



Trident **Protect**でアプリケーションを保護する Trident

NetApp
February 05, 2026

目次

Trident Protectでアプリケーションを保護する	1
Tridentプロテクトについて学ぶ	1
次の手順	1
Trident Protectをインストールする	1
Tridentプロテクトの要件	1
Trident Protectのインストールと設定	4
Trident Protect CLIプラグインをインストールする	10
Trident Protectの管理	14
Trident Protectの認証とアクセス制御を管理する	14
Trident Protect サポートバンドルを生成する	21
Tridentプロテクトのアップグレード	23
アプリケーションの管理と保護	23
Trident Protect AppVault オブジェクトを使用してバケットを管理する	23
Trident Protectで管理するアプリケーションを定義する	32
Trident Protectを使用してアプリケーションを保護する	34
Trident Protectを使用してアプリケーションを復元する	42
NetApp SnapMirrorとTrident Protectを使用してアプリケーションを複製する	58
Trident Protectを使用してアプリケーションを移行する	72
Trident Protect実行フックを管理する	76
Trident Protectをアンインストールする	81

Trident Protectでアプリケーションを保護する

Tridentプロテクトについて学ぶ

NetApp Trident Protect は、NetApp ONTAPストレージ システムとNetApp Trident CSI ストレージ プロビジョナーによってサポートされるステートフル Kubernetes アプリケーションの機能と可用性を強化する高度なアプリケーション データ管理機能を提供します。Trident Protect は、パブリック クラウドとオンプレミス環境全体でのコンテナ化されたワークロードの管理、保護、移動を簡素化します。また、API と CLI を通じて自動化機能も提供します。

カスタム リソース (CR) を作成するか、Trident Protect CLI を使用することで、Trident Protect を使用してアプリケーションを保護できます。

次の手順

Trident Protect をインストールする前に、その要件について知ることができます。

- ["Tridentプロテクトの要件"](#)

Trident Protectをインストールする

Tridentプロテクトの要件

まず、運用環境、アプリケーション クラスター、アプリケーション、ライセンスの準備状況を確認します。Trident Protect を展開および運用するには、環境がこれらの要件を満たしていることを確認してください。

Trident Protect Kubernetes クラスターの互換性

Trident Protect は、以下を含む幅広いフルマネージドおよびセルフマネージド Kubernetes 製品と互換性があります。

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes Service (AKS)
- Red Hat OpenShift のサービスです
- SUSE Rancher
- VMware Tanzuポートフォリオ
- アップストリームKubernetes



Trident Protect をインストールするクラスターに、実行中のスナップショット コントローラーと関連する CRD が設定されていることを確認します。スナップショットコントローラをインストールするには、["以下の手順を参照して"](#)。

Trident Protect ストレージバックエンドの互換性

Trident Protect は次のストレージ バックエンドをサポートしています。

- NetApp ONTAP 対応の Amazon FSX
- Cloud Volumes ONTAP
- ONTAPストレエシアレイ
- Google Cloud NetAppボリューム
- Azure NetApp Files

ストレージバックエンドが次の要件を満たしていることを確認します。

- クラスタに接続されているNetAppストレージがAstra Trident 24.02以降を使用していることを確認します (Trident 24.10を推奨)。
 - Astra Tridentが24.06.1より前のバージョンで、NetApp SnapMirrorディザスタリカバリ機能を使用する場合は、Astra Control Provisionerを手動で有効にする必要があります。
- 最新のAstra Control Provisionerがインストールされていることを確認します (Astra Trident 24.06.1以降ではデフォルトでインストールおよび有効化されています)。
- NetApp ONTAPストレージバックエンドがあることを確認します。
- バックアップを格納するオブジェクトストレージバケットを設定しておきます。
- アプリケーションまたはアプリケーション データ管理操作に使用する予定のアプリケーション名前空間を作成します。Trident Protect はこれらの名前空間を作成しません。カスタム リソースに存在しない名前空間を指定すると、操作は失敗します。

NASエコノミーボリュームの要件

Trident Protect は、NAS エコノミー ボリュームへのバックアップおよび復元操作をサポートします。スナップショット、クローン、および NAS エコノミー ボリュームへのSnapMirrorレプリケーションは現在サポートされていません。Trident Protect で使用する予定の各 nas-economy ボリュームに対してスナップショット ディレクトリを有効にする必要があります。



一部のアプリケーションは、Snapshotディレクトリを使用するボリュームと互換性がありません。これらのアプリケーションでは、ONTAPストレージシステムで次のコマンドを実行して、snapshotディレクトリを非表示にする必要があります。

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

snapshotディレクトリを有効にするには、NASエコノミーボリュームごとに次のコマンドを実行し、を変更するボリュームのUUIDに置き換え `<volume-UUID>` ます。

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



新しいボリュームに対してSnapshotディレクトリをデフォルトで有効にするには、Tridentバックエンド構成オプションを `true` 設定し `snapshotDir` ます。既存のボリュームには影響しません。

KubeVirt VMによるデータ保護

Trident Protect 24.10 と 24.10.1 以降では、KubeVirt VM 上で実行されているアプリケーションを保護する場合の動作が異なります。どちらのバージョンでも、データ保護操作中にファイルシステムのフリーズとフリーズ解除を有効または無効にすることができます。



すべてのTrident Protect バージョンにおいて、OpenShift 環境で自動フリーズ機能を有効または無効にするには、アプリケーション名前空間に特権を付与する必要がある場合があります。例えば：

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

Tridentプロテクト 24.10

Trident Protect 24.10 は、データ保護操作中に KubeVirt VM ファイルシステムの一貫した状態を自動的に保証しません。Trident Protect 24.10 を使用して KubeVirt VM データを保護する場合は、データ保護操作の前に、ファイルシステムのフリーズ/アンフリーズ機能を手動で有効にする必要があります。これにより、ファイルシステムが一貫した状態になることが保証されます。

Trident Protect 24.10を設定して、データ保護操作中にVMファイルシステムの凍結と解凍を管理することができます。["仮想化の設定"](#)として次のコマンドを使用します。

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

Trident Protect 24.10.1以降

Trident Protect 24.10.1以降、Trident Protectはデータ保護操作中にKubeVirtファイルシステムを自動的にフリーズおよびアンフリーズします。オプションで、次のコマンドを使用してこの自動動作を無効にすることができます：

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

SnapMirrorレプリケーションの要件

NetApp SnapMirror は、次のONTAPソリューションのTrident Protect で使用できます。

- NetApp ASA
- NetApp AFF
- NetApp FAS

- NetApp ONTAP Select の略
- NetApp Cloud Volumes ONTAP の略
- NetApp ONTAP 対応の Amazon FSX

SnapMirrorレプリケーション用のONTAPクラスタの要件

SnapMirrorレプリケーションを使用する場合は、ONTAPクラスタが次の要件を満たしていることを確認します。

- * Astra Control Provisioner または Trident *: ONTAP をバックエンドとして使用するソース Kubernetes クラスタと宛先 Kubernetes クラスタの両方に Astra Control Provisioner または Trident が存在する必要があります。Trident Protect は、次のドライバーによってサポートされるストレージクラスを使用して、NetApp SnapMirrorテクノロジーによるレプリケーションをサポートします。
 - ontap-nas
 - ontap-san
- ライセンス：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデスティネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください ["ONTAP の SnapMirror ライセンスの概要"](#)。

SnapMirrorレプリケーションのピアリングに関する考慮事項

ストレージバックエンドピアリングを使用する場合は、環境が次の要件を満たしていることを確認してください。

- * クラスタと SVM *: ONTAPストレージバックエンドにピア関係が設定されている必要があります。詳細については、を参照してください ["クラスタと SVM のピアリングの概要"](#)。



2つのONTAPクラスタ間のレプリケーション関係で使用されるSVM名が一意であることを確認してください。

- * Astra Control Provisioner または Trident と SVM *: ピア関係にあるリモートSVMは、デスティネーションクラスタのAstra Control Provisioner または Trident で使用できる必要があります。
- 管理対象バックエンド: レプリケーション関係を作成するには、Trident Protect で ONTAP ストレージ バックエンドを追加および管理する必要があります。
- **NVMe over TCP**: Trident Protect は、NVMe over TCP プロトコルを使用しているストレージ バックエンドの NetApp SnapMirror レプリケーションをサポートしていません。

SnapMirrorレプリケーション用のTrident / ONTAPの設定

Trident Protect では、ソース クラスタと宛先クラスタの両方のレプリケーションをサポートするストレージ バックエンドを少なくとも 1 つ構成する必要があります。ソース クラスタと宛先クラスタが同じ場合、復元力を最大限に高めるには、宛先アプリケーションでソース アプリケーションとは異なるストレージ バックエンドを使用する必要があります。

Trident Protectのインストールと設定

環境が Trident Protect の要件を満たしている場合は、次の手順に従ってクラスタに Trident Protect をインストールできます。Trident Protect は NetApp から入手するか、

独自のプライベートレジストリからインストールすることができます。クラスターがインターネットにアクセスできない場合は、プライベートレジストリからインストールすると便利です。



デフォルトでは、Trident Protect は、クラスターと管理対象アプリケーションに関するログ、メトリック、トポロジ情報など、開く可能性のあるNetAppサポートケースに役立つサポート情報を収集します。Trident Protect は、これらのサポートバンドルを毎日スケジュールに従ってNetAppに送信します。Trident Protect をインストールするときに、オプションでこのサポートバンドルの収集を無効にすることができます。手動で"[サポートバンドルの生成](#)"いつでも。

Trident Protectをインストールする

NetAppからTrident Protectをインストールする

手順

1. Trident Helmリポジトリを追加します。

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

2. Trident Protect CRD をインストールします。

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

3. 次のいずれかのコマンドを使用して、Helm を使用してTrident Protect をインストールします。交換する`<name_of_cluster>`クラスター名。このクラスター名はクラスターに割り当てられ、クラスターのバックアップとスナップショットを識別するために使用されます。

- Trident Protect を通常どおりインストールします。

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

- Trident Protect をインストールし、毎日スケジュールされているTrident Protect AutoSupportサポート バンドルのアップロードを無効にします。

```
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
```

プライベートレジストリからTrident Protectをインストールする

Kubernetes クラスターがインターネットにアクセスできない場合は、プライベート イメージ レジストリからTrident Protect をインストールできます。これらの例では、括弧内の値を環境の情報に置き換えます。

手順

1. 次のイメージをローカルマシンにプルし、タグを更新して、プライベートレジストリにプッシュします。

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exehook: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
```

例：

```
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. Trident Protect システム名前空間を作成します。

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

3. レジストリにログインします。

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

4. プライベートレジストリ認証に使用するプルシークレットを作成します。

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

5. Trident Helmリポジトリを追加します。

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

6. という名前のファイルを作成します `protectValues.yaml`。次のTrident Protect 設定が含まれていることを確認します。

```
---
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. Trident Protect CRD をインストールします。

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

8. 次のいずれかのコマンドを使用して、Helm を使用してTrident Protect をインストールします。交換する `<name_of_cluster>` クラスター名。このクラスター名はクラスターに割り当てられ、クラスターのバックアップとスナップショットを識別するために使用されます。

- Trident Protect を通常どおりインストールします。

```
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
```

- Trident Protect をインストールし、毎日スケジュールされているTrident Protect AutoSupportサポート バンドルのアップロードを無効にします。

```
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
```

Trident Protectコンテナのリソース制限を指定する

Trident Protect をインストールした後、構成ファイルを使用してTrident Protect コンテナのリソース制限を指定できます。リソース制限を設定すると、Trident Protect 操作によって消費されるクラスターのリソースの量を制御できます。

手順

1. という名前のファイルを作成します resourceLimits.yaml。
2. 環境のニーズに応じて、Trident Protect コンテナのリソース制限オプションをファイルに入力します。

次の構成ファイルの例は、使用可能な設定を示しています。このファイルには、各リソース制限のデフォルトの変数が含まれています。

```
---
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. ファイルから値を適用し `resourceLimits.yaml` ます。

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

Trident Protect CLIプラグインをインストールする

Trident Protectコマンドラインプラグインを使用できます。これはTridentの拡張機能です。 `tridentctl Trident Protect` カスタム リソース (CR) を作成し、操作するためのユーティリティです。

Trident Protect CLIプラグインをインストールする

コマンドラインユーティリティを使用する前に、クラスタへのアクセスに使用するマシンにインストールする必要があります。マシンがx64またはARM CPUを使用しているかどうかに応じて、次の手順を実行します。

Linux AMD64 CPU用プラグインのダウンロード

手順

1. Trident Protect CLI プラグインをダウンロードします。

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

Linux ARM64 CPU用プラグインのダウンロード

手順

1. Trident Protect CLI プラグインをダウンロードします。

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

Mac AMD64 CPU用プラグインのダウンロード

手順

1. Trident Protect CLI プラグインをダウンロードします。

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

Mac ARM64 CPU用プラグインのダウンロード

手順

1. Trident Protect CLI プラグインをダウンロードします。

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

1. プラグインバイナリの実行権限を有効にします。

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

2. プラグインバイナリをPATH変数で定義されている場所にコピーします。たとえば、`/usr/bin`または`/usr/local/bin（昇格されたPrivilegesが必要な場合があります）。`

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

- 必要に応じて、プラグインバイナリをホームディレクトリ内の場所にコピーできます。この場合、locationがPATH変数の一部であることを確認することをお勧めします。

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



プラグインをPATH変数の場所にコピーすると、任意の場所からまたはを `tridentctl protect` 入力してプラグインを使用でき `tridentctl-protect` ます。

Trident CLIプラグインのヘルプを表示

組み込みプラグインヘルプ機能を使用して、プラグインの機能に関する詳細なヘルプを表示できます。

手順

- ヘルプ機能を使用して、使用方法に関するガイダンスを表示します。

```
tridentctl-protect help
```

コマンドの自動補完を有効にする

Trident Protect CLI プラグインをインストールした後、特定のコマンドの自動補完を有効にすることができます。

Bashシェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします。

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

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

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

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.bash/completions` ます。

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

4. ホームディレクトリ内のファイルに次の行を追加し `~/.bashrc` ます。

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

Zシェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします。

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

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

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

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.zsh/completions` ます。

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

4. ホームディレクトリ内のファイルに次の行を追加し `~/.zprofile` ます。

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

結果

次のシェルログイン時に、tridentctl-protectプラグインで自動補完コマンドを使用できます。

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、ロールベースのアクセス制御 (RBAC) の Kubernetes モデルを使用します。デフォルトでは、Trident Protect は単一のシステム名前空間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。組織内に多数のユーザーや特定のセキュリティ ニーズがある場合は、Trident Protect の RBAC 機能を使用して、リソースや名前空間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、常にデフォルトのネームスペース内のリソースにアクセスできます trident-protect。また、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、追加の名前空間を作成し、それらの名前空間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。

デフォルトの名前空間にアプリケーションデータ管理CRSを作成することはできないことに注意して `trident-protect` ください。アプリケーションデータ管理CRSは、アプリケーションネームスペース内に作成する必要があります (ベストプラクティスとして、アプリケーションデータ管理CRSは、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作成します) 。

特権のあるTrident Protect カスタム リソース オブジェクトには、管理者のみがアクセスできる必要があります。これには次のものが含まれます。

- *** AppVault *** : バケット資格情報データが必要です。
- *** AutoSupportBundle:** メトリック、ログ、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。



RBACを使用して、権限付きオブジェクトへのアクセスを管理者に制限することを推奨します。

RBACでリソースおよびネームスペースへのアクセスを制御する方法の詳細については、[を参照して "Kubernetes RBACのドキュメント"](#) ください。

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

例：2つのユーザグループのアクセスを管理する

たとえば、ある組織に、クラスタ管理者、エンジニアリングユーザのグループ、およびマーケティングユーザのグループがあるとします。クラスタ管理者は次のタスクを実行して、engineeringグループとmarketingグル

ープがそれぞれのネームスペース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します。

手順1：各グループのリソースを含むネームスペースを作成する

ネームスペースを作成すると、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をより細かく制御できます。

手順

1. engineeringグループの名前空間を作成します。

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

2. marketingグループの名前空間を作成します。

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

ステップ2：各ネームスペース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

作成する新しい名前空間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ていますが、将来必要に応じてPrivilegesをグループ間でさらに分割できるように、ユーザーの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. engineering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、侵害された場合は簡単に削除および再作成できます。

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. マーケティ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: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

手順4：RoleBindingオブジェクトを作成して、ClusterRoleオブジェクトを新しい各サービスアカウントにバインドする

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。RoleBinding オブジェクトを作成して適用することで、この ClusterRole をサービス アカウントにバインドできます。

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
```

手順5：権限のテスト

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
```

手順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へのユーザーアクセスを許可する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ます。たとえば、次のCRは、AppVaultへのアクセスをユーザーに許可し`eng-user`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. 役割CRを作成して適用し、クラスタ管理者がネームスペース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、権限をeng-userというユーザにバインドします。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ます。

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. ユーザがAppVault情報を取得できるかどうかをテストして、アクセス許可を得ているかどうかを確認します。

```
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、アプリケーションデータ管理操作に承認されたAppVaultオブジェクトを使用できる必要があります。また、割り当てられた名前空間以外のリソースにアクセスしたり、アクセスでき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 サポートバンドルを生成する

Trident Protect を使用すると、管理者は、管理対象のクラスタとアプリケーションに関するログ、メトリック、トポロジ情報など、NetAppサポートに役立つ情報を含むバンドルを生成できます。インターネットに接続している場合は、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サポート サイト (NSS) にアップロードできます。

CRを使用したサポートバンドルの作成

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-support-bundle.yaml`)。
2. 次の属性を設定します。
 - `* metadata.name*`: (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.triggerType *`: (*required*) 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指定します。スケジュールされたバンドル生成は12AM UTCに行われます。有効な値:
 - スケジュール済み
 - 手動
 - `* spec.uploadEnabled *`: (*_Optional_*) サポートバンドルの生成後にNetAppサポートサイトにアップロードするかどうかを制御します。指定しない場合、デフォルトはになります `false`。有効な値:
 - 正しい
 - `false` (デフォルト)
 - `spec.dataWindowStart`: (*Optional*) サポートバンドルに含まれるデータのウィンドウを開始する日時を指定する、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期間の日付は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. ファイルに正しい値を入力したら `astra-support-bundle.yaml`、CRを適用します。

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

CLIを使用したサポートバンドルの作成

手順

1. サポートバンドルを作成し、角かっこ内の値を環境からの情報に置き換えます。は `trigger-type`、バンドルをすぐに作成するか、スケジュールによって作成時間が指定されているかを決定

し、または `Scheduled` を指定できます `Manual`。デフォルト設定は `Manual`。

例：

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

Trident プロテクトのアップグレード

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

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 をアップグレードします。

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

3. Trident プロテクトのアップグレード:

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

アプリケーションの管理と保護

Trident Protect AppVault オブジェクトを使用してバケットを管理する

Trident Protect のバケット カスタム リソース (CR) は、AppVault と呼ばれます。AppVault オブジェクトは、ストレージ バケットの宣言型 Kubernetes ワークフロー表現です。AppVault CR には、バックアップ、スナップショット、復元操作、SnapMirror レプリケーションなどの保護操作でバケットを使用するために必要な構成が含まれています。AppVault を作成できるのは管理者のみです。

キー生成とAppVault定義の例

AppVault CRを定義するときは、プロバイダがホストするリソースにアクセスするための資格情報を含める必要があります。クレデンシャルのキーの生成方法は、プロバイダによって異なります。次に、いくつかのプロバイダのコマンドラインキー生成の例を示します。次に、各プロバイダのAppVault定義の例を示します。

キー生成の例

次の例を使用して、各クラウドプロバイダのクレデンシャル用のキーを作成できます。

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
```

汎用 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の例

次のCR例を使用して、クラウドプロバイダごとにAppVaultオブジェクトを作成できます。

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
```

汎用 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
```

Trident Protect CLI を使用した AppVault 作成例

次のCLIコマンド例を使用して、プロバイダごとにAppVault CRSを作成できます。

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>
```

汎用 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>
```

AppVault ブラウザを使用して AppVault 情報を表示する

Trident Protect CLI プラグインを使用して、クラスター上に作成された AppVault オブジェクトに関する情報を表示できます。

手順

1. AppVault オブジェクトの内容を表示します。

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

出力例：

```

+-----+-----+-----+-----+
+-----+
| CLUSTER | APP | TYPE | NAME |
+-----+-----+-----+-----+
|          | 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. 必要に応じて、各リソースのAppVaultPathを表示するには、フラグを使用し `--show-paths` ます。

表の最初の列にあるクラスター名は、Trident Protect Helm インストールでクラスター名が指定された場合にのみ使用できます。例えば： `--set clusterName=production1`。

AppVaultの削除

AppVaultオブジェクトはいつでも削除できます。



AppVaultオブジェクトを削除する前に、AppVault CRのキーを削除しないで `finalizers` ください。これを行うと、AppVaultバケット内のデータが残り、クラスター内のリソースが孤立する可能性があります。

開始する前に

関連付けられているバケットに格納されているSnapshotとバックアップをすべて削除しておきます。

Kubernetes CLIを使用したAppVaultの削除

1. AppVaultオブジェクトを削除し、削除するAppVaultオブジェクトの名前に置き換え `appvault_name` ます。

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

Trident Protect CLI を使用して AppVault を削除する

1. AppVaultオブジェクトを削除し、削除するAppVaultオブジェクトの名前に置き換え `appvault_name` ます。

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

Trident Protectで管理するアプリケーションを定義する

アプリケーション CR と関連する AppVault CR を作成することで、Trident Protect で管理するアプリケーションを定義できます。

AppVault CRの作成

アプリケーションでデータ保護操作を実行するときに使用する AppVault CR を作成する必要があります。また、AppVault CR は、Trident Protect がインストールされているクラスター上に存在する必要があります。AppVault CRは環境に固有のものです。AppVault CRの例については、以下を参照してください。"[AppVaultカスタムリソース。](#)"

アプリケーションの定義

Trident Protect で管理する各アプリケーションを定義する必要があります。アプリケーション CR を手動で作成するか、Trident Protect CLI を使用して、管理対象のアプリケーションを定義できます。

CRを使用したアプリケーションの追加

手順

1. デスティネーションアプリケーションのCRファイルを作成します。
 - a. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: maria-app.yaml)。
 - b. 次の属性を設定します。
 - `* metadata.name*`: (*required*) アプリケーションカスタムリソースの名前。保護操作に必要な他のCRファイルがこの値を参照するため、選択した名前をメモします。
 - `* spec.includedNamespaces*`: (*required*) 名前空間ラベルまたは名前空間名を使用して、アプリケーションリソースが存在する名前空間を指定します。アプリケーション名前空間は、このリストの一部である必要があります。

YAMLの例:

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

2. 環境に合わせてアプリケーションCRを作成したら、CRを適用します。例:

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

CLIを使用したアプリケーションの追加

手順

1. アプリケーション定義を作成して適用し、括弧内の値を環境からの情報に置き換えます。次の例に示す引数を持つカンマ区切りリストを使用して、名前空間とリソースをアプリケーション定義に含めることができます。

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

Trident Protectを使用してアプリケーションを保護する

自動保護ポリシーを使用するかアドホック ベースでスナップショットやバックアップを取得することにより、Trident Protect によって管理されるすべてのアプリを保護できます。



データ保護操作中にファイルシステムをフリーズおよびフリーズ解除するようにTrident Protect を構成できます。["Trident Protect によるファイルシステムのフリーズ設定の詳細"](#)。

オンデマンドスナップショットを作成します

オンデマンド Snapshot はいつでも作成できます。

CRを使用したスナップショットの作成

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-snapshot-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*:` (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.applicationRef *:` スナップショットを作成するアプリケーションのKubernetes名。
 - `* spec.appVaultRef *:` (*required*) スナップショットの内容 (メタデータ) を格納するAppVaultの名前。
 - `* spec.reclaimPolicy *:` (*_Optional_*) スナップショットCRが削除されたときのスナップショットのAppArchiveの動作を定義します。つまり、に設定しても `Retain` Snapshotは削除されます。有効なオプション：
 - Retain (デフォルト)
 - 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. ファイルに正しい値を入力したら `trident-protect-snapshot-cr.yaml`、CRを適用します。

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

CLIを使用したスナップショットの作成

手順

1. スナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

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

オンデマンドバックアップの作成

アプリはいつでもバックアップできます。

CRを使用したバックアップの作成

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-backup-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*:` (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.applicationRef *:` (*required*) バックアップするアプリケーションのKubernetes名。
 - `* spec.appVaultRef *:` (*required*) バックアップ内容を格納するAppVaultの名前。
 - `*spec.DataMover *:(Optional)`バックアップ操作に使用するバックアップツールを示す文字列。有効な値 (大文字と小文字が区別されます) :
 - Restic
 - Kopia (デフォルト)
 - `* spec.reclaimPolicy *:` (*_Optional_*) 要求から解放されたバックアップの処理を定義します。有効な値:
 - Delete
 - Retain (デフォルト)
 - `* Spec.snapshotRef *:` (オプション) :バックアップのソースとして使用するSnapshotの名前。指定しない場合は、一時Snapshotが作成されてバックアップされます。

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

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

CLIを使用したバックアップの作成

手順

1. バックアップを作成します。角かっこ内の値は、使用している環境の情報に置き換えます。例:

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

データ保護スケジュールを作成

保護ポリシーは、定義されたスケジュールでスナップショット、バックアップ、またはその両方を作成することでアプリケーションを保護します。Snapshotとバックアップを毎時、日次、週次、および月単位で作成し、保持するコピーの数を指定できます。

CRを使用したスケジュールの作成

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-schedule-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name *`: (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.DataMover *:`(*Optional*)バックアップ操作に使用するバックアップツールを示す文字列。有効な値 (大文字と小文字が区別されます) :
 - Restic
 - Kopia (デフォルト)
 - `* spec.applicationRef *`: バックアップするアプリケーションのKubernetes名。
 - `* spec.appVaultRef *`: (*required*) バックアップ内容を格納するAppVaultの名前。
 - `* spec.backupRetention *`: 保持するバックアップの数。ゼロは、バックアップを作成しないことを示します。
 - `* spec.snapshotRetention *`: 保持するSnapshotの数。ゼロは、スナップショットを作成しないことを示します。
 - `* spec.granularity*`:スケジュールを実行する頻度。指定可能な値と必須の関連フィールドは次のとおりです。
 - `hourly` (を指定する必要がある `spec.minute` ます)
 - `daily` (とを指定する必要がある `spec.minute` `spec.hour` ます)。
 - `weekly` (および `spec.dayOfWeek` を指定する必要がある `spec.minute, spec.hour` ます)。
 - `monthly` (および `spec.dayOfMonth` を指定する必要がある `spec.minute, spec.hour` ます)。
 - `* spec.dayOfMonth *`: (*_Optional_*) スケジュールを実行する月の日 (1~31)。粒度がに設定されている場合、このフィールドは必須 `monthly` です。
 - `* spec.DayOfWeek *`: (*_Optional_*) スケジュールを実行する曜日 (0~7)。0または7の値は日曜日を示します。粒度がに設定されている場合、このフィールドは必須 `weekly` です。
 - `* spec.hour *`: (*_Optional_*) スケジュールを実行する時刻 (0~23)。粒度が、またはに設定されている場合、このフィールドは必須 `daily weekly` `monthly` です。
 - `* spec.minute *`: (*_Optional_*) スケジュールを実行する分 (0~59)。このフィールドは、粒度が、、、またはに設定されている場合は必須 `hourly daily weekly` `monthly` です。

```

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

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

CLIを使用してスケジュールを作成する

手順

1. 保護スケジュールを作成し、角かっこ内の値を環境からの情報に置き換えます。例：



を使用すると、このコマンドの詳細なヘルプ情報を表示できます `tridentctl-protect create schedule --help`。

```

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>

```

Snapshot を削除します

不要になったスケジュール済みまたはオンデマンドの Snapshot を削除します。

手順

1. Snapshotに関連付けられているSnapshot CRを削除します。

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

バックアップを削除します

不要になったスケジュール済みまたはオンデマンドのバックアップを削除します。

手順

1. バックアップに関連付けられているバックアップCRを削除します。

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

バックアップ処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したバックアップ処理のステータスを確認できます。

手順

1. 次のコマンドを使用してバックアップ処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

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

azure-anf-files NetApp (ANF) 処理のバックアップとリストアを実現

Trident Protect をインストールしている場合は、azure-netapp-files ストレージ クラスを使用し、Trident 24.06 より前に作成されたストレージ バックエンドに対して、スペース効率の高いバックアップと復元機能を有効にすることができます。この機能は NFSv4 ボリュームで動作し、容量プールから追加のスペースを消費しません。

開始する前に

次の点を確認します。

- Trident Protect をインストールしました。
- Trident Protect でアプリケーションを定義しました。この手順を完了するまで、このアプリケーションの保護機能は制限されます。
- ストレージバックエンドのデフォルトのストレージクラスとしてを選択し、azure-netapp-files ました。

1. Trident 24.10にアップグレードする前にANFボリュームを作成した場合は、Tridentで次の手順を実行します。

- a. アプリケーションに関連付けられているNetAppファイルベースの各PVのSnapshotディレクトリを有効にします。

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

- b. 関連付けられている各PVに対してSnapshotディレクトリが有効になっていることを確認します。

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

応答：

```
snapshotDirectory: "true"
```

+

スナップショットディレクトリが有効になっていない場合、Trident Protectは通常のバックアップ機能を選択し、バックアッププロセス中に容量プールのスペースを一時的に消費します。この場合、バックアップ対象のボリュームのサイズの一時ボリュームを作成するために十分なスペースが容量プールにあることを確認してください。

結果

アプリケーションは、Trident Protectを使用してバックアップおよび復元する準備ができています。各PVCは、バックアップや復元のために他のアプリケーションでも使用できます。

Trident Protectを使用してアプリケーションを復元する

Trident Protectを使用して、スナップショットまたはバックアップからアプリケーションを復元できます。アプリケーションを同じクラスターに復元する場合、既存のスナップショットからの復元の方が高速になります。



アプリケーションを復元すると、そのアプリケーションに設定されているすべての実行フックがアプリケーションとともに復元されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。

リストア処理とフェイルオーバー処理時のネームスペースのアノテーションとラベル

リストア処理とフェイルオーバー処理では、デスティネーションネームスペースのラベルとアノテーションがソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネーム

スペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、すでに存在するラベルまたはアノテーションがソースネームスペースの値に一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルやアノテーションは変更されません。



RedHat OpenShiftを使用する場合は、OpenShift環境でのネームスペースのアノテーションの重要な役割に注意することが重要です。ネームスペースのアノテーションを使用すると、リストアしたポッドがOpenShift Security Context Constraint (SCC; セキュリティコンテキスト制約) で定義された適切な権限とセキュリティ設定に従っていることが確認され、権限の問題なしにボリュームにアクセスできるようになります。詳細については、を参照して "[OpenShiftセキュリティコンテキスト制約に関するドキュメント](#)" ください。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数を設定することで、デスティネーションネームスペースの特定のアノテーションが上書きされないようにすることができます

RESTORE_SKIP_NAMESPACE_ANNOTATIONS。例：

```
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>
```

ソースアプリケーションをHelmを使用してインストールした場合、`--create-namespace` 旗には特別な扱いが与えられます `name` ラベルキー。復元またはフェイルオーバー プロセス中に、Trident Protectはこのラベルを宛先名前空間にコピーしますが、ソースからの値がソース名前空間と一致する場合は、値を宛先名前空間の値に更新します。この値がソース名前空間と一致しない場合は、変更されずに宛先名前空間にコピーされません。

例

次の例は、ソースとデスティネーションのネームスペースを示しています。それぞれにアノテーションとラベルが設定されています。処理の前後のデスティネーションネームスペースの状態、およびデスティネーションネームスペースでアノテーションやラベルを組み合わせた上書きしたりする方法を確認できます。

リストアまたはフェイルオーバー処理の前

次の表に、リストアまたはフェイルオーバー処理を実行する前のソースネームスペースとデスティネーションネームスペースの状態を示します。

ネームスペース	アノテーション	ラベル
ネームスペースns-1 (ソース)	<ul style="list-style-type: none">• Annotation.one/key : "UpdatedValue"• Annotation.Two/key : "true"	<ul style="list-style-type: none">• 環境=本番• コンプライアンス= HIPAA• 名前= ns-1
ネームスペースns-2 (デスティネーション)	<ul style="list-style-type: none">• Annotation.one/key : "true"• annotation.three/key : "false"	<ul style="list-style-type: none">• ロール=データベース

リストア処理後

次の表に、リストアまたはフェイルオーバー処理後の例のデスティネーション名前空間の状態を示します。一部のキーが追加され、一部のキーが上書きされ、`name` デスティネーション名前空間に一致するようにラベルが更新されました。

名前空間	アノテーション	ラベル
名前空間ns-2 (デスティネーション)	<ul style="list-style-type: none">• Annotation.one/key: "UpdatedValue"• Annotation.Two/key: "true"• annotation.three/key: "false"	<ul style="list-style-type: none">• 名前= ns-2• コンプライアンス= HIPAA• 環境=本番• ロール=データベース

バックアップから別の名前空間へのリストア

BackupRestore CR を使用してバックアップを別の名前空間に復元すると、Trident Protect はアプリケーションを新しい名前空間に復元し、復元されたアプリケーションのアプリケーション CR を作成します。復元されたアプリケーションを保護するには、オンデマンド バックアップまたはスナップショットを作成するか、保護スケジュールを確立します。



既存のリソースがある別の名前空間にバックアップをリストアしても、バックアップ内のリソースと名前を共有するリソースは変更されません。バックアップ内のすべてのリソースをリストアするには、ターゲット名前空間を削除して再作成するか、新しい名前空間にバックアップをリストアします。

CRの使用

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-backup-restore-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - *** metadata.name*:** (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - **spec.appArchivePath:**バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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

- *** spec.appVaultRef*:** (*required*) バックアップコンテンツが格納されているAppVaultの名前。
- *** spec.namespaceMapping*:**リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace` を、使用している環境の情報に置き換え `my-source-namespace` ます。
- *** spec.storageClassMapping*:** リストア処理のソースストレージクラスからデスティネーションストレージクラスへのマッピング。および `sourceStorageClass` を、使用している環境の情報に置き換え `destinationStorageClass` ます。

```
---  
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. (オプション) リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
 - **resourceFilter.resourceSelectionCriteria:**(フィルタリングに必要) resourceMatchersで定義されたリソースを使用 `Include` または `Exclude` 除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
 - **resourceFilter.resourceMatchers:** resourceMatcherオブジェクトの配列。この配列に複数の

要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。

- `*resourceMatchers[].group` *:(*Optional*)フィルタリングするリソースのグループ。
- `*resourceMatchers[].kind` *:(*optional*)フィルタリングするリソースの種類。
- **`resourceMatchers[].version`**:(*Optional*)フィルタリングするリソースのバージョン。
- `*resourceMatchers[].names` * : (*optional*) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
- `*resourceMatchers[].namespaces` *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- `*resourceMatchers[].labelSelectors` *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "[Kubernetes のドキュメント](#)"。例：`"trident.netapp.io/os=linux"`。

例：

```
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. ファイルに正しい値を入力したら `trident-protect-backup-restore-cr.yaml`、CRを適用します。

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

CLI を使用します

手順

1. バックアップを別の名前スペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。`namespace-mapping` 引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2` ます。例：
：

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

バックアップから元のネームスペースへのリストア

バックアップはいつでも元のネームスペースにリストアできます。

CRの使用

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-backup-ipr-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。

- *** metadata.name*:** (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
- **spec.appArchivePath:**バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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

- *** spec.appVaultRef*:** (*required*) バックアップコンテンツが格納されているAppVaultの名前。

例:

```
---  
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. (オプション) リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
 - **resourceFilter.resourceSelectionCriteria:**(フィルタリングに必要) resourceMatchersで定義されたリソースを使用 `Include` または `Exclude` 除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
 - **resourceFilter.resourceMatchers:** resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド (グループ、種類、バージョン) はAND演算として照合されます。
 - ***resourceMatchers[].group*:**(*Optional*)フィルタリングするリソースのグループ。
 - ***resourceMatchers[].kind*:**(*optional*)フィルタリングするリソースの種類。
 - **resourceMatchers[].version:**(*Optional*)フィルタリングするリソースのバージョン。
 - *** resourceMatchers[].names*:** (*optional*) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。

- `*resourceMatchers[].namespaces` *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- `*resourceMatchers[].labelSelectors` *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例: `"trident.netapp.io/os=linux"`。

例:

```
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. ファイルに正しい値を入力したら `trident-protect-backup-ipr-cr.yaml`、CRを適用します。

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

CLI を使用します

手順

1. バックアップを元の名前スペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。この ``backup`` 引数では、という形式の名スペースとバックアップ名を使用し ``<namespace>/<name>`` ます。例:

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

バックアップから別のクラスタへのリストア

元のクラスタで問題が発生した場合は、バックアップを別のクラスタにリストアできます。

- 始める前に *

次の前提条件が満たされていることを確認します。

- 宛先クラスターにはTrident Protect がインストールされています。
- デスティネーションクラスターは、バックアップが格納されているソースクラスターと同じAppVaultのバケットパスにアクセスできます。

手順

1. Trident Protect CLI プラグインを使用して、宛先クラスター上の AppVault CR の可用性を確認します。

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



アプリケーションのリストア用の名前スペースがデスティネーションクラスターに存在することを確認します。

2. デスティネーションクラスターから使用可能なAppVaultのバックアップ内容を表示します。

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

このコマンドを実行すると、AppVaultで使用可能なバックアップが表示されます。これには、元のクラスター、対応するアプリケーション名、タイムスタンプ、アーカイブパスが含まれます。

出力例：

```
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| 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. AppVault名とアーカイブパスを使用して、アプリケーションをデスティネーションクラスターにリストアします。

CRの使用

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-backup-restore-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*`: (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.appVaultRef *`: (*required*) バックアップコンテンツが格納されているAppVaultの名前。
 - `spec.appArchivePath`: バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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



BackupRestore CRを使用できない場合は、手順2のコマンドを使用してバックアップの内容を表示できます。

- `* spec.namespaceMapping*`: リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace` を、使用している環境の情報に置き換え `my-source-namespace` ます。

例：

```
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"}]
```

3. ファイルに正しい値を入力したら `trident-protect-backup-restore-cr.yaml`、CRを適用します。

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

CLI を使用します

1. 次のコマンドを使用してアプリケーションをリストアし、括弧内の値を環境の情報に置き換えます。namespace-mapping引数では、コロンで区切られた名前空間を使用して、ソース名前空間

をsource1:dest1、source2:dest2の形式で正しいデスティネーション名前空間にマッピングします。
例：

```
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>
```

Snapshotから別のネームスペースへのリストア

カスタム リソース (CR) ファイルを使用して、スナップショットからデータを別の名前空間または元のソース名前空間に復元できます。SnapshotRestore CR を使用してスナップショットを別の名前空間に復元すると、Trident Protect はアプリケーションを新しい名前空間に復元し、復元されたアプリケーションのアプリケーション CR を作成します。復元されたアプリケーションを保護するには、オンデマンド バックアップまたはスナップショットを作成するか、保護スケジュールを確立します。

CRの使用

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*`: (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.appVaultRef *`: (*required*) スナップショットコンテンツが格納されているAppVaultの名前。
 - `* spec.appArchivePath *`: スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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

- `* spec.namespaceMapping*`: リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace` を、使用している環境の情報に置き換え `my-source-namespace` ます。
- `* spec.storageClassMapping *`: リストア処理のソースストレージクラスからデスティネーションストレージクラスへのマッピング。および `sourceStorageClass` を、使用している環境の情報に置き換え `destinationStorageClass` ます。

```
---  
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. (オプション) リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
 - **resourceFilter.resourceSelectionCriteria**: (フィルタリングに必要) resourceMatchersで定義されたリソースを使用 `Include` または `Exclude` 除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。

- **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
 - *resourceMatchers[].group *:(Optional)フィルタリングするリソースのグループ。
 - *resourceMatchers[].kind *:(optional)フィルタリングするリソースの種類。
 - **resourceMatchers[].version**:(Optional)フィルタリングするリソースのバージョン。
 - *resourceMatchers[].names * : (optional) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
 - *resourceMatchers[].namespaces *:(optional)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
 - *resourceMatchers[].labelSelectors *:(Optional)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "[Kubernetes のドキュメント](#)"。例：
"trident.netapp.io/os=linux"。

例：

```
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. ファイルに正しい値を入力したら trident-protect-snapshot-restore-cr.yaml、CRを適用します。

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

CLI を使用します

手順

1. スナップショットを別の名前スペースにリストアし、括弧内の値を環境の情報に置き換えます。
 - `snapshot` 引数では、という形式の名前スペースとSnapshot名を使用し

`<namespace>/<name>`ます。

- `namespace-mapping` 引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2` ます。

例：

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

Snapshotから元のネームスペースへのリストア

Snapshotはいつでも元のネームスペースにリストアできます。

CRの使用

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-snapshot-ipr-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*:` (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.appVaultRef*:` (*required*) スナップショットコンテンツが格納されているAppVaultの名前。
 - `* spec.appArchivePath*:` スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

```
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. (オプション) リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
 - **resourceFilter.resourceSelectionCriteria:**(フィルタリングに必要) resourceMatchersで定義されたリソースを使用 `Include` または `Exclude` 除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
 - **resourceFilter.resourceMatchers:** resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド (グループ、種類、バージョン) はAND演算として照合されます。
 - `*resourceMatchers[].group*:` (*Optional*) フィルタリングするリソースのグループ。
 - `*resourceMatchers[].kind*:` (*optional*) フィルタリングするリソースの種類。
 - **resourceMatchers[].version:** (*Optional*) フィルタリングするリソースのバージョン。
 - `* resourceMatchers[].names*:` (*optional*) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
 - `*resourceMatchers[].namespaces*:` (*optional*) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。

- `*resourceMatchers[].labelSelectors` *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：
`"trident.netapp.io/os=linux"`。

例：

```
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. ファイルに正しい値を入力したら `trident-protect-snapshot-ipr-cr.yaml`、CRを適用します。

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

CLI を使用します

手順

1. Snapshotを元のネームスペースにリストアします。括弧内の値は、環境の情報に置き換えてください。例：

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

リストア処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したリストア処理のステータスを確認できます。

手順

1. 次のコマンドを使用してリストア処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

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

NetApp SnapMirrorとTrident Protectを使用してアプリケーションを複製する

Trident Protect を使用すると、NetApp SnapMirrorテクノロジーの非同期レプリケーション機能を使用して、データとアプリケーションの変更を、同じクラスター上または異なるクラスター間で、あるストレージ バックエンドから別のストレージ バックエンドに複製できます。

リストア処理とフェイルオーバー処理時のネームスペースのアノテーションとラベル

リストア処理とフェイルオーバー処理では、デスティネーションネームスペースのラベルとアノテーションがソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネームスペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、すでに存在するラベルまたはアノテーションがソースネームスペースの値に一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルやアノテーションは変更されません。



RedHat OpenShiftを使用する場合は、OpenShift環境でのネームスペースのアノテーションの重要な役割に注意することが重要です。ネームスペースのアノテーションを使用すると、リストアしたポッドがOpenShift Security Context Constraint (SCC；セキュリティコンテキスト制約) で定義された適切な権限とセキュリティ設定に従っていることが確認され、権限の問題なしにボリュームにアクセスできるようになります。詳細については、を参照して "[OpenShiftセキュリティコンテキスト制約に関するドキュメント](#)" ください。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数を設定することで、デスティネーションネームスペースの特定のアノテーションが上書きされないようにすることができます

RESTORE_SKIP_NAMESPACE_ANNOTATIONS。例：

```
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>
```

ソースアプリケーションをHelmを使用してインストールした場合、`--create-namespace` 旗には特別な扱いが与えられます `name` ラベルキー。復元またはフェイルオーバー プロセス中に、Trident Protect はこのラベルを宛先名前空間にコピーしますが、ソースからの値がソース名前空間と一致する場合は、値を宛先名前空間の値に更新します。この値がソース名前空間と一致しない場合は、変更されずに宛先名前空間にコピーされます。

例

次の例は、ソースとデスティネーションのネームスペースを示しています。それぞれにアノテーションとラベルが設定されています。処理の前後のデスティネーションネームスペースの状態、およびデスティネーションネームスペースでアノテーションやラベルを組み合わせたり上書きしたりする方法を確認できます。

リストアまたはフェイルオーバー処理の前

次の表に、リストアまたはフェイルオーバー処理を実行する前のソース名前スペースとデスティネーション名前スペースの状態を示します。

名前スペース	アノテーション	ラベル
名前スペースns-1 (ソース)	<ul style="list-style-type: none">• Annotation.one/key : "UpdatedValue"• Annotation.Two/key : "true"	<ul style="list-style-type: none">• 環境=本番• コンプライアンス= HIPAA• 名前= ns-1
名前スペースns-2 (デスティネーション)	<ul style="list-style-type: none">• Annotation.one/key : "true"• annotation.three/key : "false"	<ul style="list-style-type: none">• ロール=データベース

リストア処理後

次の表に、リストアまたはフェイルオーバー処理後の例のデスティネーション名前スペースの状態を示します。一部のキーが追加され、一部のキーが上書きされ、`name`デスティネーション名前スペースに一致するようにラベルが更新されました。

名前スペース	アノテーション	ラベル
名前スペースns-2 (デスティネーション)	<ul style="list-style-type: none">• Annotation.one/key : "UpdatedValue"• Annotation.Two/key : "true"• annotation.three/key : "false"	<ul style="list-style-type: none">• 名前= ns-2• コンプライアンス= HIPAA• 環境=本番• ロール=データベース



データ保護操作中にファイルシステムをフリーズおよびフリーズ解除するようにTrident Protectを構成できます。["Trident Protect によるファイルシステムのフリーズ設定の詳細"](#)。

レプリケーション関係を設定

レプリケーション関係の設定には、次の作業が含まれます。

- Trident Protect がアプリのスナップショット (アプリの Kubernetes リソースとアプリの各ボリュームのボリューム スナップショットが含まれます) を作成する頻度を選択します。
- レプリケーションスケジュールの選択 (Kubernetesリソースと永続ボリュームデータを含む)
- Snapshotの作成時間の設定

手順

1. ソースクラスタにソースアプリケーション用のAppVaultを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します["AppVaultカスタムリソース"](#)。

CRを使用したAppVaultの作成

- a. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-appvault-primary-source.yaml`) 。
- b. 次の属性を設定します。
 - `* metadata.name*`: (*required*) AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
 - `* spec.providerConfig*`: (*required*) は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。bucketNameとプロバイダーに必要なその他の詳細を選択します。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"[AppVaultカスタムリソース](#)"。
 - `* spec.providerCredentials*`: (*required*) は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
 - `* spec.providerCredentials.valueFromSecret*`: (*required*) は、クレデンシャル値がシークレットから取得される必要があることを示します。
 - `* key *`: (*required*) 選択するシークレットの有効なキー。
 - `* name *`: (*required*) このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
 - `* spec.providerCredentials.secretAccessKey*`: (*required*) プロバイダへのアクセスに使用するアクセスキー。name は `spec.providerCredentials.valueFromSecret.name*`と一致している必要があります。
 - `* spec.providerType*`: (*required*) は、バックアップの提供元 (NetApp ONTAP S3、汎用 S3、Google Cloud、Microsoft Azureなど) を決定します。有効な値:
 - AWS
 - Azure
 - GCP
 - 汎用- s3
 - ONTAP - s3
 - StorageGRID - s3
- c. ファイルに正しい値を入力したら `trident-protect-appvault-primary-source.yaml`、CRを適用します。

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

CLIを使用したAppVaultの作成

- a. AppVaultを作成し、括弧内の値を環境からの情報に置き換えます。

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

2. ソースアプリケーションCRを作成します。

CRを使用したソースアプリケーションの作成

- a. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-app-source.yaml`)。
- b. 次の属性を設定します。

- `* metadata.name*`: (*required*) アプリケーションカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
- `* spec.includedNamespaces*`: (*required*) 名前空間と関連ラベルの配列。名前空間名を使用し、必要に応じてラベルを使用して名前空間の範囲を絞り込み、ここにリストされている名前空間に存在するリソースを指定します。アプリケーション名前空間は、この配列の一部である必要があります。
 - YAMLの例*:

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

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

CLIを使用したソースアプリケーションの作成

- a. ソースアプリケーションを作成します。例:

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

3. 必要に応じて、ソースアプリケーションのスナップショットを作成します。このSnapshotは、デスティネーションクラスタのアプリケーションのベースとして使用されます。この手順を省略した場合は、スケジュールされた次のSnapshotが実行されて最新のSnapshotが作成されるまで待つ必要があります。

CRを使用したスナップショットの作成

a. ソースアプリケーションのレプリケーションスケジュールを作成します。

i. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-schedule.yaml`)。

ii. 次の属性を設定します。

- `* metadata.name *: (required)` スケジュールカスタムリソースの名前。
- `* spec.AppVaultRef *: (required)` この値は、ソースアプリケーションのAppVaultの`metadata.name`フィールドと一致する必要があります。
- `* spec.ApplicationRef *: (required)` この値は、ソースアプリケーションCRの`metadata.name`フィールドと一致する必要があります。
- `* spec.backupRetention *: (required)` このフィールドは必須であり、値は0に設定する必要があります。
- `* spec.enabled *: true`に設定する必要があります。
- `* spec.granularity *:`はに設定する必要があります `Custom`。
- `* spec.recurrenceRule *:`開始日をUTC時間と繰り返し間隔で定義します。
- `* spec.snapshotRetention *:`を2に設定する必要があります。

YAMLの例：

```
---
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"
```

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

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

CLIを使用したスナップショットの作成

- a. スナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

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

4. ソースクラスタに適用したAppVault CRと同じソースアプリケーションAppVault CRをデスティネーションクラスタに作成し、という名前を付けます（例：trident-protect-appvault-primary-destination.yaml）。
5. CRを適用します。

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

6. デスティネーションクラスタにデスティネーションアプリケーション用のAppVaultを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します"[AppVaultカスタムリソース](#)".
 - a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例：trident-protect-appvault-secondary-destination.yaml）。
 - b. 次の属性を設定します。
 - * metadata.name*: (*required*) AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
 - * spec.providerConfig*: (*required*) は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。およびプロバイダに必要なその他の詳細情報を選択します bucketName。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"[AppVaultカスタムリソース](#)".
 - * spec.providerCredentials*: (*required*) は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
 - * spec.providerCredentials.valueFromSecret*: (*required*) は、クレデンシャル値がシークレットから取得される必要があることを示します。
 - * key * : (*required*) 選択するシークレットの有効なキー。
 - * name * : (*required*) このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
 - * spec.providerCredentials.secretAccessKey*: (*required*) プロバイダへのアクセスに使用するアクセスキー。name は spec.providerCredentials.valueFromSecret.name*と一致している必要があります。

- * spec.providerType*: (*required*) は、バックアップの提供元（NetApp ONTAP S3、汎用S3、Google Cloud、Microsoft Azureなど）を決定します。有効な値：
 - AWS
 - Azure
 - GCP
 - 汎用- s3
 - ONTAP - s3
 - StorageGRID - s3

c. ファイルに正しい値を入力したら `trident-protect-appvault-secondary-destination.yaml`、CRを適用します。

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

7. AppMirrorRelationship CRファイルを作成します。

CRを使用したAppMirrorRelationshipの作成

- a. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-relationship.yaml`)。
- b. 次の属性を設定します。
 - `* metadata.name:*` (必須) AppMirrorRelationshipカスタムリソースの名前。
 - `* spec.destinationAppVaultRef*`: (*required*) この値は、デスティネーションクラスタ上のデスティネーションアプリケーションのAppVaultの名前と一致する必要があります。
 - `* spec.namespaceMapping*:(required)`宛先およびソースの名前空間は、それぞれのアプリケーションCRで定義されているアプリケーション名前空間と一致している必要があります。
 - `*spec.sourceAppVaultRef *:(required)`この値は、ソースアプリケーションのAppVaultの名前と一致する必要があります。
 - `spec.sourceApplicationName:(required)`この値は、ソースアプリケーションCRで定義したソースアプリケーションの名前と一致する必要があります。
 - `* spec.storageClassName * :` (*required*) クラスタ上の有効なストレージクラスの名前を選択します。ソース環境とピア関係にあるONTAP Storage VMにストレージクラスをリンクする必要があります。
 - `*spec.recurrenceRule *:`開始日をUTC時間と繰り返し間隔で定義します。

YAMLの例：

```
---
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-vsims-2
```

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

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

CLIを使用したAppMirrorRelationshipの作成

- a. AppMirrorRelationshipオブジェクトを作成して適用し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl-protect create appmirrorrelationship  
<name_of_appmirrorrelationship> --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. (オプション) レプリケーション関係の状態とステータスを確認します。

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

デスティネーションクラスターへのフェイルオーバー

Trident Protect を使用すると、複製されたアプリケーションを宛先クラスターにフェイルオーバーできます。この手順により、レプリケーション関係が停止され、宛先クラスターでアプリがオンラインになります。Trident Protect は、ソース クラスター上のアプリが動作中であった場合、そのアプリを停止しません。

手順

1. AppMirrorRelationship CRファイル（など）を開き `trident-protect-relationship.yaml`、`*spec.desiredState*`の値をに変更します Promoted。
2. CR ファイルを保存します。
3. CRを適用します。

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

4. (オプション) フェイルオーバーされたアプリケーションに必要な保護スケジュールを作成します。
5. (オプション) レプリケーション関係の状態とステータスを確認します。

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

フェイルオーバーされたレプリケーション関係を再同期します。

再同期処理によってレプリケーション関係が再確立されます。再同期処理を実行すると、元のソースアプリケーションが実行中のアプリケーションになり、デスティネーションクラスタで実行中のアプリケーションに加えた変更は破棄されます。

このプロセスは、レプリケーションを再確立する前に、デスティネーションクラスタ上のアプリケーションを停止します。



フェイルオーバー中にデスティネーションアプリケーションに書き込まれたデータはすべて失われます。

手順

1. ソースアプリケーションのスナップショットを作成します。
2. AppMirrorRelationship CRファイル（など）を開き `trident-protect-relationship.yaml`、`spec.desiredState`の値をに変更します。 `Established`
3. CR ファイルを保存します。
4. CRを適用します。

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

5. フェイルオーバーされたアプリケーションを保護するためにデスティネーションクラスタで保護スケジュールを作成した場合は削除します。スケジュールが残っていると、ボリュームSnapshotが失敗します。

フェイルオーバーされたレプリケーション関係の逆再同期

フェイルオーバーされたレプリケーション関係を逆再同期すると、デスティネーションアプリケーションがソースアプリケーションになり、ソースがデスティネーションになります。フェイルオーバー中にデスティネーションアプリケーションに加えられた変更は保持されます。

手順

1. 元のデスティネーションクラスタでAppMirrorRelationship CRを削除します。これにより、デスティネーションがソースになります。新しいデスティネーションクラスタに保護スケジュールが残っている場合は削除します。
2. レプリケーション関係を設定するには、元々その関係を反対側のクラスタに設定するために使用したCRファイルを適用します。
3. 各クラスタでAppVault CRSの準備が完了していることを確認します。
4. 反対側のクラスタにレプリケーション関係を設定し、逆方向の値を設定します。

アプリケーションのレプリケーション方向を反転

レプリケーションの方向を逆にすると、Trident Protect は、元のソース ストレージ バックエンドへのレプリケーションを継続しながら、アプリケーションを宛先ストレージ バックエンドに移動します。Trident Protect は、ソース アプリケーションを停止し、宛先アプリにフェールオーバーする前にデータを宛先に複製します。

この状況では、ソースとデスティネーションを交換しようとしています。

手順

1. シャットダウンスナップショットを作成します。

CRを使用したシャットダウンスナップショットの作成

- a. ソースアプリケーションの保護ポリシースケジュールを無効にします。
- b. ShutdownSnapshot CRファイルを作成します。
 - i. カスタムリソース (CR) ファイルを作成し、という名前を付けます (例: `trident-protect-shutdownsnapshot.yaml`)。
 - ii. 次の属性を設定します。
 - `* metadata.name*:` (*required*) カスタムリソースの名前。
 - `*spec.AppVaultRef*:` (*required*) この値は、ソースアプリケーションのAppVaultの`metadata.name`フィールドと一致する必要があります。
 - `*spec.ApplicationRef*:` (*required*) この値は、ソースアプリケーションCRファイルの`metadata.name`フィールドと一致する必要があります。

YAMLの例:

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

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

CLIを使用したシャットダウンスナップショットの作成

- a. シャットダウンスナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例:

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

2. Snapshotが完了したら、Snapshotのステータスを取得します。

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

3. 次のコマンドを使用して* shutdownsnapshot.status.appArchivePath *の値を検索し、ファイルパスの最後の部分 (basenameとも呼ばれます。これは最後のスラッシュのあとのすべてになります) を記録します。

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

4. 次のように変更して、デスティネーションクラスタからソースクラスタへのフェイルオーバーを実行します。



フェイルオーバー手順のステップ2では、AppMirrorRelationship CRファイルにフィールドを含め、`spec.promotedSnapshot` その値を上記の手順3で記録したベースネームに設定します。

5. の逆再同期の手順を実行し[\[フェイルオーバーされたレプリケーション関係の逆再同期\]](#)ます。
6. 新しいソースクラスタで保護スケジュールを有効にします。

結果

リバースレプリケーションが実行されると、次の処理が実行されます。

- 元のソースアプリのKubernetesリソースのスナップショットが作成されます。
- 元のソースアプリケーションのポッドは、アプリケーションのKubernetesリソースを削除することで正常に停止されます (PVCとPVはそのまま維持されます)。
- ポッドがシャットダウンされると、アプリのボリュームのスナップショットが取得され、レプリケートされます。
- SnapMirror関係が解除され、デスティネーションボリュームが読み取り/書き込み可能な状態になります。
- アプリのKubernetesリソースは、元のソースアプリがシャットダウンされた後に複製されたボリュームデータを使用して、シャットダウン前のスナップショットから復元されます。
- 逆方向にレプリケーションが再確立されます。

アプリケーションを元のソースクラスタにフェイルバックします

Trident Protect を使用すると、次の一連の操作によって、フェイルオーバー操作後の「フェイルバック」を実現できます。元のレプリケーション方向を復元するこのワークフローでは、Trident Protect は、レプリケーション方向を反転する前に、アプリケーションの変更を元のソース アプリケーションに複製 (再同期) します。

このプロセスは、デスティネーションへのフェイルオーバーが完了した関係から開始し、次の手順を実行します。

- フェイルオーバー状態から開始します。

- レプリケーション関係を逆再同期します。



通常の再同期操作は実行しないでください。フェイルオーバー中にデスティネーションクラスタに書き込まれたデータが破棄されます。

- レプリケーションの方向を逆にします。

手順

1. 手順を実行します[フェイルオーバーされたレプリケーション関係の逆再同期]。
2. 手順を実行します[アプリケーションのレプリケーション方向を反転]。

レプリケーション関係を削除する

レプリケーション関係はいつでも削除できます。アプリケーションレプリケーション関係を削除すると、2つの別々のアプリケーションが作成され、それらのアプリケーション間に関係がなくなります。

手順

1. AppMirrorRelationship CRを削除します。

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

Trident Protectを使用してアプリケーションを移行する

バックアップデータまたはSnapshotデータを別のクラスタまたはストレージクラスにリストアすることで、クラスタ間またはストレージクラス間でアプリケーションを移行できます。



アプリケーションを移行すると、そのアプリケーション用に構成されたすべての実行フックがアプリケーションとともに移行されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。

バックアップとリストアの処理

次のシナリオでバックアップとリストアの処理を実行するには、特定のバックアップとリストアのタスクを自動化します。

同じクラスタにクローニング

アプリケーションを同じクラスタにクローニングするには、Snapshotまたはバックアップを作成し、同じクラスタにデータをリストアします。

手順

1. 次のいずれかを実行します。
 - a. "Snapshotを作成します"です。
 - b. "バックアップを作成します"です。

2. Snapshotとバックアップのどちらを作成したかに応じて、同じクラスタで次のいずれかを実行します。
 - a. "スナップショットからデータをリストア"です。
 - b. "バックアップからデータをリストア"です。

別のクラスタにクローニング

アプリケーションを別のクラスターに複製するには (クラスター間複製を実行する)、ソース クラスターでバックアップを作成し、そのバックアップを別のクラスターに復元します。宛先クラスターにTrident Protect がインストールされていることを確認します。



を使用して、異なるクラスタ間でアプリケーションをレプリケートできます"[SnapMirrorレプリケーション](#)"。

手順

1. "バックアップを作成します"です。
2. バックアップを含むオブジェクトストレージバケットのAppVault CRがデスティネーションクラスタで設定されていることを確認します。
3. デスティネーションクラスタで、"バックアップからデータをリストア"を実行します。

あるストレージクラスから別のストレージクラスへのアプリケーションの移行

スナップショットを別のデスティネーションストレージクラスにリストアすることで、あるストレージクラスから別のストレージクラスにアプリケーションを移行できます。

たとえば、次のようになります (リストアCRのシークレットを除く)。

```
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
```

CRを使用したスナップショットのリストア

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
 - `* metadata.name*:` (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.appArchivePath*:` スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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

- `* spec.appVaultRef*:` (*required*) スナップショットコンテンツが格納されているAppVaultの名前。
- `* spec.namespaceMapping*:` リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace` を、使用している環境の情報に置き換え `my-source-namespace` ます。

```
---  
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. リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
 - **resourceFilter.resourceSelectionCriteria:**(フィルタリングに必要) `include` or `exclude` `resourceMatchers`で定義されたリソースを含めるか除外するかを指定します。次の `resourceMatchers`パラメータを追加して、追加または除外するリソースを定義します。
 - **resourceFilter.resourceMatchers:** `resourceMatcher`オブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
 - `*resourceMatchers[].group*:(Optional)`フィルタリングするリソースのグループ。
 - `*resourceMatchers[].kind*:(optional)`フィルタリングするリソースの種類。

- `resourceMatchers[].version`:(*Optional*)フィルタリングするリソースのバージョン。
- `* resourceMatchers[].names *`: (*optional*) フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
- `*resourceMatchers[].namespaces *`:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- `*resourceMatchers[].labelSelectors *`:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "[Kubernetes のドキュメント](#)"。例: `"trident.netapp.io/os=linux"`。

例:

```
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. ファイルに正しい値を入力したら `trident-protect-snapshot-restore-cr.yaml`、CRを適用します。

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

CLIを使用したスナップショットのリストア

手順

1. スナップショットを別の名前スペースにリストアし、括弧内の値を環境の情報に置き換えます。
 - ``snapshot``引数では、という形式の名前スペースとSnapshot名を使用し ``<namespace>/<name>``ます。
 - ``namespace-mapping``引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし ``source1:dest1,source2:dest2``ます。

例:

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

Trident Protect実行フックを管理する

実行フックは、管理対象アプリケーションのデータ保護操作と組み合わせて実行するように構成できるカスタムアクションです。たとえば、データベースアプリケーションがある場合、実行フックを使用して、スナップショットの前にすべてのデータベーストランザクションを一時停止し、スナップショットの完了後にトランザクションを再開できます。これにより、アプリケーションと整合性のある Snapshot を作成できます。

実行フックのタイプ

Trident Protect は、実行できるタイミングに基づいて、次のタイプの実行フックをサポートしています。

- Snapshot前
- Snapshot後
- バックアップ前
- バックアップ後
- リストア後のPOSTコマンドです
- フェイルオーバー後

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

1. 適用可能なカスタムプリオペレーション実行フックは、適切なコンテナで実行されます。カスタムのプリオペレーションフックは必要なだけ作成して実行できますが、操作前のこれらのフックの実行順序は保証も構成もされていません。
2. 該当する場合、ファイルシステムのフリーズが発生します。["Trident Protect によるファイルシステムのフリーズ設定の詳細"](#)。
3. データ保護処理が実行されます。
4. フリーズされたファイルシステムは、該当する場合はフリーズ解除されます。
5. 適用可能なカスタムポストオペレーション実行フックは、適切なコンテナで実行されます。必要な数のカスタムポストオペレーションフックを作成して実行できますが、操作後のこれらのフックの実行順序は保証されず、設定もできません。

同じ種類の実行フック（スナップショット前など）を複数作成する場合、これらのフックの実行順序は保証されません。ただし、異なるタイプのフックの実行順序は保証されています。たとえば'以下は'すべての異なるタイプのフックを持つ構成の実行順序です

1. スナップショット前フックが実行されます

2. スナップショット後フックが実行されます
3. 予備フックが実行されます
4. バックアップ後のフックが実行されます



上記の順序の例は、既存のSnapshotを使用しないバックアップを実行する場合にのみ該当します。



本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にしたら、作成されたSnapshotとバックアップをテストして整合性があることを確認します。これを行うには、アプリケーションを一時的な名前スペースにクローニングし、スナップショットまたはバックアップをリストアしてから、アプリケーションをテストします。

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

- 実行フックは、スクリプトを使用してアクションを実行する必要があります。多くの実行フックは、同じスクリプトを参照できます。
- Trident Protect では、実行フックが使用するスクリプトを実行可能なシェル スクリプトの形式で記述する必要があります。
- スクリプトのサイズは96KBに制限されています。
- Trident Protect は、実行フックの設定と一致する基準を使用して、スナップショット、バックアップ、または復元操作に適用可能なフックを決定します。



実行フックは、実行中のアプリケーションの機能を低下させたり、完全に無効にしたりすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。実行フックが関連付けられている状態でバックアップまたはスナップショット操作を開始した後キャンセルした場合でもバックアップまたはスナップショット操作がすでに開始されている場合はフックは実行できますつまり、バックアップ後の実行フックで使用されるロジックは、バックアップが完了したとは見なされません。

実行フックフィルタ

アプリケーションの実行フックを追加または編集するときに、実行フックにフィルタを追加して、フックが一致するコンテナを管理できます。フィルタは、すべてのコンテナで同じコンテナイメージを使用し、各イメージを別の目的（Elasticsearchなど）に使用するアプリケーションに便利です。フィルタを使用すると、一部の同一コンテナで実行フックが実行されるシナリオを作成できます。1つの実行フックに対して複数のフィルタを作成すると、それらは論理AND演算子と結合されます。実行フックごとに最大10個のアクティブフィルタを使用できます。

実行フックに追加する各フィルターは、正規表現を使用してクラスター内のコンテナを照合します。フックがコンテナに一致すると、フックはそのコンテナ上で関連付けられたスクリプトを実行します。フィルターの正規表現では正規表現 2 (RE2) 構文が使用されますが、一致リストからコンテナを除外するフィルターの作成はサポートされていません。Trident Protectが実行フックフィルタでサポートする正規表現の構文については、以下を参照してください。"[正規表現2 \(RE2\) 構文のサポート](#)"。



リストアまたはクローン処理のあとに実行される実行フックにネームスペースフィルタを追加し、リストアまたはクローンのソースとデスティネーションが異なるネームスペースにある場合、ネームスペースフィルタはデスティネーションネームスペースにのみ適用されます。

実行フックの例

にアクセスして "[NetApp Verda GitHubプロジェクト](#)"、Apache CassandraやElasticsearchなどの一般的なアプリケーションの実際の実行フックをダウンロードします。また、独自のカスタム実行フックを構築するための例やアイデアを得ることもできます。

実行フックの作成

Trident Protect を使用して、アプリのカスタム実行フックを作成できます。実行フックを作成するには、所有者、管理者、またはメンバーの権限が必要です。

CRの使用

手順

1. カスタムリソース (CR) ファイルを作成し、という名前を付け `trident-protect-hook.yaml` ます。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
 - `* metadata.name*:` (*required*) このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
 - `* spec.applicationRef *:`(*required*)実行フックを実行するアプリケーションのKubernetes名。
 - `* spec.stage *:`(*required*)実行フックが実行されるアクションのステージを示す文字列。有効な値：
 - 前
 - 投稿
 - `* spec.action *:`(*required*)指定された実行フックフィルタが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値：
 - Snapshot
 - バックアップ
 - リストア
 - フェイルオーバー
 - `* spec.enabled *:`(*Optional*)この実行フックが有効か無効かを示します。指定しない場合、デフォルト値はtrueです。
 - `spec.hookSource:`(*required*) base64でエンコードされたフックスクリプトを含む文字列。
 - `* spec.timeout *:`(*Optional*)実行フックの実行を許可する時間を分単位で定義する数値。最小値は1分で、指定しない場合のデフォルト値は25分です。
 - `* spec.arguments *:`(*Optional*)実行フックに指定できる引数のYAMLリスト。
 - `* spec.matchingCriteria *:`(*Optional*)実行フックフィルタを構成する各ペアの基準キー値ペアのオプションリスト。実行フックごとに最大10個のフィルタを追加できます。
 - `* spec.matchingCriteria.type *:`(*Optional*)実行フックフィルタタイプを識別する文字列。有効な値：
 - コンテナイメージ
 - コンテナ名
 - ポッド名
 - PodLabel
 - ネームスペース名
 - `* spec.matchingCriteria.value *:`(*Optional*)実行フックフィルタ値を識別する文字列または正規表現。

YAMLの例：

```
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: IyEvYmluL2Jhc2gKZWNoYAiZXhhbXBsZSBzY3JpcHQiCg==
  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. CR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-hook.yaml
```

CLI を使用します

手順

1. 実行フックを作成し、括弧内の値を環境からの情報に置き換えます。例：

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

Trident Protectをアンインストールする

試用版から製品の完全版にアップグレードする場合は、Trident Protect コンポーネントを削除する必要がある場合があります。

Trident Protect を削除するには、次の手順を実行します。

手順

1. Trident Protect CR ファイルを削除します。

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

2. Trident Protectを削除します。

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

3. Trident Protect 名前空間を削除します。

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

著作権に関する情報

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および/または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。