it contains the following Trident protect settings:

```
---
image:
  registry: <private-registry-url>
imagePullSecrets:
  - name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
    - name: regcred
webhooksCleanup:
  imagePullSecrets:
    - name: regcred
```

7. Install the Trident protect CRDs:

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

8. Use Helm to install Trident protect using one of the following commands. Replace `<name_of_cluster>` with a cluster name, which will be assigned to the cluster and used to identify the cluster's backups and snapshots:
   - Install Trident protect normally:

   ```
   helm install trident-protect netapp-trident-protect/trident-
   protect --set clusterName=<name_of_cluster> --version 100.2410.1
   --create-namespace --namespace trident-protect -f
   protectValues.yaml
   ```

   - Install Trident protect and disable the scheduled daily Trident protect AutoSupport support bundle uploads:

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

Specify Trident protect container resource limits

You can use a configuration file to specify resource limits for Trident protect containers after you install Trident protect. Setting resource limits enables you to control how much of the cluster's resources are consumed by Trident protect operations.

Steps

1. Create a file named `resourceLimits.yaml`.
2. Populate the file with resource limit options for Trident protect containers according to the needs of your environment.

   The following example configuration file shows the available settings and contains the default vaules for each resource limit:

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. Apply the values from the `resourceLimits.yaml` file:

```
helm upgrade trident-protect -n trident-protect -f <resourceLimits.yaml>
--reuse-values
```

Install the Trident protect CLI plugin

You can use the Trident protect command line plugin, which is an extension of the Trident `tridentctl` utility, to create and interact with Trident protect custom resources (CRs).

Install the Trident protect CLI plugin

Before using the command line utility, you need to install it on the machine you use to access your cluster. Follow these steps, depending on if your machine uses an x64 or ARM CPU.

Download plugin for Linux AMD64 CPUs

Steps

1. Download the Trident protect CLI plugin:

   ```
   curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-
   protect/releases/download/24.10.1/tridentctl-protect-linux-amd64
   ```

Download plugin for Linux ARM64 CPUs

Steps

1. Download the Trident protect CLI plugin:

   ```
   curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-
   protect/releases/download/24.10.1/tridentctl-protect-linux-arm64
   ```

Download plugin for Mac AMD64 CPUs

Steps

1. Download the Trident protect CLI plugin:

   ```
   curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-
   protect/releases/download/24.10.1/tridentctl-protect-macos-amd64
   ```

Download plugin for Mac ARM64 CPUs

Steps

1. Download the Trident protect CLI plugin:

   ```
   curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-
   protect/releases/download/24.10.1/tridentctl-protect-macos-arm64
   ```

2. Enable execute permissions for the plugin binary:

   ```
   chmod +x tridentctl-protect
   ```

3. Copy the plugin binary to a location that is defined in your PATH variable. For example, `/usr/bin` or `/usr/local/bin` (you might need elevated privileges):

   ```
   cp ./tridentctl-protect /usr/local/bin/
   ```

4. Optionally, you can copy the plugin binary to a location in your home directory. In this case, it is recommended to ensure the location is part of your PATH variable:

```
cp ./tridentctl-protect ~/bin/
```

Copying the plugin to a location in your PATH variable enables you to use the plugin by typing `tridentctl-protect` or `tridentctl protect` from any location.

View Trident CLI plugin help

You can use the built-in plugin help features to get detailed help on the capabilities of the plugin:

Steps

1. Use the help function to view usage guidance:

```
tridentctl-protect help
```

Enable command auto-completion

After you have installed the Trident protect CLI plugin, you can enable auto-completion for certain commands.

Enable auto-completion for the Bash shell

Steps

1. Download the completion script:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/tridentctl-completion.bash
```

2. Make a new directory in your home directory to contain the script:

```
mkdir -p ~/.bash/completions
```

3. Move the downloaded script to the `~/.bash/completions` directory:

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. Add the following line to the `~/.bashrc` file in your home directory:

```
source ~/.bash/completions/tridentctl-completion.bash
```

Enable auto-completion for the Z shell

Steps

1. Download the completion script:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/tridentctl-completion.zsh
```

2. Make a new directory in your home directory to contain the script:

```
mkdir -p ~/.zsh/completions
```

3. Move the downloaded script to the `~/.zsh/completions` directory:

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. Add the following line to the `~/.zprofile` file in your home directory:

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

Result

Upon your next shell login, you can use command auto-completion with the tridentctl-protect plugin.

Manage Trident protect

Manage Trident protect authorization and access control

Trident protect uses the Kubernetes model of role-based access control (RBAC). By default, Trident protect provides a single system namespace and its associated default service account. If you have an organization with many users or specific security needs, you can use the RBAC features of Trident protect to gain more granular control over access to resources and namespaces.

The cluster administrator always has access to resources in the default `trident-protect` namespace, and can also access resources in all other namespaces. To control access to resources and applications, you need to create additional namespaces and add resources and applications to those namespaces.

Note that no users can create application data management CRs in the default `trident-protect` namespace. You need to create application data management CRs in an application namespace (as a best practice, create application data management CRs in the same namespace as their associated application).

Only administrators should have access to privileged Trident protect custom resource objects, which include:

- **AppVault**: Requires bucket credential data
- **AutoSupportBundle**: Collects metrics, logs, and other sensitive Trident protect data
- **AutoSupportBundleSchedule**: Manages log collection schedules

As a best practice, use RBAC to restrict access to privileged objects to administrators.

For more information about how RBAC regulates access to resources and namespaces, refer to the Kubernetes RBAC documentation.

Fore information about service accounts, refer to the Kubernetes service account documentation.

Example: Manage access for two groups of users

For example, an organization has a cluster administrator, a group of engineering users, and a group of marketing users. The cluster administrator would complete the following tasks to create an environment where the engineering group and the marketing group each have access to only the resources assigned to their respective namespaces.

Step 1: Create a namespace to contain resources for each group

Creating a namespace enables you to logically separate resources and better control who has access to those resources.

Steps

1. Create a namespace for the engineering group:

```
kubectl create ns engineering-ns
```

2. Create a namespace for the marketing group:

```
kubectl create ns marketing-ns
```

Step 2: Create new service accounts to interact with resources in each namespace

Each new namespace you create comes with a default service account, but you should create a service account for each group of users so that you can further divide privileges between groups in the future if necessary.

Steps

1. Create a service account for the engineering group:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. Create a service account for the marketing group:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

Step 3: Create a secret for each new service account

A service account secret is used to authenticate with the service account, and can easily be deleted and recreated if compromised.

Steps

1. Create a secret for the engineering service account:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. Create a secret for the marketing service account:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

Step 4: Create a RoleBinding object to bind the ClusterRole object to each new service account

A default ClusterRole object is created when you install Trident protect. You can bind this ClusterRole to the service account by creating and applying a RoleBinding object.

Steps

1. Bind the ClusterRole to the engineering service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. Bind the ClusterRole to the marketing service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

Step 5: Test permissions

Test that the permissions are correct.

Steps

1. Confirm that engineering users can access engineering resources:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. Confirm that engineering users cannot access marketing resources:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

Step 6: Grant access to AppVault objects

To perform data management tasks such as backups and snapshots, the cluster administrator needs to grant access to AppVault objects to individual users.

Steps

1. Create and apply an AppVault and secret combination YAML file that grants a user access to an AppVault. For example, the following CR grants access to an AppVault to the user `eng-user`:

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. Create and apply a Role CR to enable cluster administrators to grant access to specific resources in a namespace. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. Create and apply a RoleBinding CR to bind the permissions to the user eng-user. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. Verify that the permissions are correct.
   a. Attempt to retrieve AppVault object information for all namespaces:

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   You should see output similar to the following:

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. Test to see if the user can get the AppVault information that they now have permission to access:

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

You should see output similar to the following:

```
yes
```

Result

The users you have granted AppVault permissions to should be able to use authorized AppVault objects for application data management operations, and should not be able to access any resources outside of the assigned namespaces or create new resources that they do not have access to.

Generate a Trident protect support bundle

Trident protect enables administrators to generate bundles that include information useful to NetApp Support, including logs, metrics, and topology information about the clusters and apps under management. If you are connected to the Internet, you can upload support bundles to the NetApp Support Site (NSS) using a custom resource (CR) file.

Create a support bundle using a CR

Steps

1. Create the custom resource (CR) file and name it (for example, `trident-protect-support-bundle.yaml`).
2. Configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.triggerType**: (*Required*) Determines whether the support bundle is generated immediately, or scheduled. Scheduled bundle generation happens at 12AM UTC. Possible values:
     - Scheduled
     - Manual
   - **spec.uploadEnabled**: (*Optional*) Controls whether the support bundle should be uploaded to the NetApp Support Site after it is generated. If not specified, defaults to `false`. Possible values:
     - true
     - false (default)
   - **spec.dataWindowStart**: (*Optional*) A date string in RFC 3339 format that specifies the date and time that the window of included data in the support bundle should begin. If not specified, defaults to 24 hours ago. The earliest window date you can specify is 7 days ago.

     Example YAML:

     ```yaml
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: AutoSupportBundle
     metadata:
       name: trident-protect-support-bundle
     spec:
       triggerType: Manual
       uploadEnabled: true
       dataWindowStart: 2024-05-05T12:30:00Z
     ```
3. After you populate the `astra-support-bundle.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-support-bundle.yaml
   ```

Create a support bundle using the CLI

Steps

1. Create the support bundle, replacing values in brackets with information from your environment. The `trigger-type` determines whether the bundle is created immediately or if creation time is dictated by the schedule, and can be `Manual` or `Scheduled`. The default setting is `Manual`.

   For example:

```
tridentctl-protect create autosupportbundle <my_bundle_name>
--trigger-type <trigger_type>
```

Upgrade Trident protect

You can upgrade Trident protect to the latest version to take advantage of new features or bug fixes.

To upgrade Trident protect, perform the following steps.

Steps

1. Update the Trident Helm repository:

   ```
   helm repo update
   ```

2. Upgrade the Trident protect CRDs:

   ```
   helm upgrade trident-protect-crds netapp-trident-protect/trident-
   protect-crds --version 100.2410.1  --namespace trident-protect
   ```

3. Upgrade Trident protect:

   ```
   helm upgrade trident-protect netapp-trident-protect/trident-protect
   --version 100.2410.1 --namespace trident-protect
   ```

Manage and protect applications

Use Trident protect AppVault objects to manage buckets

The bucket custom resource (CR) for Trident protect is known as an AppVault. AppVault objects are the declarative Kubernetes workflow representation of a storage bucket. An AppVault CR contains the configurations necessary for a bucket to be used in protection operations, such as backups, snapshots, restore operations, and SnapMirror replication. Only administrators can create AppVaults.

Key generation and AppVault definition examples

When defining an AppVault CR, you need to include credentials to access the resources hosted by the provider. How you generate the keys for the credentials will differ depending on the provider. The following are command line key generation examples for several providers, followed by example AppVault definitions for each provider.

Key generation examples

You can use the following examples to create keys for the credentials of each cloud provider.

Google Cloud

```
kubectl create secret generic <secret-name> --from-file=credentials
=<mycreds-file.json> -n trident-protect
```

Amazon S3 (AWS)

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

Microsoft Azure

```
kubectl create secret generic <secret-name> --from-literal=accountKey
=<secret-name> -n trident-protect
```

Generic S3

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

ONTAP S3

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

StorageGrid S3

```
kubectl create secret generic  <secret-name> --from-literal=
accessKeyID=<objectstorage-accesskey> --from-literal=secretAccessKey
=<generic-s3-trident-protect-src-bucket-secret> -n trident-protect
```

AppVault CR examples

You can use the following CR examples to create AppVault objects for each cloud provider.

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Amazon S3 (AWS)

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

Generic S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

StorageGrid S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-
971f-ac4a83621922
  namespace: trident-protect
spec:
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

AppVault creation examples using the Trident protect CLI

You can use the following CLI command examples to create AppVault CRs for each provider.

Google Cloud

```
tridentctl-protect create vault GCP my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3 (AWS)

```
tridentctl-protect create vault AWS <vault-name> --bucket <bucket-name>
--secret  <secret-name>  --endpoint <s3-endpoint>
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> --account <account-
name> --bucket <bucket-name> --secret <secret-name>
```

Generic S3

```
tridentctl-protect create vault GenericS3 <vault-name> --bucket
<bucket-name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

StorageGrid S3

```
tridentctl-protect create vault StorageGridS3 s3vault --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

Use the AppVault browser to view AppVault information

You can use the Trident protect CLI plugin to view information about AppVault objects that have been created on the cluster.

Steps

1. View the contents of an AppVault object:

```
tridentctl-protect get appvaultcontent gcp-vault --show-resources all
```

Example output:

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+-----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+-----------------------------
+---------------------------+
```

2. Optionally, to see the AppVaultPath for each resource, use the flag `--show-paths`.

   The cluster name in the first column of the table is only available if a cluster name was specified in the Trident protect helm installation. For example: `--set clusterName=production1`.

Remove an AppVault

You can remove an AppVault object at any time.

Do not remove the `finalizers` key in the AppVault CR before deleting the AppVault object. If you do so, it can result in residual data in the AppVault bucket and orphaned resources in the cluster.

Before you begin

Ensure that you have deleted all snapshots and backups stored in the associated bucket.

Remove an AppVault using the Kubernetes CLI

1. Remove the AppVault object, replacing `appvault_name` with the name of the AppVault object to remove:

```
kubectl delete appvault <appvault_name> -n trident-protect
```

Remove an AppVault using the Trident protect CLI

1. Remove the AppVault object, replacing `appvault_name` with the name of the AppVault object to remove:

```
tridentctl-protect delete appvault <appvault_name> -n trident-
protect
```

Define an application for management with Trident protect

You can define an application that you want to manage with Trident protect by creating an application CR and an associated AppVault CR.

Create an AppVault CR

You need to create an AppVault CR that will be used when performing data protection operations on the application, and the AppVault CR needs to reside on the cluster where Trident protect is installed. The AppVault CR is specific to your environment; for examples of AppVault CRs, refer to AppVault custom resources.

Define an application

You need to define each application that you want to manage with Trident protect. You can define an application for management by either manually creating an application CR or by using the Trident protect CLI.

Add an application using a CR

Steps

1. Create the destination application CR file:

   a. Create the custom resource (CR) file and name it (for example, `maria-app.yaml`).

   b. Configure the following attributes:

      - **metadata.name**: (*Required*) The name of the application custom resource. Note the name you choose because other CR files needed for protection operations refer to this value.
      - **spec.includedNamespaces**: (*Required*) Use namespace labels or a namespace name to specify namespaces that the application resources exist in. The application namespace must be part of this list.

        Example YAML:

        ```
        ---
        apiVersion: protect.trident.netapp.io/v1
        kind: Application
        metadata:
          name: maria
          namespace: my-app-namespace
        spec:
          includedNamespaces:
            - namespace: my-app-namespace
        ```

2. After you create the application CR to match your environment, apply the CR. For example:

   ```
   kubectl apply -f maria-app.yaml
   ```

Add an application using the CLI

Steps

1. Create and apply the application definition, replacing values in brackets with information from your environment. You can include namespaces and resources in the application definition using comma-separated lists with the arguments shown in the following example:

   ```
   tridentctl-protect create application <my_new_app_cr_name>
   --namespaces <namespaces_to_include> --csr
   <cluster_scoped_resources_to_include> --namespace <my-app-namespace>
   ```

Protect applications using Trident protect

You can protect all apps managed by Trident protect by taking snapshots and backups using an automated protection policy or on an ad-hoc basis.

You can configure Trident protect to freeze and unfreeze filesystems during data protection operations. Learn more about configuring filesystem freezing with Trident protect.

Create an on-demand snapshot

You can create an on-demand snapshot at any time.

Create a snapshot using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: The Kubernetes name of the application to snapshot.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents (metadata) should be stored.
   - **spec.reclaimPolicy**: (*Optional*) Defines what happens to the AppArchive of a snapshot when the snapshot CR is deleted. This means that even when set to `Retain`, the snapshot will be deleted. Valid options:
     - `Retain` (default)
     - `Delete`

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: Snapshot
     metadata:
       namespace: my-app-namespace
       name: my-cr-name
     spec:
       applicationRef: my-application
       appVaultRef: appvault-name
       reclaimPolicy: Delete
     ```

3. After you populate the `trident-protect-snapshot-cr.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-snapshot-cr.yaml
   ```

Create a snapshot using the CLI

Steps

1. Create the snapshot, replacing values in brackets with information from your environment. For example:

   ```
   tridentctl-protect create snapshot <my_snapshot_name> --appvault
   <my_appvault_name> --app <name_of_app_to_snapshot> -n
   <application_namespace>
   ```

Create an on-demand backup

You can back up an app at any time.

Create a backup using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) The Kubernetes name of the application to back up.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents should be stored.
   - **spec.dataMover**: (*Optional*) A string indicating which backup tool to use for the backup operation. Possible values (case sensitive):
     - `Restic`
     - `Kopia` (default)
   - **spec.reclaimPolicy**: (*Optional*) Defines what happens to a backup when released from its claim. Possible values:
     - `Delete`
     - `Retain` (default)
   - **Spec.snapshotRef**: (*Optional*): Name of the snapshot to use as the source of the backup. If not provided, a temporary snapshot will be created and backed up.

   ```yaml
   ---
   apiVersion: protect.trident.netapp.io/v1
   kind: Backup
   metadata:
     namespace: my-app-namespace
     name: my-cr-name
   spec:
     applicationRef: my-application
     appVaultRef: appvault-name
     dataMover: Kopia
   ```

3. After you populate the `trident-protect-backup-cr.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-backup-cr.yaml
   ```

Create a backup using the CLI

Steps

1. Create the backup, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up> -n
<application_namespace>
```

Create a data protection schedule

A protection policy protects an app by creating snapshots, backups, or both at a defined schedule. You can choose to create snapshots and backups hourly, daily, weekly, and monthly, and you can specify the number of copies to retain.

Create a schedule using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-schedule-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.dataMover**: (*Optional*) A string indicating which backup tool to use for the backup operation. Possible values (case sensitive):
     - `Restic`
     - `Kopia` (default)
   - **spec.applicationRef**: The Kubernetes name of the application to back up.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents should be stored.
   - **spec.backupRetention**: The number of backups to retain. Zero indicates that no backups should be created.
   - **spec.snapshotRetention**: The number of snapshots to retain. Zero indicates that no snapshots should be created.
   - **spec.granularity**: The frequency at which the schedule should run. Possible values, along with required associated fields:
     - `hourly` (requires that you specify `spec.minute`)
     - `daily` (requires that you specify `spec.minute` and `spec.hour`)
     - `weekly` (requires that you specify `spec.minute, spec.hour`, and `spec.dayOfWeek`)
     - `monthly` (requires that you specify `spec.minute, spec.hour`, and `spec.dayOfMonth`)
   - **spec.dayOfMonth**: (*Optional*) The day of the month (1 - 31) that the schedule should run. This field is required if the granularity is set to `monthly`.
   - **spec.dayOfWeek**: (*Optional*) The day of the week (0 - 7) that the schedule should run. Values of 0 or 7 indicate Sunday. This field is required if the granularity is set to `weekly`.
   - **spec.hour**: (*Optional*) The hour of the day (0 - 23) that the schedule should run. This field is required if the granularity is set to `daily`, `weekly`, or `monthly`.
   - **spec.minute**: (*Optional*) The minute of the hour (0 - 59) that the schedule should run. This field is required if the granularity is set to `hourly`, `daily`, `weekly`, or `monthly`.

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"
```

3. After you populate the `trident-protect-schedule-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

Create a schedule using the CLI

Steps

1. Create the protection schedule, replacing values in brackets with information from your environment. For example:

   You can use `tridentctl-protect create schedule --help` to view detailed help information for this command.

```
tridentctl-protect create schedule <my_schedule_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> --backup
-retention <how_many_backups_to_retain> --data-mover
<kopia_or_restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain> -n <application_namespace>
```

Delete a snapshot

Delete the scheduled or on-demand snapshots that you no longer need.

Steps

1. Remove the snapshot CR associated with the snapshot:

   ```
   kubectl delete snapshot <snapshot_name> -n my-app-namespace
   ```

Delete a backup

Delete the scheduled or on-demand backups that you no longer need.

Steps

1. Remove the backup CR associated with the backup:

   ```
   kubectl delete backup <backup_name> -n my-app-namespace
   ```

Check the status of a backup operation

You can use the command line to check the status of a backup operation that is in progress, has completed, or has failed.

Steps

1. Use the following command to retrieve status of the backup operation, replacing values in brackes with information from your environment:

   ```
   kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
   ='{.status}'
   ```

Enable backup and restore for azure-netapp-files (ANF) operations

If you have installed Trident protect, you can enable space-efficient backup and restore functionality for storage backends that use the azure-netapp-files storage class and were created prior to Trident 24.06. This funtionality works with NFSv4 volumes and does not consume additional space from the capacity pool.

Before you begin

Ensure the following:

- You have installed Trident protect.
- You have defined an application in Trident protect. This application will have limited protection functionality until you complete this procedure.
- You have `azure-netapp-files` selected as the default storage class for your storage backend.

Expand for configuration steps

1. Do the following in Trident if the ANF volume was created prior to upgrading to Trident 24.10:
   a. Enable the snapshot directory for each PV that is azure-netapp-files based and associated with the application:

      ```
      tridentctl update volume <pv name> --snapshot-dir=true -n trident
      ```

   b. Confirm that the snapshot directory has been enabled for each associated PV:

      ```
      tridentctl get volume <pv name> -n trident -o yaml | grep
      snapshotDir
      ```

      Response:

      ```
      snapshotDirectory: "true"
      ```

      When the snapshot directory is not enabled, Trident protect chooses the regular backup functionality, which temporarily consumes space in the capacity pool during the backup process. In this case, ensure that sufficient space is available in the capacity pool to create a temporary volume of the size of the volume being backed up.

Result

The application is ready for backup and restore using Trident protect. Each PVC is also available to be used by other applications for backups and restores.

Restore applications using Trident protect

You can use Trident protect to restore your application from a snapshot or backup. Restoring from an existing snapshot will be faster when restoring the application to the same cluster.

When you restore an application, all execution hooks configured for the application are restored with the app. If a post-restore execution hook is present, it runs automatically as part of the restore operation.

Namespace annotations and labels during restore and failover operations

During restore and failover operations, labels and annotations in the destination namespace are made to match the labels and annotations in the source namespace. Labels or annotations from the source namespace that don't exist in the destination namespace are added, and any labels or annotations that already exist are overwritten to match the value from the source namespace. Labels or annotations that exist only on the destination namespace remain unchanged.

If you use RedHat OpenShift, it's important to note the critical role of namespace annotations in OpenShift environments. Namespace annotations ensure that restored pods adhere to the appropriate permissions and security configurations defined by OpenShift security context constraints (SCCs) and can access volumes without permission issues. For more information, refer to the OpenShift security context constraints documentation.

You can prevent specific annotations in the destination namespace from being overwritten by setting the Kubernetes environment variable `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` before you perform the restore or failover operation. For example:

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

If you installed the source application using Helm with the `--create-namespace` flag, special treatment is given to the `name` label key. During the restore or failover process, Trident protect copies this label to the destination namespace, but updates the value to the destination namespace value if the value from source matches the source namespace. If this value doesn't match the source namespace it is copied to the destination namespace with no changes.

Example

The following example presents a source and destination namespace, each with different annotations and labels. You can see the state of the destination namespace before and after the operation, and how the annotations and labels are combined or overwritten in the destination namespace.

Before the restore or failover operation

The following table illustrates the state of the example source and destination namespaces before the restore or failover operation:

Namespace	Annotations	Labels
Namespace ns-1 (source)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• environment=production • compliance=hipaa • name=ns-1
Namespace ns-2 (destination)	• annotation.one/key: "true" • annotation.three/key: "false"	• role=database

After the restore operation

The following table illustrates the state of the example destination namespace after the restore or failover operation. Some keys have been added, some have been overwritten, and the `name` label has been updated to match the destination namespace:

Namespace	Annotations	Labels
Namespace ns-2 (destination)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• name=ns-2 • compliance=hipaa • environment=production • role=database

Restore from a backup to a different namespace

When you restore a backup to a different namespace using a BackupRestore CR, Trident protect restores the application in a new namespace and creates an application CR for the restored application. To protect the restored application, create on-demand backups or snapshots, or establish a protection schedule.

> Restoring a backup to a different namespace with existing resources will not alter any resources that share names with those in the backup. To restore all resources in the backup, either delete and re-create the target namespace, or restore the backup to a new namespace.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.
   - **spec.storageClassMapping**: The mapping of the source storage class of the restore operation to the destination storage class. Replace `destinationStorageClass` and `sourceStorageClass` with information from your environment.

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each

element (group, kind, version) match as an AND operation.

- **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
- **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
- **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
- **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-backup-restore-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

Use the CLI

Steps

1. Restore the backup to a different namespace, replacing values in brackets with information from your environment. The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format `source1:dest1,source2:dest2`. For example:

```
tridentctl-protect create backuprestore <my_restore_name> --backup
<backup_namespace>/<backup_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping> -n <application_namespace>
```

Restore from a backup to the original namespace

You can restore a backup to the original namespace at any time.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-backup-ipr-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.

     For example:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupInplaceRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
       - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For

example: "trident.netapp.io/os=linux".

For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-backup-ipr-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

Use the CLI

Steps

1. Restore the backup to the original namespace, replacing values in brackets with information from your environment. The `backup` argument uses a namespace and backup name in the format `<namespace>/<name>`. For example:

```
tridentctl-protect create backupinplacerestore <my_restore_name>
--backup <namespace/backup_to_restore> -n <application_namespace>
```

Restore from a backup to a different cluster

You can restore a backup to a different cluster if there is an issue with the original cluster.

Before you begin

Ensure the following prerequisites are met:

- The destination cluster has Trident protect installed.

- The destination cluster has access to the bucket path of the same AppVault as the source cluster, where the backup is stored.

Steps

1. Check the availability of the AppVault CR on the destination cluster using Trident protect CLI plugin:

   ```
   tridentctl-protect get appvault --context <destination_cluster_name>
   ```

   > Ensure that the namespace intended for the application restore exists on the destination cluster.

2. View the backup contents of the available AppVault from the destination cluster:

   ```
   tridentctl-protect get appvaultcontent <appvault_name> --show-resources
   backup --show-paths --context <destination_cluster_name>
   ```

   Running this command displays the available backups in the AppVault, including their originating clusters, corresponding application names, timestamps, and archive paths.

   Example output:

   ```
   +-------------+-----------+--------+-----------------
   +-------------------------+-------------+
   |   CLUSTER   |    APP    |  TYPE  |      NAME       |         TIMESTAMP
   |    PATH     |
   +-------------+-----------+--------+-----------------
   +-------------------------+-------------+
   | production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
   08:37:40 (UTC)| backuppath1 |
   | production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
   08:37:40 (UTC)| backuppath2 |
   +-------------+-----------+--------+-----------------
   +-------------------------+-------------+
   ```

3. Restore the application to the destination cluster using the AppVault name and archive path:

Use a CR

4. Create the custom resource (CR) file and name it `trident-protect-backup-restore-cr.yaml`.
5. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the backup contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the backup contents are stored. You can use the following command to find this path:

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     If BackupRestore CR is not available, you can use the command mentioned in step 2 to view the backup contents.

   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

     For example:

     ```
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-backup-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

6. After you populate the `trident-protect-backup-restore-cr.yaml` file with the correct values, apply the CR:

   ```
   kubectl apply -f trident-protect-backup-restore-cr.yaml
   ```

Use the CLI

4. Use the following command to restore the application, replacing values in brackets with information from your environment. The namespace-mapping argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format source1:dest1,source2:dest2. For example:

```
tridentctl-protect create backuprestore <restore_name> --namespace
-mapping <source_to_destination_namespace_mapping> --appvault
<appvault_name> --path <backup_path> -n <application_namespace>
--context <destination_cluster_name>
```

Restore from a snapshot to a different namespace

You can restore data from a snapshot using a custom resource (CR) file either to a different namespace or the original source namespace. When you restore a snapshot to a different namespace using a SnapshotRestore CR, Trident protect restores the application in a new namespace and creates an application CR for the restored application. To protect the restored application, create on-demand backups or snapshots, or establish a protection schedule.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

     ```
     kubectl get snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.
   - **spec.storageClassMapping**: The mapping of the source storage class of the restore operation to the destination storage class. Replace `destinationStorageClass` and `sourceStorageClass` with information from your environment.

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define

multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.

- **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
- **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
- **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
- **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-restore-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

Use the CLI

Steps

1. Restore the snapshot to a different namespace, replacing values in brackets with information from your environment.
   - The `snapshot` argument uses a namespace and snapshot name in the format `<namespace>/<name>`.

- The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format `source1:dest1,source2:dest2`.

  For example:

  ```
  tridentctl-protect create snapshotrestore <my_restore_name>
  --snapshot <namespace/snapshot_to_restore> --namespace-mapping
  <source_to_destination_namespace_mapping> -n
  <application_namespace>
  ```

Restore from a snapshot to the original namespace

You can restore a snapshot to the original namespace at any time.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-ipr-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

     ```
     kubectl get snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotInplaceRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
         appArchivePath: my-snapshot-path
     ```

3. (*Optional*) If you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `Include` or `Exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.
       - **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
       - **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

For example:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-ipr-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

Use the CLI

Steps

1. Restore the snapshot to the original namespace, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create snapshotinplacerestore <my_restore_name>
--snapshot <snapshot_to_restore> -n <application_namespace>
```

Check the status of a restore operation

You can use the command line to check the status of a restore operation that is in progress, has completed, or has failed.

Steps

1. Use the following command to retrieve status of the restore operation, replacing values in brackes with information from your environment:

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

Replicate applications using NetApp SnapMirror and Trident protect

Using Trident protect, you can use the asynchronous replication capabilities of NetApp SnapMirror technology to replicate data and application changes from one storage backend to another, on the same cluster or between different clusters.

Namespace annotations and labels during restore and failover operations

During restore and failover operations, labels and annotations in the destination namespace are made to match the labels and annotations in the source namespace. Labels or annotations from the source namespace that don't exist in the destination namespace are added, and any labels or annotations that already exist are overwritten to match the value from the source namespace. Labels or annotations that exist only on the destination namespace remain unchanged.

> If you use RedHat OpenShift, it's important to note the critical role of namespace annotations in OpenShift environments. Namespace annotations ensure that restored pods adhere to the appropriate permissions and security configurations defined by OpenShift security context constraints (SCCs) and can access volumes without permission issues. For more information, refer to the OpenShift security context constraints documentation.

You can prevent specific annotations in the destination namespace from being overwritten by setting the Kubernetes environment variable `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` before you perform the restore or failover operation. For example:

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

If you installed the source application using Helm with the `--create-namespace` flag, special treatment is given to the `name` label key. During the restore or failover process, Trident protect copies this label to the destination namespace, but updates the value to the destination namespace value if the value from source matches the source namespace. If this value doesn't match the source namespace it is copied to the destination namespace with no changes.

Example

The following example presents a source and destination namespace, each with different annotations and labels. You can see the state of the destination namespace before and after the operation, and how the annotations and labels are combined or overwritten in the destination namespace.

Before the restore or failover operation

The following table illustrates the state of the example source and destination namespaces before the restore or failover operation:

Namespace	Annotations	Labels
Namespace ns-1 (source)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• environment=production • compliance=hipaa • name=ns-1
Namespace ns-2 (destination)	• annotation.one/key: "true" • annotation.three/key: "false"	• role=database

After the restore operation

The following table illustrates the state of the example destination namespace after the restore or failover operation. Some keys have been added, some have been overwritten, and the `name` label has been updated to match the destination namespace:

Namespace	Annotations	Labels
Namespace ns-2 (destination)	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• name=ns-2 • compliance=hipaa • environment=production • role=database

You can configure Trident protect to freeze and unfreeze filesystems during data protection operations. Learn more about configuring filesystem freezing with Trident protect.

Set up a replication relationship

Setting up a replication relationship involves the following:

- Choosing how frequently you want Trident protect to take an app snapshot (which includes the app's Kubernetes resources as well as the volume snapshots for each of the app's volumes)
- Choosing the replication schedule (includes Kubernetes resources as well as persistent volume data)
- Setting the time for the snapshot to be taken

Steps

1. Create an AppVault for the source application on the source cluster. Depending on your storage provider, modify an example in AppVault custom resources to fit your environment:

Create an AppVault using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-appvault-primary-source.yaml`).

b. Configure the following attributes:

   - **metadata.name**: (*Required*) The name of the AppVault custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
   - **spec.providerConfig**: (*Required*) Stores the configuration necessary to access the AppVault using the specified provider. Choose a bucketName and any other necessary details for your provider. Make note of the values you choose, because other CR files needed for a replication relationship refer to these values. Refer to AppVault custom resources for examples of AppVault CRs with other providers.
   - **spec.providerCredentials**: (*Required*) Stores references to any credential required to access the AppVault using the specified provider.
     - **spec.providerCredentials.valueFromSecret**: (*Required*) Indicates that the credential value should come from a secret.
       - **key**: (*Required*) The valid key of the secret to select from.
       - **name**: (*Required*) Name of the secret containing the value for this field. Must be in the same namespace.
     - **spec.providerCredentials.secretAccessKey**: (*Required*) The access key used to access the provider. The **name** should match **spec.providerCredentials.valueFromSecret.name**.
   - **spec.providerType**: (*Required*) Determines what provides the backup; for example, NetApp ONTAP S3, generic S3, Google Cloud, or Microsoft Azure. Possible values:
     - aws
     - azure
     - gcp
     - generic-s3
     - ontap-s3
     - storagegrid-s3

c. After you populate the `trident-protect-appvault-primary-source.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

Create an AppVault using the CLI

a. Create the AppVault, replacing values in brackets with information from your environment:

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name>
```

2. Create the source application CR:

Create the source application using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-app-source.yaml`).

b. Configure the following attributes:

   - **metadata.name**: (*Required*) The name of the application custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
   - **spec.includedNamespaces**: (*Required*) An array of namespaces and associated labels. Use namespace names and optionally narrow the scope of the namespaces with labels to specify resources that exist in the namespaces listed here. The application namespace must be part of this array.

     Example YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. After you populate the `trident-protect-app-source.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-
namespace
```

Create the source application using the CLI

a. Create the source application. For example:

```
tridentctl-protect create app <my-app-name> --namespaces
<namespaces-to-be-included> -n <my-app-namespace>
```

3. Optionally, take a snapshot of the source application. This snapshot is used as the basis for the application on the destination cluster. If you skip this step, you'll need to wait for the next scheduled snapshot to run so that you have a recent snapshot.

Take a snapshot using a CR

a. Create a replication schedule for the source application:

   i. Create the custom resource (CR) file and name it (for example, `trident-protect-schedule.yaml`).

   ii. Configure the following attributes:

   - **metadata.name**: (*Required*) The name of the schedule custom resource.
   - **spec.AppVaultRef**: (*Required*) This value must match the metadata.name field of the AppVault for the source application.
   - **spec.ApplicationRef**: (*Required*) This value must match the metadata.name field of the source application CR.
   - **spec.backupRetention**: (*Required*) This field is required, and the value must be set to 0.
   - **spec.enabled**: Must be set to true.
   - **spec.granularity**: Must be set to `Custom`.
   - **spec.recurrenceRule**: Define a start date in UTC time and a recurrence interval.
   - **spec.snapshotRetention**: Must be set to 2.

   Example YAML:

```yaml
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-
6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-
04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

   iii. After you populate the `trident-protect-schedule.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-
namespace
```

Take a snapshot using the CLI

a. Create the snapshot, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

4. Create a source application AppVault CR on the destination cluster that is identical to the AppVault CR you applied on the source cluster and name it (for example, `trident-protect-appvault-primary-destination.yaml`).
5. Apply the CR:

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
my-app-namespace
```

6. Create an AppVault for the destination application on the destination cluster. Depending on your storage provider, modify an example in AppVault custom resources to fit your environment:
   a. Create the custom resource (CR) file and name it (for example, `trident-protect-appvault-secondary-destination.yaml`).
   b. Configure the following attributes:
      - **metadata.name**: (*Required*) The name of the AppVault custom resource. Make note of the name you choose, because other CR files needed for a replication relationship refer to this value.
      - **spec.providerConfig**: (*Required*) Stores the configuration necessary to access the AppVault using the specified provider. Choose a `bucketName` and any other necessary details for your provider. Make note of the values you choose, because other CR files needed for a replication relationship refer to these values. Refer to AppVault custom resources for examples of AppVault CRs with other providers.
      - **spec.providerCredentials**: (*Required*) Stores references to any credential required to access the AppVault using the specified provider.
        - **spec.providerCredentials.valueFromSecret**: (*Required*) Indicates that the credential value should come from a secret.
          - **key**: (*Required*) The valid key of the secret to select from.
          - **name**: (*Required*) Name of the secret containing the value for this field. Must be in the same namespace.
        - **spec.providerCredentials.secretAccessKey**: (*Required*) The access key used to access the provider. The **name** should match **spec.providerCredentials.valueFromSecret.name**.

- **spec.providerType**: (*Required*) Determines what provides the backup; for example, NetApp ONTAP S3, generic S3, Google Cloud, or Microsoft Azure. Possible values:
    - aws
    - azure
    - gcp
    - generic-s3
    - ontap-s3
    - storagegrid-s3

c. After you populate the `trident-protect-appvault-secondary-destination.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n my-app-namespace
```

7. Create an AppMirrorRelationship CR file:

Create an AppMirrorRelationship using a CR

a. Create the custom resource (CR) file and name it (for example, `trident-protect-relationship.yaml`).

b. Configure the following attributes:

- **metadata.name:** (Required) The name of the AppMirrorRelationship custom resource.
- **spec.destinationAppVaultRef**: (*Required*) This value must match the name of the AppVault for the destination application on the destination cluster.
- **spec.namespaceMapping**: (*Required*) The destination and source namespaces must match the application namespace defined in the respective application CR.
- **spec.sourceAppVaultRef**: (*Required*) This value must match the name of the AppVault for the source application.
- **spec.sourceApplicationName**: (*Required*) This value must match the name of the source application you defined in the source application CR.
- **spec.storageClassName**: (*Required*) Choose the name of a valid storage class on the cluster. The storage class must be linked to an ONTAP storage VM that is peered with the source environment.
- **spec.recurrenceRule**: Define a start date in UTC time and a recurrence interval.

  Example YAML:

```yaml
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-
bucket-8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. After you populate the `trident-protect-relationship.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

Create an AppMirrorRelationship using the CLI

a. Create and apply the AppMirrorRelationship object, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --recurrence-rule <rule> --source-app
<my_source_app> --source-app-vault <my_source_app_vault> -n
<application_namespace>
```

8. (*Optional*) Check the state and status of the replication relationship:

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

Fail over to destination cluster

Using Trident protect, you can fail over replicated applications to a destination cluster. This procedure stops the replication relationship and brings the app online on the destination cluster. Trident protect does not stop the app on the source cluster if it was operational.

Steps

1. Open the AppMirrorRelationship CR file (for example, `trident-protect-relationship.yaml`) and change the value of **spec.desiredState** to `Promoted`.
2. Save the CR file.
3. Apply the CR:

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. (*Optional*) Create any protection schedules that you need on the failed over application.
5. (*Optional*) Check the state and status of the replication relationship:

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

Resync a failed over replication relationship

The resync operation re-establishes the replication relationship. After you perform a resync operation, the original source application becomes the running application, and any changes made to the running application on the destination cluster are discarded.

The process stops the app on the destination cluster before re-establishing replication.

Any data written to the destination application during failover will be lost.

Steps

1. Create a snapshot of the source application.
2. Open the AppMirrorRelationship CR file (for example, `trident-protect-relationship.yaml`) and change the value of spec.desiredState to `Established`.
3. Save the CR file.
4. Apply the CR:

   ```
   kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
   ```

5. If you created any protection schedules on the destination cluster to protect the failed over application, remove them. Any schedules that remain cause volume snapshot failures.

Reverse resync a failed over replication relationship

When you reverse resync a failed over replication relationship, the destination application becomes the source application, and the source becomes the destination. Changes made to the destination application during failover are kept.

Steps

1. Delete the AppMirrorRelationship CR on the original destination cluster. This causes the destination to become the source. If there are any protection schedules remaining on the new destination cluster, remove them.
2. Set up a replication relationship by applying the CR files you originally used to set up the relationship to the opposite clusters.
3. Ensure the AppVault CRs are ready on each cluster.
4. Set up a replication relationship on the opposite cluster, configuring values for the reverse direction.

Reverse application replication direction

When you reverse replication direction, Trident protect moves the application to the destination storage backend while continuing to replicate back to the original source storage backend. Trident protect stops the source application and replicates the data to the destination before failing over to the destination app.

In this situation, you are swapping the source and destination.

Steps

1. Create a shutdown snapshot:

Create a shutdown snapshot using a CR

a. Disable the protection policy schedules for the source application.

b. Create a ShutdownSnapshot CR file:

   i. Create the custom resource (CR) file and name it (for example, `trident-protect-shutdownsnapshot.yaml`).

   ii. Configure the following attributes:

      - **metadata.name**: (*Required*) The name of the custom resource.
      - **spec.AppVaultRef**: (*Required*) This value must match the metadata.name field of the AppVault for the source application.
      - **spec.ApplicationRef**: (*Required*) This value must match the metadata.name field of the source application CR file.

      Example YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-
86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-
04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. After you populate the `trident-protect-shutdownsnapshot.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-
namespace
```

Create a shutdown snapshot using the CLI

a. Create the shutdown snapshot, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot>
--appvault <my_vault> --app <app_to_snapshot> -n
<application_namespace>
```

2. After the snapshot completes, get the status of the snapshot:

```
kubectl get shutdownsnapshot -n my-app-namespace
<shutdown_snapshot_name> -o yaml
```

3. Find the value of **shutdownsnapshot.status.appArchivePath** using the following command, and record the last part of the file path (also called the basename; this will be everything after the last slash):

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. Perform a fail over from the destination cluster to the source cluster, with the following change:

> In step 2 of the fail over procedure, include the `spec.promotedSnapshot` field in the AppMirrorRelationship CR file, and set its value to the basename you recorded in step 3 above.

5. Perform the reverse resync steps in Reverse resync a failed over replication relationship.
6. Enable protection schedules on the new source cluster.

Result

The following actions occur because of the reverse replication:

- A snapshot is taken of the original source app's Kubernetes resources.
- The original source app's pods are gracefully stopped by deleting the app's Kubernetes resources (leaving PVCs and PVs in place).
- After the pods are shut down, snapshots of the app's volumes are taken and replicated.
- The SnapMirror relationships are broken, making the destination volumes ready for read/write.
- The app's Kubernetes resources are restored from the pre-shutdown snapshot, using the volume data replicated after the original source app was shut down.
- Replication is re-established in the reverse direction.

Fail back applications to the original source cluster

Using Trident protect, you can achieve "fail back" after a failover operation by using the following sequence of operations. In this workflow to restore the original replication direction, Trident protect replicates (resyncs) any application changes back to the original source application before reversing the replication direction.

This process starts from a relationship that has completed a failover to a destination and involves the following steps:

- Start with a failed over state.
- Reverse resync the replication relationship.

> Do not perform a normal resync operation, as this will discard data written to the destination cluster during the fail over procedure.

- Reverse the replication direction.

Steps

1. Perform the Reverse resync a failed over replication relationship steps.
2. Perform the Reverse application replication direction steps.

Delete a replication relationship

You can delete a replication relationship at any time. When you delete the application replication relationship, it results in two separate applications with no relationship between them.

Steps

1. Delete the AppMirrorRelationship CR:

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

Migrate applications using Trident protect

You can migrate your applications between clusters or storage classes by restoring your backup or snapshot data to a different cluster or storage class.

> When you migrate an application, all execution hooks configured for the application are migrated with the app. If a post-restore execution hook is present, it runs automatically as part of the restore operation.

Backup and restore operations

To perform backup and restore operations for the following scenarios, you can automate specific backup and restore tasks.

Clone to same cluster

To clone an application to the same cluster, create a snapshot or backup and restore the data to the same cluster.

Steps

1. Do one of the following:
   a. Create a snapshot.
   b. Create a backup.
2. On the same cluster, do one of the following, depending on if you created a snapshot or a backup:
   a. Restore your data from the snapshot.
   b. Restore your data from the backup.

Clone to different cluster

To clone an application to a different cluster (perform a cross-cluster clone), create a backup on the source cluster, and then restore the backup to a different cluster. Make sure that Trident protect is installed on the destination cluster.

You can replicate an application between different clusters using SnapMirror replication.

Steps

1. Create a backup.
2. Ensure that the AppVault CR for the object storage bucket that contains the backup has been configured on the destination cluster.
3. On the destination cluster, restore your data from the backup.

Migrate applications from one storage class to another storage class

You can migrate applications from one storage class to a different storage class by restoring a snapshot to the different destination storage class.

For example (excluding the secrets from the restore CR):

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    destination: "${destinationNamespace}"
    source: "${sourceNamespace}"
  storageClassMapping:
    destination: "${destinationStorageClass}"
    source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

Restore the snapshot using a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-snapshot-restore-cr.yaml`.
2. In the file you created, configure the following attributes:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.appArchivePath**: The path inside AppVault where the snapshot contents are stored. You can use the following command to find this path:

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (*Required*) The name of the AppVault where the snapshot contents are stored.
   - **spec.namespaceMapping**: The mapping of the source namespace of the restore operation to the destination namespace. Replace `my-source-namespace` and `my-destination-namespace` with information from your environment.

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
       appArchivePath: my-snapshot-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. Optionally, if you need to select only certain resources of the application to restore, add filtering that includes or excludes resources marked with particular labels:
   - **resourceFilter.resourceSelectionCriteria**: (Required for filtering) Use `include or exclude` to include or exclude a resource defined in resourceMatchers. Add the following resourceMatchers parameters to define the resources to be included or excluded:
     - **resourceFilter.resourceMatchers**: An array of resourceMatcher objects. If you define multiple elements in this array, they match as an OR operation, and the fields inside each element (group, kind, version) match as an AND operation.
       - **resourceMatchers[].group**: (*Optional*) Group of the resource to be filtered.
       - **resourceMatchers[].kind**: (*Optional*) Kind of the resource to be filtered.
       - **resourceMatchers[].version**: (*Optional*) Version of the resource to be filtered.

- **resourceMatchers[].names**: (*Optional*) Names in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].namespaces**: (*Optional*) Namespaces in the Kubernetes metadata.name field of the resource to be filtered.
- **resourceMatchers[].labelSelectors**: (*Optional*) Label selector string in the Kubernetes metadata.name field of the resource as defined in the Kubernetes documentation. For example: `"trident.netapp.io/os=linux"`.

  For example:

```yaml
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. After you populate the `trident-protect-snapshot-restore-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

Restore the snapshot using the CLI

Steps

1. Restore the snapshot to a different namespace, replacing values in brackets with information from your environment.
   - The `snapshot` argument uses a namespace and snapshot name in the format `<namespace>/<name>`.
   - The `namespace-mapping` argument uses colon-separated namespaces to map source namespaces to the correct destination namespaces in the format `source1:dest1,source2:dest2`.

     For example:

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

Manage Trident protect execution hooks

An execution hook is a custom action that you can configure to run in conjunction with a data protection operation of a managed app. For example, if you have a database app, you can use an execution hook to pause all database transactions before a snapshot, and resume transactions after the snapshot is complete. This ensures application-consistent snapshots.

Types of execution hooks

Trident protect supports the following types of execution hooks, based on when they can be run:

- Pre-snapshot
- Post-snapshot
- Pre-backup
- Post-backup
- Post-restore
- Post-failover

Order of execution

When a data protection operation is run, execution hook events take place in the following order:

1. Any applicable custom pre-operation execution hooks are run on the appropriate containers. You can create and run as many custom pre-operation hooks as you need, but the order of execution of these hooks before the operation is neither guaranteed nor configurable.
2. Filesystem freezes occur, if applicable. Learn more about configuring filesystem freezing with Trident protect.
3. The data protection operation is performed.
4. Frozen filesystems are unfrozen, if applicable.
5. Any applicable custom post-operation execution hooks are run on the appropriate containers. You can create and run as many custom post-operation hooks as you need, but the order of execution of these hooks after the operation is neither guaranteed nor configurable.

If you create multiple execution hooks of the same type (for example, pre-snapshot), the order of execution of those hooks is not guaranteed. However, the order of execution of hooks of different types is guaranteed. For example, the following is the order of execution of a configuration that has all of the different types of hooks:

1. Pre-snapshot hooks executed
2. Post-snapshot hooks executed

3. Pre-backup hooks executed
4. Post-backup hooks executed

The preceding order example only applies when you run a backup that does not use an existing snapshot.

You should always test your execution hook scripts before enabling them in a production environment. You can use the 'kubectl exec' command to conveniently test the scripts. After you enable the execution hooks in a production environment, test the resulting snapshots and backups to ensure they are consistent. You can do this by cloning the app to a temporary namespace, restoring the snapshot or backup, and then testing the app.

Important notes about custom execution hooks

Consider the following when planning execution hooks for your apps.

- An execution hook must use a script to perform actions. Many execution hooks can reference the same script.
- Trident protect requires the scripts that execution hooks use to be written in the format of executable shell scripts.
- Script size is limited to 96KB.
- Trident protect uses execution hook settings and any matching criteria to determine which hooks are applicable to a snapshot, backup, or restore operation.

Because execution hooks often reduce or completely disable the functionality of the application they are running against, you should always try to minimize the time your custom execution hooks take to run. If you start a backup or snapshot operation with associated execution hooks but then cancel it, the hooks are still allowed to run if the backup or snapshot operation has already begun. This means that the logic used in a post-backup execution hook cannot assume that the backup was completed.

Execution hook filters

When you add or edit an execution hook for an application, you can add filters to the execution hook to manage which containers the hook will match. Filters are useful for applications that use the same container image on all containers, but might use each image for a different purpose (such as Elasticsearch). Filters allow you to create scenarios where execution hooks run on some but not necessarily all identical containers. If you create multiple filters for a single execution hook, they are combined with a logical AND operator. You can have up to 10 active filters per execution hook.

Each filter you add to an execution hook uses a regular expression to match containers in your cluster. When a hook matches a container, the hook will run its associated script on that container. Regular expressions for filters use the Regular Expression 2 (RE2) syntax, which does not support creating a filter that excludes containers from the list of matches. For information on the syntax that Trident protect supports for regular expressions in execution hook filters, see Regular Expression 2 (RE2) syntax support.

If you add a namespace filter to an execution hook that runs after a restore or clone operation and the restore or clone source and destination are in different namespaces, the namespace filter is only applied to the destination namespace.

Execution hook examples

Visit the NetApp Verda GitHub project to download real execution hooks for popular apps such as Apache Cassandra and Elasticsearch. You can also see examples and get ideas for structuring your own custom execution hooks.

Create an execution hook

You can create a custom execution hook for an app using Trident protect. You need to have Owner, Admin, or Member permissions to create execution hooks.

Use a CR

Steps

1. Create the custom resource (CR) file and name it `trident-protect-hook.yaml`.
2. Configure the following attributes to match your Trident protect environment and cluster configuration:
   - **metadata.name**: (*Required*) The name of this custom resource; choose a unique and sensible name for your environment.
   - **spec.applicationRef**: (*Required*) The Kubernetes name of the application for which to run the execution hook.
   - **spec.stage**: (*Required*) A string indicating which stage during the action that the execution hook should run. Possible values:
     - Pre
     - Post
   - **spec.action**: (*Required*) A string indicating which action the execution hook will take, assuming any execution hook filters specified are matched. Possible values:
     - Snapshot
     - Backup
     - Restore
     - Failover
   - **spec.enabled**: (*Optional*) Indicates whether this execution hook is enabled or disabled. If not specified, the default value is true.
   - **spec.hookSource**: (*Required*) A string containing the base64-encoded hook script.
   - **spec.timeout**: (*Optional*) A number defining how long in minutes that the execution hook is allowed to run. The minimum value is 1 minute, and the default value is 25 minutes if not specified.
   - **spec.arguments**: (*Optional*) A YAML list of arguments that you can specify for the execution hook.
   - **spec.matchingCriteria**: (*Optional*) An optional list of criteria key value pairs, each pair making up an execution hook filter. You can add up to 10 filters per execution hook.
   - **spec.matchingCriteria.type**: (*Optional*) A string identifying the execution hook filter type. Possible values:
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value**: (*Optional*) A string or regular expression identifying the execution hook filter value.

     Example YAML:

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. After you populate the CR file with the correct values, apply the CR:

```
kubectl apply -f trident-protect-hook.yaml
```

Use the CLI

Steps

1. Create the execution hook, replacing values in brackets with information from your environment. For example:

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

Uninstall Trident protect

You might need to remove Trident protect components if you are upgrading from a trial to a full version of the product.

To remove Trident protect, perform the following steps.

Steps

1. Remove the Trident protect CR files:

   ```
   helm uninstall -n trident-protect trident-protect-crds
   ```

2. Remove Trident protect:

   ```
   helm uninstall -n trident-protect trident-protect
   ```

3. Remove the Trident protect namespace:

   ```
   kubectl delete ns trident-protect
   ```