Trident Protectの管理

Trident

NetApp
July 01, 2026

目次

Trident Protectの管理

Trident Protectの認証とアクセス制御を管理する

Trident Protect は Kubernetes モデルのロールベースアクセス制御（RBAC）を使用します。デフォルトでは、Trident Protectは、単一のシステムネームスペースとそれに関連付けられたデフォルトのサービスアカウントを提供します。多数のユーザーを抱える組織や特定のセキュリティニーズを持つ組織の場合は、Trident Protect の RBAC 機能を使用して、リソースとネームスペースへのアクセスをより細かく制御できます。

クラスタ管理者は常にdefault `trident-protect`ネームスペース内のリソースにアクセスでき、他のすべてのネームスペース内のリソースにもアクセスできます。リソースとアプリケーションへのアクセスを制御するには、追加のネームスペースを作成し、それらのネームスペースにリソースとアプリケーションを追加する必要があります。

デフォルトの `trident-protect`ネームスペースでは、どのユーザーもアプリケーションデータ管理CRを作成できないことに注意してください。アプリケーションネームスペースにアプリケーションデータ管理CRを作成する必要があります（ベストプラクティスとして、関連付けられているアプリケーションと同じネームスペースにアプリケーションデータ管理CRを作成します）。

> 特権を持つTrident Protectカスタムリソースオブジェクトへのアクセスは、管理者のみが持つ必要があります。これには次のものが含まれます：
>
> - **AppVault**：バケット認証情報データが必要です
> - **AutoSupportBundle**：メトリクス、ログ、およびその他の機密性の高いTrident Protectデータを収集します
> - **AutoSupportBundleSchedule**：ログ収集スケジュールを管理します
>
> ベストプラクティスとして、RBAC を使用して、特権オブジェクトへのアクセスを管理者に制限します。

RBACがリソースと名前空間へのアクセスをどのように制御するかの詳細については、 "Kubernetes RBAC ドキュメント"を参照してください。

サービスアカウントの詳細については、 "Kubernetes サービスアカウントのドキュメント"を参照してください。

例：2つのユーザーグループのアクセスを管理する

たとえば、組織にはクラスタ管理者、エンジニアリングユーザーのグループ、マーケティングユーザーのグループがあります。クラスタ管理者は、エンジニアリンググループとマーケティンググループがそれぞれの名前空間に割り当てられたリソースのみにアクセスできる環境を作成するために、次のタスクを実行します（：）

ステップ1：各グループのリソースを格納する名前空間を作成する

名前空間を作成すると、リソースを論理的に分離し、それらのリソースにアクセスできるユーザーをより適切に制御できるようになります。

手順

1. エンジニアリンググループの名前空間を作成します：

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

2. マーケティンググループのネームスペースを作成します：

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

ステップ**2**：各名前空間内のリソースとやり取りするための新しいサービスアカウントを作成する

作成する新しいネームスペースにはそれぞれデフォルトのサービスアカウントが付属しますが、将来必要に応じてグループ間で権限をさらに分割できるように、ユーザーグループごとにサービスアカウントを作成する必要があります。

手順

1. エンジニアリンググループのサービスアカウントを作成します：

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

2. マーケティンググループのサービスアカウントを作成します：

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

ステップ**3**：新しいサービスアカウントごとにシークレットを作成する

サービスアカウントシークレットは、サービスアカウントでの認証に使用され、侵害された場合には簡単に削除して再作成できます。

手順

1. エンジニアリングサービスアカウントのシークレットを作成します：

```
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. marketingサービスアカウントのシークレットを作成します：

```
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
```

ステップ**4**：**RoleBinding**オブジェクトを作成して、**ClusterRole**オブジェクトを各新しいサービスアカウントにバインドします

Trident Protectをインストールすると、デフォルトのClusterRoleオブジェクトが作成されます。このClusterRoleをサービスアカウントにバインドするには、RoleBindingオブジェクトを作成して適用します。

手順

1. ClusterRoleをエンジニアリングサービスアカウントにバインドします：

```
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. ClusterRoleをマーケティングサービスアカウントにバインドします：

```
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
```

ステップ**5**：権限をテストする

権限が正しいことをテストします。

手順

1. エンジニアリングユーザーがエンジニアリングリソースにアクセスできることを確認します：

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

2. エンジニアリングユーザーがマーケティングリソースにアクセスできないことを確認します：

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

ステップ**6**：**AppVault**オブジェクトへのアクセスを許可する

バックアップやスナップショットなどのデータ管理タスクを実行するには、クラスタ管理者が個々のユーザーにAppVaultオブジェクトへのアクセスを許可する必要があります。

手順

1. AppVaultとシークレットの組み合わせYAMLファイルを作成して適用し、ユーザーにAppVaultへのアクセス権を付与します。たとえば、次のCRは、ユーザー `eng-user`にAppVaultへのアクセス権を付与します：

```
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. ロール CR を作成して適用し、クラスター管理者が名前空間内の特定のリソースへのアクセスを許可できるようにします。例：

```
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. RoleBinding CR を作成して適用し、ユーザー eng-user に権限をバインドします。例：

```
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. 権限が正しいことを確認してください。

   a. すべての名前空間のAppVaultオブジェクト情報の取得を試みます：

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

   次のような出力が表示されます。

```
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. ユーザーがアクセス権限を持つようになったAppVault情報を取得できるかどうかをテストします：

```
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
```

次のような出力が表示されます。

```
yes
```

結果

AppVault権限を付与したユーザーは、承認されたAppVaultオブジェクトをアプリケーションデータ管理操作に使用できる必要があり、割り当てられた名前空間外のリソースにアクセスしたり、アクセス権のない新しいリソースを作成したりすることはできません。

Trident Protectリソースを監視

kube-state-metrics、Prometheus、およびAlertmanagerオープンソースツールを使用して、Trident Protectで保護されているリソースの健全性を監視できます。

kube-state-metricsサービスは、Kubernetes API通信からメトリックを生成します。Trident Protectと併用することで、環境内のリソースの状態に関する有用な情報が公開されます。

Prometheus は、kube-state-metrics によって生成されたデータを取り込み、これらのオブジェクトに関する読みやすい情報として提示できるツールキットです。kube-state-metrics と Prometheus を組み合わせることで、Trident Protect で管理しているリソースの健全性とステータスを監視できます。

Alertmanager は、Prometheus などのツールによって送信されたアラートを取り込み、設定した宛先にルーティングするサービスです。

> これらの手順に含まれる構成とガイダンスは単なる例であり、環境に合わせてカスタマイズする必要があります。具体的な手順とサポートについては、次の公式ドキュメントを参照してください：
>
> - "kube-state-metricsのドキュメント"
> - "Prometheusのドキュメント"
> - "Alertmanager ドキュメント"

ステップ**1**：監視ツールをインストールする

Trident Protect でリソース監視を有効にするには、kube-state-metrics、Prometheus、および Alertmanager をインストールして構成する必要があります。

kube-state-metricsをインストールする

Helm を使用して kube-state-metrics をインストールできます。

手順

1. kube-state-metrics Helm チャートを追加します。例：

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

2. Prometheus ServiceMonitor CRD をクラスタに適用します：

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

3. Helmチャートの設定ファイルを作成します（例： `metrics-config.yaml`）。次の例の構成を、環境に合わせてカスタマイズできます：

metrics-config.yaml：kube-state-metrics Helm チャート設定

```
---
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. Helm チャートをデプロイして kube-state-metrics をインストールします。例：

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

5. "kube-state-metricsカスタムリソースのドキュメント"の手順に従って、Trident Protectで使用されるカスタムリソースのメトリックを生成するようにkube-state-metricsを設定します。

Prometheusのインストール

Prometheusは、 "Prometheusのドキュメント"の手順に従ってインストールできます。

Alertmanagerをインストールする

Alertmanagerは、 "Alertmanager ドキュメント"の手順に従ってインストールできます。

ステップ2：監視ツールを連携するように設定する

監視ツールをインストールした後、それらが連携するように構成する必要があります。

手順

1. kube-state-metricsをPrometheusと統合します。Prometheus設定ファイル(`prometheus.yaml`）を編集し、kube-state-metricsサービス情報を追加します。例：

prometheus.yaml：**kube-state-metrics** サービスと **Prometheus** の統合

```
---
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. アラートを Alertmanager にルーティングするように Prometheus を構成します。Prometheus 設定ファイル(`prometheus.yaml`を編集し、次のセクションを追加します：

prometheus.yaml：**Alertmanager**にアラートを送信する

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

結果

Prometheus は kube-state-metrics からメトリクスを収集し、Alertmanager にアラートを送信できるようになりました。これで、アラートをトリガーする条件とアラートの送信先を設定する準備が整いました。

ステップ3：アラートとアラートの送信先を設定する

ツールが連携するように構成したら、アラートをトリガーする情報の種類とアラートの送信先を構成する必要があります。

アラートの例：バックアップ失敗

次の例では、バックアップカスタムリソースのステータスが `Error`に設定されてから5秒以上経過したときにトリガーされる重要なアラートを定義します。この例を環境に合わせてカスタマイズし、このYAMLスニペットを `prometheus.yaml`設定ファイルに含めることができます：

rules.yaml：失敗したバックアップに対する **Prometheus** アラートを定義する

```
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."
```

Alertmanager を設定して他のチャネルにアラートを送信する

Alertmanagerを設定して、他のチャネルに通知を送信することもできます。Eメール、PagerDuty、Microsoft Teams、またはその他の通知サービスを設定するには、 `alertmanager.yaml`ファイルでそれぞれの設定を指定します。

次の例では、Slack チャネルに通知を送信するように Alertmanager を構成します。この例を自分の環境に合わせてカスタマイズするには、 `api_url`キーの値をお使いの環境で使用されている Slack Webhook URL に置き換えます：

alertmanager.yaml：Slackチャンネルにアラートを送信する

```
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
```

Trident Protect サポートバンドルを生成する

Trident Protectは、管理者がログ、メトリック、管理下のクラスタやアプリに関するトポロジ情報など、NetApp Supportに役立つ情報を含むバンドルを生成できるようにします。インターネットに接続されている場合、カスタムリソース（CR）ファイルを使用して、サポートバンドルをNetApp Support Site（NSS）にアップロードできます。

CR を使用してサポートバンドルを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、名前を付けます（例： `trident-protect-support-bundle.yaml`）。
2. 次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.triggerType**：（必須）サポート バンドルをすぐに生成するか、スケジュールするかを決定します。スケジュールされたバンドル生成は、UTC の午前 12 時に行われます。有効な値は次のとおりです。
     - スケジュール済み
     - 手動
   - **spec.uploadEnabled**：（オプション）サポートバンドルの生成後にNetApp Support Siteにアップロードするかどうかを制御します。指定されていない場合、デフォルトは `false`です。有効な値は次のとおりです。
     - true
     - false （デフォルト）
   - **spec.dataWindowStart**：（オプション）サポート バンドルに含まれるデータのウィンドウの開始日時を指定する RFC 3339 形式の日付文字列。指定しない場合は、デフォルトで 24 時間前になります。指定できる最も早いウィンドウ日付は 7 日前です。

     YAMLの例：

```
---
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. `trident-protect-support-bundle.yaml`ファイルに正しい値を入力したら、CRを適用します：

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

CLI を使用してサポートバンドルを作成する

手順

1. 括弧内の値を環境の情報に置き換えて、サポート バンドルを作成します。 `trigger-type`は、バンドルがすぐに作成されるか、作成時間がスケジュールによって決定されるかを決定します。 `Manual`または `Scheduled`を指定できます。デフォルト設定は `Manual`です。

   次に例を示します。

   ```
   tridentctl-protect create autosupportbundle <my-bundle-name>
   --trigger-type <trigger-type> -n trident-protect
   ```

サポートバンドルを監視して取得する

いずれかの方法を使用してサポート バンドルを作成した後、その生成の進行状況を監視し、ローカル システムに取得することができます。

手順

1. `status.generationState`が `Completed`状態になるまで待ちます。次のコマンドで生成の進行状況を監視できます：

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

2. サポート バンドルをローカル システムに取得します。完了したAutoSupportバンドルからコピー コマンドを取得します：

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

   出力から `kubectl cp`コマンドを見つけて実行し、宛先引数を希望のローカル ディレクトリに置き換えます。

Trident Protectのアップグレード

新しい機能やバグ修正を活用するには、Trident Protectを最新バージョンにアップグレードできます。

- バージョン24.10からアップグレードすると、アップグレード中に実行されるスナップショットが失敗する可能性があります。この障害により、手動またはスケジュールされた将来のスナップショットの作成が妨げられることはありません。アップグレード中にスナップショットが失敗した場合は、アプリケーションが保護されるように手動で新しいスナップショットを作成できます。

  潜在的な障害を回避するために、アップグレード前にすべてのスナップショット スケジュールを無効にして、アップグレード後に再度有効にすることができます。ただし、これにより、アップグレード期間中にスケジュールされたスナップショットが失われることになります。

- プライベート レジストリのインストールでは、ターゲット バージョンに必要な Helm チャートとイメージがプライベート レジストリで使用できることを確認し、カスタム Helm 値が新しいチャート バージョンと互換性があることを確認します。詳細については、"プライベートレジストリからTrident Protectをインストールする"を参照してください。

ステップ1：バージョンを選択する

Trident Protectのバージョンは日付ベースの `YY.MM`命名規則に従い、「YY」は年の下2桁、「MM」は月を表します。ドットリリースは `YY.MM.X`命名規則に従い、「X」はパッチレベルを表します。アップグレード元のバージョンに基づいて、アップグレード先のバージョンを選択してください。

- インストールされているバージョンから4リリース以内の任意のターゲットリリースへの直接アップグレードを実行できます。たとえば、24.10（または任意の24.10ドットリリース）から25.10に直接アップグレードできます。
- 4つのリリース期間外のリリースからアップグレードする場合は、複数ステップのアップグレードを実行してください。"以前のバージョン"からアップグレードする場合は、そのバージョンのアップグレード手順を使用して、4リリース期間に収まる最新のリリースにアップグレードします。例えば、24.10 を実行していて、26.02 にアップグレードしたい場合（：）
  a. 最初に 24.10 から 25.02 にアップグレードします。
  b. 次に、25.02から26.02にアップグレードします。

ステップ2：Trident Protectをアップグレードする

Trident Protectをアップグレードするには、次の手順を実行します。

手順

1. Trident Helm リポジトリを更新します：

```
helm repo update
```

2. Trident Protect CRD をアップグレードする：

   25.06より前のバージョンからアップグレードする場合は、CRDがTrident Protect Helmチャートに含まれているため、この手順が必要です。

   a. このコマンドを実行して、CRDの管理を `trident-protect-crds`から `trident-protect`に移行します：

```
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. このコマンドを実行して `trident-protect-crds`チャートのHelmシークレットを削除します：

> Helm を使用して `trident-protect-crds`チャートをアンインストールしないでください。CRD と関連データが削除される可能性があります。

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

3. Trident Protectのアップグレード：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2602.0 --namespace trident-protect
```

> アップグレード中にログレベルを設定するには、アップグレード コマンドに `--set logLevel=debug`を追加します。デフォルトのログレベルは `warn`です。デバッグログは、ログ レベルの変更や問題の再現を必要とせずにNetAppサポートが問題を診断するのに役立つため、トラブルシューティングに推奨されます。