Manage Trident Protect

Trident

NetApp
January 15, 2026

Table of Contents

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.

Monitor Trident Protect resources

You can use the kube-state-metrics, Prometheus, and Alertmanager open source tools to monitor the health of the resources protected by Trident Protect.

The kube-state-metrics service generates metrics from Kubernetes API communication. Using it with Trident Protect exposes useful information about the state of resources in your environment.

Prometheus is a toolkit that can ingest the data generated by kube-state-metrics and present it as easily readable information about these objects. Together, kube-state-metrics and Prometheus provide a way for you to monitor the health and status of the resources you are managing with Trident Protect.

Alertmanager is a service that ingests the alerts sent by tools such as Prometheus and routes them to destinations that you configure.

> The configurations and guidance included in these steps are only examples; you need to customize them to match your environment. Refer to the following official documentation for specific instructions and support:
>
> - kube-state-metrics documentation
> - Prometheus documentation
> - Alertmanager documentation

Step 1: Install the monitoring tools

To enable resource monitoring in Trident Protect, you need to install and configure kube-state-metrics, Promethus, and Alertmanager.

Install kube-state-metrics

You can install kube-state-metrics using Helm.

Steps

1. Add the kube-state-metrics Helm chart. For example:

   ```
   helm repo add prometheus-community https://prometheus-
   community.github.io/helm-charts
   helm repo update
   ```

2. Apply the Prometheus ServiceMonitor CRD to the cluster:

   ```
   kubectl apply -f https://raw.githubusercontent.com/prometheus-
   operator/prometheus-operator/main/example/prometheus-operator-
   crd/monitoring.coreos.com_servicemonitors.yaml
   ```

3. Create a configuration file for the Helm chart (for example, `metrics-config.yaml`). You can customize the following example configuration to match your environment:

metrics-config.yaml: kube-state-metrics Helm chart configuration

```yaml
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: protect.trident.netapp.io
          kind: "Backup"
          version: "v1"
        labelsFromPath:
          backup_uid: [metadata, uid]
          backup_name: [metadata, name]
          creation_time: [metadata, creationTimestamp]
        metrics:
        - name: backup_info
          help: "Exposes details about the Backup state"
          each:
            type: Info
            info:
              labelsFromPath:
                appVaultReference: ["spec", "appVaultRef"]
                appReference: ["spec", "applicationRef"]
rbac:
  extraRules:
  - apiGroups: ["protect.trident.netapp.io"]
    resources: ["backups"]
    verbs: ["list", "watch"]

# Collect metrics from all namespaces
namespaces: ""

# Ensure that the metrics are collected by Prometheus
prometheus:
  monitor:
    enabled: true
```

4. Install kube-state-metrics by deploying the Helm chart. For example:

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. Configure kube-state-metrics to generate metrics for the custom resources used by Trident Protect by following the instructions in the kube-state-metrics custom resource documentation.

Install Prometheus

You can install Prometheus by following the instructions in the Prometheus documentation.

Install Alertmanager

You can install Alertmanager by following the instructions in the Alertmanager documentation.

Step 2: Configure the monitoring tools to work together

After you install the monitoring tools, you need to configure them to work together.

Steps

1. Integrate kube-state-metrics with Prometheus. Edit the Prometheus configuration file (`prometheus.yaml`) and add the kube-state-metrics service information. For example:

   prometheus.yaml: kube-state-metrics service integration with Prometheus

   ```
   ---
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: prometheus-config
     namespace: trident-protect
   data:
     prometheus.yaml: |
       global:
         scrape_interval: 15s
       scrape_configs:
         - job_name: 'kube-state-metrics'
           static_configs:
             - targets: ['kube-state-metrics.trident-protect.svc:8080']
   ```

2. Configure Prometheus to route alerts to Alertmanager. Edit the Prometheus configuration file (`prometheus.yaml`) and add the following section:

prometheus.yaml: Send alerts to Alertmanager

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

Result

Prometheus can now gather metrics from kube-state-metrics, and can send alerts to Alertmanager. You are now ready to configure what conditions trigger an alert and where the alerts should be sent.

Step 3: Configure alerts and alert destinations

After you configure the tools to work together, you need to configure what type of information triggers alerts, and where the alerts should be sent.

Alert example: backup failure

The following example defines a critical alert that is triggered when the status of the backup custom resource is set to `Error` for 5 seconds or longer. You can customize this example to match your environment, and include this YAML snippet in your `prometheus.yaml` configuration file:

rules.yaml: Define a Prometheus alert for failed backups

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

Configure Alertmanager to send alerts to other channels

You can configure Alertmanager to send notifications to other channels, such as e-mail, PagerDuty, Microsoft Teams, or other notification services by specifying the respective configuration in the `alertmanager.yaml` file.

The following example configures Alertmanager to send notifications to a Slack channel. To customize this example to your environment, replace the value of the `api_url` key with the Slack webhook URL used in your environment:

alertmanager.yaml: Send alerts to a Slack channel

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

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 `trident-protect-support-bundle.yaml` file with the correct values, apply the CR:

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

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> -n trident-protect
```

Monitor and retrieve the support bundle

After creating a support bundle using either method, you can monitor its generation progress and retrieve it to your local system.

Steps

1. Wait for the `status.generationState` to reach `Completed` state. You can monitor the generation progress with the following command:

   ```
   kubectl get autosupportbundle trident-protect-support-bundle -n trident-
   protect
   ```

2. Retrieve the support bundle to your local system. Get the copy command from the completed AutoSupport bundle:

   ```
   kubectl describe autosupportbundle trident-protect-support-bundle -n
   trident-protect
   ```

   Find the `kubectl cp` command from the output and run it, replacing the destination argument with your preferred local directory.

Upgrade Trident Protect

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

> ⓘ When you upgrade from version 24.10, snapshots running during the upgrade might fail. This failure does not prevent future snapshots, whether manual or scheduled, from being created. If a snapshot fails during the upgrade, you can manually create a new snapshot to ensure your application is protected.
>
> To avoid potential failures, you can disable all snapshot schedules before the upgrade and re-enable them afterward. However, this results in missing any scheduled snapshots during the upgrade period.

To upgrade Trident Protect, perform the following steps.

Steps

1. Update the Trident Helm repository:

```
helm repo update
```

2. Upgrade the Trident Protect CRDs:

   > This step is required if you are upgrading from a version earlier than 25.06, as the CRDs are now included in the Trident Protect Helm chart.

   a. Run this command to shift management of CRDs from `trident-protect-crds` to `trident-protect`:

   ```
   kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
   xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
   {"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
   ```

   b. Run this command to delete the Helm secret for the `trident-protect-crds` chart:

   > Do not uninstall the `trident-protect-crds` chart using Helm, as this could remove your CRDs and any related data.

   ```
   kubectl delete secret -n trident-protect -l name=trident-protect-
   crds,owner=helm
   ```

3. Upgrade Trident Protect:

   ```
   helm upgrade trident-protect netapp-trident-protect/trident-protect
   --version 100.2506.0 --namespace trident-protect
   ```

Copyright information

Trademark information