# Trident Protectでアプリケーションを保護

Trident

NetApp
October 29, 2026

This PDF was generated from https://docs.netapp.com/ja-jp/trident-2606/trident-protect/learn-about-trident-protect.html on October 29, 2026. Always check docs.netapp.com for the latest.

# 目次


Trident Protectでアプリケーションを保護 1
- Trident Protectについて学ぶ 1
  - 次の手順 1
- Trident Protectのインストール 1
  - Trident Protect の要件 1
  - Trident Protectのインストールと設定 5
  - Trident Protect CLI プラグインをインストールします 8
  - Trident Protectのインストールをカスタマイズする 11
- Trident Protectの管理 16
  - Trident Protectの認証とアクセス制御を管理する 16
  - Trident Protectリソースを監視 22
  - Trident Protect サポートバンドルを生成する 26
  - Trident Protectのアップグレード 28
- アプリケーションの管理と保護 30
  - Trident Protect AppVaultオブジェクトを使用してバケットを管理する 30
  - Trident Protectで管理用のアプリケーションを定義する 41
  - Trident Protectを使用してアプリケーションを保護 46
  - アプリケーションをリストアする 56
  - NetApp SnapMirrorとTrident Protectを使用したアプリケーションのレプリケート 78
  - Trident Protectを使用してアプリケーションを移行する 91
  - KubeVirt 仮想マシンを Trident Protect で保護する 95
  - Trident Protect 実行フックを管理する 107
- Trident Protectをアンインストールする 114

# Trident Protectでアプリケーションを保護

## Trident Protectについて学ぶ

NetApp Trident Protect は、NetApp ONTAP ストレージシステムおよび 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な Kubernetes アプリケーションの機能性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、その要件について確認できます：

- "Trident Protect の要件"

## Trident Protectのインストール

### Trident Protect の要件

まず、運用環境、アプリケーション クラスタ、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を導入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 サービス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
- SUSE Harvester 1.7.0（ONTAP iSCSI）
- SUSE Rancher
- VMware Tanzu ポートフォリオ
- アップストリーム Kubernetes

- Trident Protect バックアップは Linux コンピューティング ノードでのみサポートされます。Windows コンピューティング ノードはバックアップ操作ではサポートされていませ

ん。

- Trident Protect をインストールするクラスタが、実行中のスナップショット コントローラと関連する CRD で構成されていることを確認してください。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、 "これらの手順"を参照してください。
- 少なくとも1つのVolumeSnapshotClassが存在することを確認してください。詳細については、"VolumeSnapshotClass"を参照してください。
- Trident Protect をインストールするには Helm 4.x 以降が必要です。

**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：

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP ストレージアレイ
- Google Cloud NetApp Volumes
- 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たNetAppストレージが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）。
- NetApp ONTAP ストレージバックエンドがあることを確認してください。
- バックアップを保存するためのオブジェクト ストレージ バケットが設定されていることを確認します。
- 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 ではこれらの名前空間は作成されません。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-economy**ボリュームの要件

Trident Protect は、nas-economy ボリュームへのバックアップおよびリストア処理をサポートします。スナップショット、クローン、および SnapMirror レプリケーションの nas-economy ボリュームへの実行は、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 ボリュームについて、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、スナップショット 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スナップショット 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
```

スナップショットディレクトリを有効にするには、各nas-economyボリュームに対して次のコマンドを実行し、 `<volume-UUID>`を変更したいボリュームのUUIDに置き換えます：

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

新しいボリュームのスナップショット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 `snapshotDir`を `true`に設定します。既存のボリュームは影響を受けません。

**KubeVirt VM** でデータを保護

Trident Protectは、データ保護操作中にKubeVirt仮想マシンのファイルシステムのフリーズとアンフリーズ機能を提供し、データの整合性を確保します。VMフリーズ操作の設定方法とデフォルトの動作はTrident Protectのバージョンによって異なり、新しいリリースではHelmチャートパラメータを通じて簡素化された設定が提供されます。

復元操作中は、仮想マシン（VM）用に作成された `VirtualMachineSnapshots`は復元されません。

▼ **Trident Protect 25.10**以降

Trident Protectは、データ保護処理中に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て、整合性を確保します。Trident Protect 25.10以降では、Helmチャートのインストール時に `vm.freeze`パラメータを使用してこの動作を無効にできます。このパラメータはデフォルトで有効になっています。

```
helm install ... --set vm.freeze=false ...
```

▼ **Trident Protect 24.10.1～25.06**

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
```

▼ **Trident Protect 24.10**

Trident Protect 24.10は、データ保護処理中にKubeVirt VMファイルシステムの整合性のある状態を自動的に保証するものではありません。KubeVirt VMデータをTrident Protect 24.10を使用して保護する場合は、データ保護処理の前に、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整合性のある状態になることが保証されます。

Trident Protect 24.10を設定して、"仮想化の設定"を使用し、次のコマンドを実行することで、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できます：

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

Trident Protect を使用してアプリケーションを定義し、VM をリストアする方法については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

### SnapMirror レプリケーションの要件

NetApp SnapMirrorレプリケーションは、以下のONTAPソリューションでTrident Protectと併用できます：

- オンプレミスのNetApp FAS、AFF、ASAシステム。Trident保護を使用したSnapMirrorレプリケーションは、現在ASA r2システムではサポートされていません。
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

#### SnapMirror レプリケーションのための ONTAP クラスタ要件

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てください：

- **NetApp Trident**：NetApp Tridentは、ONTAPをバックエンドとして利用するソースKubernetesクラスタとデスティネーションKubernetesクラスタの両方に存在する必要があります。Trident Protectは、次のドライバでサポートされるストレージクラスを使用して、NetApp SnapMirrorテクノロジによるレプリケーションをサポートします：
  - `ontap-nas`：NFS
  - `ontap-san`：iSCSI
  - `ontap-san`：FC
  - `ontap-san`：NVMe/TCP（ONTAP バージョン 9.15.1 以降が必要）
- ライセンス：ONTAP SnapMirror データ保護バンドルを使用する非同期ライセンスは、ソースとデスティネーション クラスタの両方で有効にする必要があります。詳細については、 "ONTAP における SnapMirror ライセンスの概要"を参照してください。

  ONTAP 9.10.1以降では、すべてのライセンスがNetAppライセンス ファイル（NLF）として提供されます。これは、複数の機能を有効にする単一のファイルです。詳細については、"ONTAP Oneに含まれるライセンス"を参照してください。

  SnapMirror非同期保護のみがサポートされています。

#### 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予定の場合は、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 バックエンドはピアリングする必要があります。詳細については、 "クラスタとSVMのピアリングの概要"を参照してください。

  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。

- **NetApp Trident**および**SVM**：ピアリングされたリモートSVMは、デスティネーション クラスタ上のNetApp Tridentで使用できる必要があります。
- 管理されたバックエンド：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して管理する必要があります。

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

Trident Protect では、ソース クラスタとデスティネーション 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 つ設定する必要があります。ソース クラスタとデスティネーション クラスタが同じ場合、復元力を最大限に高めるには、デスティネーション 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。

#### **SnapMirror** レプリケーションの **Kubernetes** クラスタの要件

Kubernetes クラスタが次の要件を満たしていることを確認します。

- **AppVault** アクセシビリティ：ソース クラスタとデスティネーション クラスタの両方が、アプリケーション オブジェクトのレプリケーションのために AppVault への読み取りと書き込みを行うためのネットワーク アクセスを持っている必要があります。
- ネットワーク接続：ファイアウォール ルール、バケット権限、IP 許可リストを設定して、両方のクラスタと AppVault 間の WAN 経由の通信を有効にします。

> 多くの企業環境では、WAN 接続全体に厳格なファイアウォール ポリシーが実装されています。レプリケーションを設定する前に、これらのネットワーク要件をインフラストラクチャ チームに確認してください。

## **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できます。クラスタ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。

#### **Trident Protect**のインストール

**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. Helmを使用してTrident Protectをインストールします。 `<name-of-cluster>`をクラスタ名に置き換えます。このクラスタ名はクラスタに割り当てられ、クラスタのバックアップとSnapshotを識別するために使用されます：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2606.0 --create
-namespace --namespace trident-protect
```

3. オプションで、デバッグ ログを有効にするには（トラブルシューティングに推奨）、次のコマンドを使用します：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2606.0 --create-namespace --namespace trident-protect
```

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 サポートが問題のトラブルシューティングを行うのに役立ちます。

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：

```
docker.io/netapp/controller:26.06.0
docker.io/netapp/restic:26.06.0
docker.io/netapp/kopia:26.06.0
docker.io/netapp/kopiablockrestore:26.06.0
docker.io/netapp/trident-autosupport:26.06.0
docker.io/netapp/exechook:26.06.0
docker.io/netapp/resourcebackup:26.06.0
docker.io/netapp/resourcerestore:26.06.0
docker.io/netapp/resourcedelete:26.06.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

次に例を示します。

```
docker pull docker.io/netapp/controller:26.06.0
```

```
docker tag docker.io/netapp/controller:26.06.0 <private-registry-
url>/controller:26.06.0
```

```
docker push <private-registry-url>/controller:26.06.0
```

Helmチャートを取得するには、まず `helm pull trident-protect --version 100.2606.0 --repo https://netapp.github.io/trident-protect-helm-chart`を使用してインターネットにアクセスできるマシンにHelmチャートをダウンロードし、その結果の `trident-protect-100.2606.0.tgz`ファイルをオフライン環境にコピーして、最後のステップでリポジトリ参照の代わりに `helm install trident-protect ./trident-protect-100.2606.0.tgz`を使用してインストールします。

2. Trident Protect システムネームスペースを作成します。

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

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

`imageRegistry`および `imagePullSecrets`の値は、 `resourcebackup`および `resourcerestore`を含むすべてのコンポーネントイメージに適用されます。レジストリ内の特定のリポジトリパスにイメージをプッシュする場合（例： `example.com:443/my-repo`）、レジストリフィールドに完全パスを含めてくださ

い。これにより、すべてのイメージが `<private-registry-url>/<image-name>:<tag>` から取得されます。

7. Helmを使用してTrident Protectをインストールします。 `<name_of_cluster>`をクラスタ名に置き換えます。このクラスタ名はクラスタに割り当てられ、クラスタのバックアップとSnapshotを識別するために使用されます：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2606.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

8. オプションで、デバッグ ログを有効にするには（トラブルシューティングに推奨）、次のコマンドを使用します：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2606.0 --create-namespace --namespace trident-protect -f
protectValues.yaml
```

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 サポートが問題のトラブルシューティングを行うのに役立ちます。

AutoSupport設定や名前空間フィルタリングなどの追加のHelmチャート設定オプションについては、"Trident Protectのインストールをカスタマイズする"を参照してください。

## Trident Protect CLI プラグインをインストールします

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 `tridentctl` ユーティリティの拡張機能であり、Trident Protect カスタム リソース（CR）の作成と操作に使用できます。

### Trident Protect CLI プラグインをインストールします

コマンドライン ユーティリティを使用する前に、クラスターにアクセスするために使用するマシンにそれをインストールする必要があります。マシンが x64 と ARM のどちらの CPU を使用しているかに応じて、次の手順に従ってください。

#### Linux AMD64 CPU用プラグインをダウンロード

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/v26.06.0/tridentctl-protect-linux-amd64
```

**Linux ARM64 CPU**用プラグインをダウンロード

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/v26.06.0/tridentctl-protect-linux-arm64
```

**Mac AMD64 CPU**用プラグインをダウンロード

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/v26.06.0/tridentctl-protect-macos-amd64
```

**Mac ARM64 CPU**用プラグインをダウンロード

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/v26.06.0/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. プラグインのバイナリを PATH 変数で定義されている場所にコピーします。例えば、 `/usr/bin`または `/usr/local/bin`（昇格した権限が必要になる場合があります）：

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

3. オプションで、プラグインのバイナリをホームディレクトリ内の場所にコピーすることもできます。この場合、場所が 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、 `tridentctl-protect`または `tridentctl protect`と

> 入力することで、どこからでもプラグインを使用できるようになります。

**Trident CLI** プラグインのヘルプを表示

プラグインの機能に関する詳細なヘルプを取得するには、組み込みのプラグイン ヘルプ機能を使用できます：

手順

1. ヘルプ機能を使用して使用方法の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ら、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. 完了スクリプトを作成します：

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. スクリプトを格納するための新しいディレクトリをホーム ディレクトリに作成します：

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

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

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

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

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

**Z** シェルの自動補完を有効にする

手順

1. 完了スクリプトを作成します：

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. スクリプトを格納するための新しいディレクトリをホーム ディレクトリに作成します：

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

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

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

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

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

結果

次回のシェルログイン時には、tridentctl-protect プラグインによるコマンドの自動補完を使用できます。

# Trident Protectのインストールをカスタマイズする

環境の特定の要件を満たすように Trident Protect のデフォルト設定をカスタマイズできます。

## Trident Protectコンテナのリソース制限を指定する

Trident Protectのインストール後、設定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のリソース量を制御できます。

手順

1. `resourceLimits.yaml`という名前のファイルを作成します。
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おり、各リソース制限のデフォルト値が含まれています：

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

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

セキュリティコンテキスト制約をカスタマイズする

Trident Protect のインストール後、設定ファイルを使用して Trident Protect コンテナの OpenShift セキュリティコンテキスト制約（SCC）を変更できます。これらの制約は、Red Hat OpenShift クラスタ内のポッドのセキュリティ制限を定義します。

手順

1. `sccconfig.yaml`という名前のファイルを作成します。
2. ファイルに SCC オプションを追加し、環境のニーズに応じてパラメータを変更します。

   次の例は、SCC オプションのパラメータのデフォルト値を示しています：

   ```
   scc:
     create: true
     name: trident-protect-job
     priority: 1
   ```

   この表は、SCC オプションのパラメータについて説明しています：

| パラメータ | 概要 | デフォルト |
| --- | --- | --- |
| 作成する | SCC リソースを作成できるかどうかを決定します。SCC リソースは、 `scc.create`が `true`に設定されていて、Helm インストールプロセスがOpenShift環境を識別した場合にのみ作成されます。OpenShiftで動作していない場合、または `scc.create`が `false`に設定されている場合、SCC リソースは作成されません。 | true |
| 名前 | SCCの名前を指定します。 | trident-protect-job |
| 優先度 | SCC の優先度を定義します。優先度の値が高い SCC は、値が低い SCC よりも先に評価されます。 | 1 |

3. `sccconfig.yaml`ファイルから値を適用します：

   ```
   helm upgrade trident-protect -n trident-protect netapp-trident-
   protect/trident-protect -f sccconfig.yaml --reuse-values
   ```

   これにより、デフォルト値が `sccconfig.yaml`ファイルで指定された値に置き換えられます。

追加の**Trident Protect**ヘルムチャート設定を構成する

AutoSupport 設定、名前空間フィルタリング、および OpenShift Webhook 証明書管理をカスタマイズして、特定の要件を満たすことができます。次の表に、使用可能な設定パラメータを示します：

| パラメータ | タイプ | 概要 |
|---|---|---|
| autoSupport.proxy | string | NetApp AutoSupport接続のプロキシURLを設定します。これを使用して、プロキシ サーバ経由でサポートバンドルのアップロードをルーティングします。例：`http://my.proxy.url`。 |
| autoSupport.insecure | ブーリアン | `true`に設定すると、`AutoSupport`プロキシ接続の`TLS`検証をスキップします。安全でないプロキシ接続にのみ使用してください。（デフォルト：`false`） |
| autoSupport.enabled | ブーリアン | Trident Protect AutoSupportバンドルの日次アップロードを有効または無効にします。 `false`に設定すると、スケジュールされた日次アップロードは無効になりますが、サポート バンドルを手動で生成することはできます。（デフォルト： `true`） |
| restoreSkipNamespaceAnnotations | string | バックアップおよびリストア処理から除外するネームスペースアノテーションのカンマ区切りリスト。アノテーションに基づいてネームスペースをフィルタリングできます。 |
| restoreSkipNamespaceLabels | string | バックアップおよび復元操作から除外する名前空間ラベルのコンマ区切りリスト。ラベルに基づいて名前空間をフィルタリングできます。 |
| useOpenshiftServiceCA | ブーリアン | OpenShiftのネイティブ Service CA Operator との統合を有効にし、Trident Protect アドミッション Webhook の TLS 証明書の自動作成と挿入を行います。 `true`に設定すると、`Service CA Operator`が `Webhook TLS` 証明書を自動的に管理します。`OpenShift` クラスターでのみ使用してください。（デフォルト： `false`） |

これらのオプションは、YAML 構成ファイルまたはコマンドライン フラグを使用して設定できます：

**YAML**ファイルを使用する

手順

1. 設定ファイルを作成し、 `values.yaml`という名前を付けます。
2. 作成したファイルに、カスタマイズする設定オプションを追加します。

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
useOpenshiftServiceCA: true
```

3. `values.yaml`ファイルに正しい値を入力したら、構成ファイルを適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

**CLI** フラグを使用する

手順

1. `--set`フラグを指定して次のコマンドを使用し、個々のパラメータを指定します。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set-string
restoreSkipNamespaceAnnotations="{annotation1,annotation2}" \
  --set-string restoreSkipNamespaceLabels="{label1,label2}" \
  --set useOpenshiftServiceCA=true \
  --reuse-values
```

特定のノードへの**Trident Protect**ポッドの制限

Kubernetes の nodeSelector ノード選択制約を使用して、ノードラベルに基づいて Trident Protect ポッドを実行できるノードを制御できます。デフォルトでは、Trident Protect は Linux を実行しているノードに制限されます。ニーズに応じてこれらの制約をさらにカスタマイズできます。

手順

1. `nodeSelectorConfig.yaml`という名前のファイルを作成します。
2. nodeSelectorオプションをファイルに追加し、環境のニーズに応じて制限するノード ラベルを追加または

変更するようにファイルを修正します。たとえば、次のファイルにはデフォルトの OS 制限が含まれていますが、特定の地域とアプリ名もターゲットにしています：

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. `nodeSelectorConfig.yaml`ファイルから値を適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

これにより、デフォルトの制限が、 `nodeSelectorConfig.yaml`ファイルで指定した制限に置き換えられます。

# 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 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。次に例を示します。

```
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。次に例を示します。

```
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。次に例を示します。

```
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。次に例を示します。

```
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。次に例を示します。

**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、26.06 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26.06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 100.2606.0 --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。

# 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として知られています。AppVaultオブジェクトは、ストレージバケットの宣言的な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、リストア操作、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設定が含まれています。管理者のみがAppVaultsを作成できます。

アプリケーションでデータ保護操作を実行するときは、AppVault CRを手動またはコマンドラインから作成する必要があります。AppVault CRは環境に固有のものであり、このページの例をAppVault CRを作成する際のガイドとして使用できます。

AppVault CRがTrident Protectがインストールされているクラスタ上にあることを確認してください。AppVault CRが存在しないか、アクセスできない場合は、コマンドラインにエラーが表示されます。

### AppVault認証とパスワードを設定

AppVault CRを作成する前に、AppVaultと選択したデータムーバーがプロバイダおよび関連リソースで認証できることを確認してください。

#### データムーバーリポジトリのパスワード

CRやTrident Protect CLIプラグインを使用してAppVaultオブジェクトを作成する際、ResticおよびKopia暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。シークレットを指定しない場合、Trident Protectはデフォルトのパスワードを使用します。

- AppVault CR を手動で作成する場合は、**spec.dataMoverPasswordSecretRef** フィールドを使用してシークレットを指定します。
- Trident Protect CLIを使用してAppVaultオブジェクトを作成する場合は、 `--data-mover-password-secret-ref`引数を使用してシークレットを指定します。

データムーバーリポジトリのパスワードシークレットを作成する

次の例を使用して、パスワードシークレットを作成します。AppVaultオブジェクトを作成する際、Trident Protectにこのシークレットを使用してデータムーバーリポジトリで認証するように指示できます。

- 使用しているデータムーバーに応じて、そのデータムーバーに対応するパスワードのみを含める必要があります。たとえば、Resticを使用しており、将来Kopiaを使用する予定がない場合は、シークレットを作成するときにResticパスワードのみを含めることができます。
- パスワードは安全な場所に保管してください。同じクラスタまたは別のクラスタでデータをリストアするには、これが必要になります。クラスタまたは `trident-protect`ネームスペースが削除されると、パスワードなしでバックアップやSnapshotをリストアできなくなります。

**CR**を使用する

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

**CLI** を使用する

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

**S3** 互換ストレージの **IAM** 権限

Trident Protectを使用してAmazon S3、Generic S3、 "StorageGrid S3"、または "ONTAP S3"などのS3互換ストレージにアクセスする場合は、指定するユーザクレデンシャルにバケットへのアクセスに必要な権限があることを確認する必要があります。次に、Trident Protectでアクセスするために必要な最小限の権限を付与するポリシーの例を示します。このポリシーは、S3互換バケットポリシーを管理するユーザに適用できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
        "Action": [
          "s3:PutObject",
          "s3:GetObject",
          "s3:ListBucket",
          "s3:DeleteObject"
        ],
        "Resource": "*"
      }
    ]
}
```

Amazon S3 ポリシーの詳細については、 "Amazon S3 ドキュメント"の例を参照してください。

#### Amazon S3（AWS）認証用のEKS Pod Identity

Trident Protect は、Kopia データ ムーバー操作用の EKS Pod Identity をサポートします。この機能により、Kubernetes シークレットに AWS 認証情報を保存せずに、S3 バケットへの安全なアクセスが可能になります。

**Trident Protect** を使用した **EKS Pod Identity** の要件

EKS Pod Identity を Trident Protect で使用する前に、次の点を確認してください：

- EKS クラスタで Pod Identity が有効になっています。
- 必要な S3 バケット権限を持つ IAM ロールを作成しました。詳細については、"S3 互換ストレージの IAM 権限"を参照してください。
- IAM ロールは次の Trident Protect サービス アカウントに関連付けられています：
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

Pod Identity を有効にして IAM ロールをサービスアカウントに関連付ける詳細な手順については、 "AWS EKS Pod Identity のドキュメント"を参照してください。

**AppVault**設定 EKS Pod Identityを使用する場合は、明示的なクレデンシャルの代わりに `useIAM: true`フラグを使用してAppVault CRを設定します：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: eks-protect-vault
  namespace: trident-protect
spec:
  providerType: AWS
```

```
  providerConfig:
    s3:
      bucketName: trident-protect-aws
      endpoint: s3.example.com
      useIAM: true
```

#### AppVault クラウド プロバイダ向けのキー生成例

AppVault CR を定義する際には、IAM 認証を使用していない限り、プロバイダによってホストされているリソースにアクセスするための認証情報を含める必要があります。認証情報のキーを生成する方法は、プロバイダによって異なります。以下は、いくつかのプロバイダのコマンドライン キー生成の例です。次の例を使用して、各クラウド プロバイダの認証情報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=<amazon-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=<ontap-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=<storagegrid-s3-trident-protect-src
-bucket-secret> \
-n trident-protect
```

### AppVault 作成例

以下は、各プロバイダーの AppVault 定義の例です。

#### AppVault CR の例

以下のCRの例を使用して、各クラウド プロバイダのAppVaultオブジェクトを作成できます。

- オプションで、Restic および Kopia リポジトリの暗号化用のカスタム パスワードを含む Kubernetes シークレットを指定できます。詳細については、[データムーバーリポジトリのパスワード]を参照してください。
- Amazon S3（AWS）AppVaultオブジェクトの場合、オプションでsessionTokenを指定できます。これは、認証にシングル サインオン（SSO）を使用している場合に便利です。このトークンは、AppVault クラウド プロバイダ向けのキー生成例でプロバイダのキーを生成するときに作成されます。
- S3 AppVaultオブジェクトの場合、 `spec.providerConfig.S3.proxyURL`キーを使用して、アウトバウンドS3トラフィックの出力プロキシURLをオプションで指定できます。

**Google Cloud**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  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
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3-secret
```

Kopia データムーバーで Pod Identity を使用している EKS 環境の場合、`providerCredentials` セクションを削除し、代わりに `s3` 設定の下に `useIAM: true` を追加できます。

**Microsoft Azure**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
```

```
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  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
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

**ONTAP S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
```

```
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

**StorageGrid S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

#### 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、各プロバイダーの AppVault CR を作成できます。

- オプションで、Restic および Kopia リポジトリの暗号化用のカスタム パスワードを含む Kubernetes シークレットを指定できます。詳細については、[データムーバーリポジトリのパスワード]を参照してください。
- S3 AppVaultオブジェクトの場合、 `--proxy-url <ip_address:port>`引数を使用して、アウトバウンドS3トラフィックの出力プロキシURLをオプションで指定できます。

**Google Cloud**

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

**Amazon S3（AWS）**

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

**Microsoft Azure**

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

**汎用 S3**

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
```

```
-n trident-protect
```

**ONTAP S3**

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

**StorageGrid S3**

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

サポートされている `**providerConfig.s3**`設定オプション

S3 プロバイダの構成オプションについては、次の表を参照してください：

| パラメータ | 概要 | デフォルト | 例 |
|---|---|---|---|
| `providerConfig.s3.skipCertValidation` | SSL/TLS 証明書の検証を無効にします。 | false | "true"、"false" |
| `providerConfig.s3.secure` | S3 エンドポイントとの安全な HTTPS 通信を有効にします。 | true | "true"、"false" |
| `providerConfig.s3.proxyURL` | S3 に接続するために使用されるプロキシ サーバの URL を指定します。 | なし | http://proxy.example.com:8080 |
| `providerConfig.s3.rootCA` | SSL/TLS 検証用のカスタム ルート CA 証明書を提供します。 | なし | "CN=MyCustomCA" |
| `providerConfig.s3.useIAM` | S3 バケットにアクセスするための IAM 認証を有効にします。EKS Pod Identity に適用可能です。 | false | true、false |

### AppVault 情報を表示します

Trident Protect CLI プラグインを使用して、クラスタ上に作成した 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 \
-n trident-protect
```

出力例：

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

2. オプションで、各リソースのAppVaultPathを表示するには、フラグ `--show-paths`を使用します。

表の最初の列のクラスタ名は、Trident Protect Helm のインストール時にクラスタ名が指定された場合にのみ使用できます。例： `--set clusterName=production1`

#### AppVault を削除します

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 内の `finalizers`キーを削除しないでください。削除すると、AppVault バケット内に残留データが残り、クラスター内にリソースが孤立する可能性があります。

開始する前に

削除するAppVaultで使用されているすべてのスナップショットとバックアップCRが削除されていることを確認してください。

**Kubernetes CLI** を使用して **AppVault** を削除する

1. AppVaultオブジェクトを削除します。 `appvault-name`を削除するAppVaultオブジェクトの名前に置き換えます：

```
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ます。 `appvault-name`を削除するAppVaultオブジェクトの名前に置き換えます：

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

## Trident Protectで管理用のアプリケーションを定義する

Trident Protectで管理するアプリケーションを定義するには、アプリケーションCRと関連するAppVault CRを作成します。

#### AppVault CR を作成します

アプリケーションでデータ保護処理を実行する際に使用されるAppVault CRを作成する必要があります。AppVault CRは、Trident Protectがインストールされているクラスタに配置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、"AppVault カスタム リソース。"を参照してください。

アプリケーションを定義する

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。管理対象アプリケーションを定義するには、アプリケーションCRを手動で作成するか、Trident Protect CLIを使用します。

> 名前空間全体ではなく名前で特定のKubeVirt仮想マシン（VM）を保護するVMスコープのアプリケーションを定義するには、Application CRの `spec.includedVirtualMachines`フィールドま

たはCLIの `--virtual-machines`フラグを使用します。詳細、フィールドリファレンス、およびYAMLの例については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

**CR**を使用してアプリケーションを追加する

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、名前を付けます（例： `maria-app.yaml`）。
   b. 次の属性を設定します：
      - **metadata.name**：（必須）アプリケーションのカスタム リソースの名前。保護操作に必要な他の CR ファイルはこの値を参照するため、選択した名前を書き留めておいてください。
      - **spec.includedNamespaces**：（必須）ネームスペースとラベルセレクタを使用して、アプリケーションが使用するネームスペースとリソースを指定します。アプリケーションネームスペースはこのリストに含まれている必要があります。ラベルセレクタはオプションであり、指定された各ネームスペース内のリソースをフィルタリングするために使用できます。
      - **spec.includedClusterScopedResources**：（オプション）この属性を使用して、アプリケーション定義に含めるクラスタスコープのリソースを指定します。この属性を使用すると、グループ、バージョン、種類、ラベルに基づいてこれらのリソースを選択できます。
        - **groupVersionKind**：（必須）クラスタースコープのリソースの API グループ、バージョン、および種類を指定します。
        - **labelSelector**：（オプション）ラベルに基づいてクラスタースコープのリソースをフィルタリングします。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**：（オプション）このアノテーションは、KubeVirt環境など、スナップショットの前にファイルシステムのフリーズが発生する仮想マシンから定義されたアプリケーションにのみ適用されます。スナップショット作成中にこのアプリケーションがファイルシステムに書き込みできるかどうかを指定します。trueに設定すると、アプリケーションはグローバル設定を無視し、スナップショット作成中にファイルシステムへの書き込みが可能になります。falseに設定すると、アプリケーションはグローバル設定を無視し、スナップショット作成中はファイルシステムがフリーズします。指定されていても、アプリケーション定義に仮想マシンが含まれていない場合、このアノテーションは無視されます。指定されていない場合、アプリケーションは"グローバルTrident Protect freeze設定"に従います。

アプリケーションがすでに作成された後にこのアノテーションを適用する必要がある場合は、次のコマンドを使用できます：

```
kubectl annotate application -n <application CR namespace> <application CR name> protect.trident.netapp.io/skip-vm-freeze="true"
```

+
YAMLの例：

+

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. （オプション）必要に応じて、同じCRにリソースフィルタリングを追加して、特定のリソースを含めるか除外することができます：

   - 一般的なフィルタの例：

     - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須）`Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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演算で一致し、各要素内のフィールド（group、kind、version）はAND演算で一致します。
       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
       - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
       - **resourceMatchers[].names**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前。
       - **resourceMatchers[].namespaces**：（オプション）フィルタリングするリソースの

Kubernetes metadata.name フィールド内の名前空間。

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

> ℹ `resourceFilter`と `labelSelector`の両方が使用される場合、`resourceFilter`が最初に実行され、次に `labelSelector`が結果のリソースに適用されます。

次に例を示します。

```
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"]
```

- **PVC**のみのフィルターの例：

  PVCのみのアプリケーションを定義するには、 `PersistentVolume`と `VolumeSnapshotClass`もリソースフィルターに含める必要があります。スナップショットとバックアップの操作は、 `PersistentVolume`（各PVCにバインドされたクラスタースコープのボリューム）および `VolumeSnapshotClass`（スナップショットドライバ）に依存しており、これらがないと失敗します。次に例を示します。

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

```
    - kind: PersistentVolumeClaim
      version: v1
    - kind: PersistentVolume
      version: v1
    - kind: VolumeSnapshotClass
      version: v1
  resourceSelectionCriteria: Include
```

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。次に例を示します。

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

手順

1. 次のいずれかの例を使用してアプリケーション定義を作成し、適用します。括弧内の値は、ご使用の環境の情報に置き換えてください。例に示す引数を含むコンマ区切りリストを使用して、アプリケーション定義に名前空間とリソースを含めることができます。

アプリを作成するときにオプションで注釈を使用して、スナップショット中にアプリケーションがファイルシステムに書き込むことができるかどうかを指定できます。これは、KubeVirt環境など、スナップショットの前にファイルシステムのフリーズが発生する仮想マシンから定義されたアプリケーションにのみ適用されます。注釈を `true`に設定すると、アプリケーションはグローバル設定を無視し、スナップショット中にファイルシステムに書き込むことができます。 `false`に設定すると、アプリケーションはグローバル設定を無視し、スナップショット中にファイルシステムがフリーズされます。アノテーションを使用しても、アプリケーション定義にアプリケーションの仮想マシンがない場合、アノテーションは無視されます。アノテーションを使用しない場合、アプリケーションは"グローバルTrident Protect freeze設定"に従います。

+ CLI を使用してアプリケーションを作成する際に注釈を指定するには、 `--annotation` フラグを使用できます。

+ * アプリケーションを作成し、ファイルシステムのフリーズ動作に関するグローバル設定を使用します。

+

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

+ * アプリケーションを作成し、ファイルシステムのフリーズ動作に関するローカルアプリケーション設定を構成します：

+

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

```
protect.trident.netapp.io/skip-vm-freeze=<"true"|"false">
```

+ * `--resource-filter-include`および `--resource-filter-exclude`フラグを使用して、次の例に示すように、グループ、種類、バージョン、ラベル、名前、名前空間などの `resourceSelectionCriteria`に基づいてリソースを含めたり除外したりすることができます。

+

```
tridentctl-protect create application <my_new_app_cr_name> --namespaces
<namespaces_to_include> --csr <cluster_scoped_resources_to_include>
--namespace <my-app-namespace> --resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my-
deployment"],"Namespaces":["my-namespace"],"LabelSelectors":["app=my-
app"]}]'
```

+ * PVCのみのアプリケーションを定義するには、リソースフィルターに `PersistentVolume`および `VolumeSnapshotClass``も含める必要があります。スナップショットとバックアップの操作は、`PersistentVolume`（各PVCにバインドされたクラスタースコープのボリューム）および `VolumeSnapshotClass`（スナップショットドライバ）に依存しており、これらがないと失敗します。次に例を示します。

+

```
tridentctl-protect create app my-pvc-app --namespaces <my-app-
namespace> --resource-filter-include
'[{"Kind":"PersistentVolumeClaim","Version":"v1"},{"Kind":"PersistentVo
lume","Version":"v1"},{"Kind":"VolumeSnapshotClass","Version":"v1"}]'
-n <my-app-namespace>
```

## Trident Protectを使用してアプリケーションを保護

Trident Protectで管理されているすべてのアプリを、自動化された保護ポリシーまたはアドホックベースでスナップショットとバックアップを作成して保護できます。

Trident Protectを設定して、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ことができます。"Trident Protectを使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください"。

### オンデマンドスナップショットを作成する

オンデマンド スナップショットはいつでも作成できます。

クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、ま

たはクローンに含まれます。

**CR** を使用してスナップショットを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.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**：（必須）スナップショットの内容（メタデータ）を保存するAppVaultの名前。
   - **spec.reclaimPolicy**：（オプション）スナップショット CR が削除されたときに、スナップショットの AppArchive に対して何が起こるかを定義します。つまり、 `Retain`に設定されている場合でも、スナップショットは削除されます。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. `trident-protect-snapshot-cr.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-snapshot-cr.yaml
```

**CLI** を使用してスナップショットを作成する

手順

1. 括弧内の値を環境の情報に置き換えてスナップショットを作成します。次に例を示します。

```
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 バックアップを作成する

アプリはいつでもバックアップできます。

クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 バックアップ処理に十分であることを確認します。バックアップ処理中にトークンの有効期限が切れると、処理が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

**CR** を使用してバックアップを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.applicationRef**：（必須）バックアップするアプリケーションの Kubernetes 名。
   - **spec.appVaultRef**：（必須）バックアップコンテンツを保存するAppVaultの名前。
   - **spec.dataMover**：（オプション）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。可能な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia`（デフォルト）
   - **spec.reclaimPolicy**：（オプション）クレームから解放されたときにバックアップに何が起こるかを定義します。有効な値は次のとおりです。
     - `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スナップショットの名前。指定しない場合は、一時ボリュームが作成され、バックアップされます。

   YAMLの例：

```
---
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. `trident-protect-backup-cr.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-backup-cr.yaml
```

**CLI**を使用してバックアップを作成する

手順

1. 括弧内の値を環境の情報に置き換えてバックアップを作成します。次に例を示します。

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

オプションで `--full-backup`フラグを使用して、バックアップを非増分にするかどうかを指定できます。デフォルトでは、すべてのバックアップは増分バックアップになります。このフラグを使用すると、バックアップは非増分になります。ベストプラクティスとしては、定期的にフル バックアップを実行し、その間に増分バックアップを実行して、リストアに伴うリスクを最小限に抑えます。

サポートされているバックアップアノテーション

次の表は、バックアップ CR を作成するときに使用できる注釈を示しています：

| 注釈 | タイプ | 概要 | デフォルト値 |
|---|---|---|---|
| protect.trident.netapp.io/full-backup | string | バックアップを非増分にするかどうかを指定します。 `true`に設定すると、非増分バックアップが作成されます。ベストプラクティスとしては、定期的にフル バックアップを実行し、その間に増分バックアップを実行して、リストアに伴うリスクを最小限に抑えます。 | "false" |
| protect.trident.netapp.io/snapshot-completion-timeout | string | スナップショット処理全体が完了するまでに許容される最大時間。 | 「60m」 |
| protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout | string | ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。 | "30分" |
| protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト | string | ボリューム スナップショットの作成に許可される最大時間。 | 「5m」 |

| 注釈 | タイプ | 概要 | デフォルト値 |
| --- | --- | --- | --- |
| protect.trident.netapp.io/pvc-bind-timeout-sec | string | 新しく作成されたPersistentVolumeClaims（PVC）が `Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。 | 「1200」（20分） |

データ保護スケジュールを作成する

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してアプリを保護します。スナップショットとバックアップを時間ごと、日ごと、週ごと、月ごとに作成するように選択でき、保持するコピーの数を指定できます。full-backup-rule アノテーションを使用して、非増分フル バックアップをスケジュールできます。デフォルトでは、すべてのバックアップは増分バックアップになります。フル バックアップを定期的に実行し、その間に増分バックアップを実行することで、復元に伴うリスクを軽減できます。

- スナップショットのみのスケジュールを作成するには、 `backupRetention`をゼロに設定し、 `snapshotRetention`をゼロより大きい値に設定します。 `snapshotRetention`をゼロに設定すると、スケジュールされたバックアップでは引き続きスナップショットが作成されますが、これらは一時ボリュームであり、バックアップが完了するとすぐに削除されます。
- クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

**CR**を使用してスケジュールを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-schedule-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.dataMover**：（オプション）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。可能な値（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**：（必須）バックアップコンテンツを保存するAppVaultの名前。
   - **spec.backupRetention**：（必須）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（スナップショットのみ）。
   - **spec.backupReclaimPolicy**：（オプション）保持期間中にバックアップ CR が削除された場合に、バックアップに何が起こるかを決定します。保持期間が過ぎると、バックアップは常に削除されます。可能な値（大文字と小文字が区別されます）：
     - `Retain`（デフォルト）

- `Delete`

- **spec.snapshotRetention**：（必須）保持するスナップショットの数。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.snapshotReclaimPolicy**：（オプション）保持期間中にスナップショット CR が削除された場合にスナップショットに何が起こるかを決定します。保持期間が過ぎると、スナップショットは常に削除されます。可能な値（大文字と小文字が区別されます）：
  - `Retain`
  - `Delete`（デフォルト）
- **spec.granularity**：スケジュールを実行する頻度。可能な値と、必須の関連フィールド：
  - `Hourly`（`spec.minute`を指定する必要があります）
  - `Daily`（`spec.minute`と`spec.hour`を指定する必要があります）
  - `Weekly`（`spec.minute`、`spec.hour`、`spec.dayOfWeek`を指定する必要があります）
  - `Monthly`（`spec.minute`、`spec.hour`、`spec.dayOfMonth`を指定する必要があります）
  - `Custom`
- **spec.dayOfMonth**：（オプション）スケジュールを実行する月の日付（1 - 31）。粒度が`Monthly`に設定されている場合、このフィールドは必須です。値は文字列として指定する必要があります。
- **spec.dayOfWeek**：（オプション）スケジュールを実行する曜日（0 - 7）。値 0 または 7 は日曜日を示します。粒度が`Weekly`に設定されている場合、このフィールドは必須です。値は文字列として指定する必要があります。
- **spec.hour**：（オプション）スケジュールを実行する時刻（0 - 23）。粒度が`Daily`、`Weekly`、または`Monthly`に設定されている場合、このフィールドは必須です。値は文字列として指定する必要があります。
- **spec.minute**：（オプション）スケジュールを実行する時刻の分（0～59）。このフィールドは、粒度が`Hourly`、`Daily`、`Weekly`、または`Monthly`に設定されている場合は必須です。値は文字列として指定する必要があります。
- **spec.runImmediately**:（オプション）スケジュールが作成されたときに、一度限りの即時ベースライン実行（保持設定に基づくバックアップおよび/またはスナップショット）をトリガーするには、`true`に設定します。デフォルトは`false`。これはその後の繰り返し実行には影響しません。

バックアップとスナップショットのスケジュールの YAML の例：

```
---
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: Daily
  hour: "0"
  minute: "0"
```

スナップショットのみのスケジュールの YAML の例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

即時実行スケジュールのYAML例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-daily-schedule-run-immediately
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "7"
  snapshotRetention: "7"
  granularity: Daily
  hour: "3"
  minute: "0"
  runImmediately: true
```

3. `trident-protect-schedule-cr.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-schedule-cr.yaml
```

**CLI** を使用してスケジュールを作成する

手順

1. 括弧内の値を環境の情報に置き換えて、保護スケジュールを作成します。次に例を示します。

> `tridentctl-protect create schedule --help`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 <my_schedule_name> \
  --appvault <my_appvault_name> \
  --app <name_of_app_to_snapshot> \
  --backup-retention <how_many_backups_to_retain> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_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> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

次のフラグを使用すると、スケジュールをさらに制御できます：

- フル バックアップのスケジュール設定： `--full-backup-rule`フラグを使用して、非増分フル バックアップをスケジュール設定します。このフラグは `--granularity Daily`でのみ機能します。有効な値は次のとおりです。
  - `Always`：毎日フル バックアップを作成します。
  - 特定の曜日：1つ以上の曜日をカンマで区切って指定します（例： `"Monday,Thursday"` ）。有効な値：Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。

> `--full-backup-rule`フラグは、時間単位、週単位、または月単位の粒度では機能しません。

- 即時ベースライン保護: `--run-immediately true`を使用して、最初のスケジュール実行時間を待つのではなく、スケジュールが作成された時点で直ちに初期バックアップまたはスナップショットを作成します。デフォルトは `false`です。
- スナップショットのみのスケジュール： `--backup-retention 0`を設定し、 `--snapshot-retention`に0より大きい値を指定します。

サポートされているスケジュール注釈

次の表は、スケジュール CR を作成するときに使用できる注釈を示しています：

| 注釈 | タイプ | 概要 | デフォルト値 |
|---|---|---|---|
| protect.trident.netapp.io/full-backup-rule | string | フル バックアップをスケジュールするためのルールを指定します。 `Always`に設定して常時フル バックアップを実行するか、要件に応じてカスタマイズできます。たとえば、日単位の粒度を選択した場合、フル バックアップを実行する曜日を指定できます（例：`"Monday,Thursday"`）。有効な曜日の値は、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sundayです。この注釈は、 `granularity`が `Daily`に設定されているスケジュールでのみ使用できます。 | 未設定（すべてのバックアップは増分） |
| protect.trident.netapp.io/snapshot-completion-timeout | string | スナップショット処理全体が完了するまでに許容される最大時間。 | 「60m」 |
| protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout | string | ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。 | "30分" |
| protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト | string | ボリューム スナップショットの作成に許可される最大時間。 | 「5m」 |
| protect.trident.netapp.io/pvc-bind-timeout-sec | string | 新しく作成されたPersistentVolumeClaims（PVC）が `Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。 | 「1200」（20分） |

スナップショットを削除する

不要になったスケジュール済みスナップショットまたはオンデマンド スナップショットを削除します。

手順

1. スナップショットに関連付けられているスナップショット 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する

不要になったスケジュール バックアップまたはオンデマンド バックアップを削除します。

> ⓘ オブジェクト ストレージからすべてのバックアップ データを削除するために、回収ポリシーが `Delete`に設定されていることを確認してください。偶発的なデータ損失を回避するため、ポリシーのデフォルト設定は `Retain`です。ポリシーが `Delete`に変更されない場合、バックアップ データはオブジェクト ストレージに残るため、手動で削除する必要があります。

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
```

バックアップ操作のステータスを確認する

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

手順

1. バックアップ操作のステータスを取得するには、次のコマンドを使用します。括弧内の値は、ご使用の環境の情報に置き換えてください：

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

**azure-netapp-files（ANF）操作のバックアップとリストアを有効にする**

Trident Protectをインストールした場合、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、スペース効率に優れたバックアップおよびリストア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

以下を確認してください。

- 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。
- `azure-netapp-files`がストレージ バックエンドのデフォルトのストレージ クラスとして選択されています。

▼ 設定手順を展開する

1. ANF ボリュームが Trident 24.10 へのアップグレード前に作成された場合は、Trident で以下の手順を実行してください：

   a. azure-netapp-files ベースでアプリケーションに関連付けられている各 PV のスナップショット 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 に対してスナップショット 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ますが、これにより、バックアップ プロセス中に容量プールのスペースが一時的に消費されます。この場合、バックアップされる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、容量プールに十分なスペースが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を使用したバックアップとリストアの準備ができています。各PVCは、バックアップとリストアのために他のアプリケーションでも使用できます。

## アプリケーションをリストアする

**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、アプリケーションに対して設定されているすべての実行フックもアプリとともに復元されます。リストア後の実行フックが存在する場合、リストア処理の一部として自動的に実行されます。
- バックアップから別の名前空間または元の名前空間への復元は、qtree ボリュームでサポートされています。ただし、スナップショットから別の名前空間または元の名前空間への復元は、qtree ボリュームではサポートされていません。
- 詳細設定を使用して復元操作をカスタマイズできます。詳細については、"高度な Trident Protect リストア設定を使用する"を参照してください。
- アーカイブからすべての名前空間リソースを復元せずに特定のKubeVirt仮想マシン（VM）を復元するには、復元CR の `includedVirtualMachines`フィールドを使用します。サポートされているCR、フィールドリファレンス、およびYAMLの例については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

バックアップから別の名前空間に復元する

BackupRestore CR を使用して異なる名前空間にバックアップを復元すると、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設定します。

- 既存のリソースを含む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、バックアップを新しい名前空間にリストアします。

- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみネームスペースを自動的に作成します。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。

**CR**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

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

   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.destinationApplicationName**: （オプション）復元されたアプリケーションの名前。指定した場合、復元されたアプリケーションはこの名前を使用します。指定しない場合、復元されたアプリケーションは元のアプリケーション名を使用します。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     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
    destinationApplicationName: my-new-app-name
    namespaceMapping: [{"source": "my-source-namespace",
  "destination": "my-destination-namespace"}]
```

3. （オプション）アプリケーションの特定のリソースのみを選択して復元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かを指定するフィルタリングを追加します。

> Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。例えば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は、関連付けられたポッドも復元します。

- **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須）`Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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演算で一致し、各要素内のフィールド（group、kind、version）はAND演算で一致します。
    - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
    - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
    - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
    - **resourceMatchers[].names**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前。
    - **resourceMatchers[].namespaces**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前空間。
    - **resourceMatchers[].labelSelectors**：（オプション）"Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`

      次に例を示します。

```
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. `trident-protect-backup-restore-cr.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-backup-restore-cr.yaml
```

**CLI** を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、バックアップを別の名前空間に復元します。 `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デスティネーション名前空間にマッピングします（形式は `source1:dest1,source2:dest2`）。次に例を示します。

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

バックアップから元の名前空間に復元する

バックアップはいつでも元の名前空間に復元できます。In Placeリストアを実行すると、Trident Protectは、無効なリカバリポイントを防止するために、保護スケジュールと進行中の操作を自動的に管理します。

- 復元が開始される前に、アプリケーションに対して有効になっているすべての保護スケジュールが無効になります。これにより、アプリケーションリソースの復元中に、スケジュールされたバックアップやスナップショットが実行されるのを防ぎます。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。既に無効化されているスケジュールは、引き続き無効化されたままです。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元処理は続行され、復元CRステータスに警告が記録されます。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。

- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

> Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。

**CR**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-ipr-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

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

   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。

     次に例を示します。

```
---
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）アプリケーションの特定のリソースのみを選択して復元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かを指定するフィルタリングを追加します。

   > Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。例えば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は、関連付けられたポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または

`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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演算で一致し、各要素内のフィールド（group、kind、version）はAND演算で一致します。
  - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
  - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
  - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
  - **resourceMatchers[].names**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前。
  - **resourceMatchers[].namespaces**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前空間。
  - **resourceMatchers[].labelSelectors**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`

    次に例を示します。

```
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. `trident-protect-backup-ipr-cr.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-backup-ipr-cr.yaml
```

**CLI** を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、バックアップを元の名前空間に復元します。 `backup`引数

は、名前空間とバックアップ名を `<namespace>/<name>`の形式で使用します。次に例を示します。

```
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にリストアする

元のクラスタに問題がある場合は、バックアップを別のクラスタにリストアできます。

- Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみネームスペースを自動的に作成します。

開始する前に

次の前提条件が満たされていることを確認してください：

- デスティネーション クラスタに Trident Protect がインストールされている。
- デスティネーション クラスタは、バックアップが保存されているソース クラスタと同じAppVaultのバケット パスにアクセスできます。
- ローカル環境が、 `tridentctl-protect get appvaultcontent`コマンドの実行時にAppVault CRで定義されたオブジェクトストレージバケットに接続できることを確認してください。ネットワーク制限によりアクセスできない場合は、代わりにデスティネーション クラスタ上のポッド内からTrident Protect CLIを実行してください。
- AWS セッション トークンの有効期限が、長時間実行される復元操作に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。
  - 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
  - AWS リソースの認証情報の詳細については、 "AWSのドキュメント"を参照してください。

手順

1. Trident Protect CLIプラグインを使用して、デスティネーション クラスタにAppVault CRが存在することを確認します：

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

AppVault CR がデスティネーション クラスタに存在しない場合は、"Trident Protect

AppVaultオブジェクトを使用してバケットを管理する"の手順に従って作成します。

2. デスティネーション クラスタで利用可能なAppVaultのバックアップコンテンツを表示し、復元するバックアップの `appArchivePath`をメモします：

```
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内の利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、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名前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 クラスタに復元します：

CRを使用する場合は、アプリケーションのリストアに使用する名前空間がデスティネーション クラスタに存在することを確認してください。

**CR**を使用する

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.appArchivePath**：（必須）AppVault内のバックアップコンテンツが保存されるパス。ステップ2のコマンドを使用してバックアップの内容を表示し、復元するバックアップの

`appArchivePath`を確認します。

- **spec.destinationApplicationName**: （オプション）復元されたアプリケーションの名前。指定した場合、復元されたアプリケーションはこの名前を使用します。指定しない場合、復元されたアプリケーションは元のアプリケーション名を使用します。
- **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

  次に例を示します。

```
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
  destinationApplicationName: my-new-app-name
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. `trident-protect-backup-restore-cr.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-backup-restore-cr.yaml
```

**CLI** を使用する

1. 次のコマンドを使用してアプリケーションを復元し、括弧内の値を環境の情報に置き換えます。namespace-mapping 引数は、コロンで区切られた名前空間を使用して、source1：dest1 、source2：dest2 の形式でソース名前空間を正しい宛先名前空間にマッピングします。次に例を示します。

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

スナップショットから別の **namespace** にリストアする

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 は新しい名前空間にアプリケーションを復元し、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設定します。

- SnapshotRestoreは `spec.storageClassMapping`属性をサポートしていますが、ソース ストレージ クラスとデスティネーション ストレージ クラスが同じストレージ バックエンドを使用する場合のみです。異なるストレージ バックエンドを使用する `StorageClass`に復元しようとすると、復元処理は失敗します。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみネームスペースを自動的に作成します。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

**CR**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

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

   - **spec.destinationApplicationName**: （オプション）復元されたアプリケーションの名前。指定した場合、復元されたアプリケーションはこの名前を使用します。指定しない場合、復元されたアプリケーションは元のアプリケーション名を使用します。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     ```

```
kind: SnapshotRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-snapshot-path
  namespaceMapping: [{"source": "my-source-namespace",
"destination": "my-destination-namespace"}]
```

3. （オプション）アプリケーションの特定のリソースのみを選択して復元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かを指定するフィルタリングを追加します。

> Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。例えば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は、関連付けられたポッドも復元します。

- **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須）`Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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演算で一致し、各要素内のフィールド（group、kind、version）はAND演算で一致します。
    - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
    - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
    - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
    - **resourceMatchers[].names**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前。
    - **resourceMatchers[].namespaces**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前空間。
    - **resourceMatchers[].labelSelectors**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`

      次に例を示します。

```
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. `trident-protect-snapshot-restore-cr.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-snapshot-restore-cr.yaml
```

**CLI** を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを別のネームスペースにリストアします。
   - その `snapshot`引数は、名前空間とスナップショット名を次の形式で使用します `<namespace>/<name>`。
   - `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デスティネーション名前空間にマッピングします（形式は `source1:dest1,source2:dest2`）。

     次に例を示します。

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name <custom_app_name> \
-n <application_namespace>
```

スナップショットから元の名前空間に復元する

スナップショットはいつでも元の名前空間に復元できます。In Placeリストアを実行すると、Trident Protectは、無効なリカバリポイントを防止するために、保護スケジュールと進行中の操作を自動的に管理します。

- 復元が開始される前に、アプリケーションに対して有効になっているすべての保護スケジュールが無効になります。これにより、アプリケーションリソースの復元中に、スケジュールされたバックアップやスナップショットが実行されるのを防ぎます。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。既に無効化されているスケジュールは、引き続き無効化されたままです。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元処理は続行され、復元CRステータスに警告が記録されます。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

**CR**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-ipr-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

```
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
```

+ .（オプション）アプリケーションの特定のリソースのみを選択して復元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かを指定するフィルタリングを追加します。

+ 注記：Trident Protect は、ユーザーが選択したリソースとの関連性に基づいて、一部のリソースを自動的に選択します。例えば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は、関連付けられたポッドも復元します。

+ * **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または`Exclude`を使用して、`resourceMatchers`で定義されたリソースを含めるか除外します。含めるリソースまたは除外するリソースを定義するには、以下の`resourceMatchers`パラメータを追加します：**`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**演算で一致し、各要素内のフィールド（**`group`**、**`kind`**、**`version`**）

はAND演算で一致します。* **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。* **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。* **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。* **resourceMatchers[].names**：（オプション）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**：（オプション）フィルタリングするリソースのKubernetes metadata.nameフィールドにある名前空間。* **resourceMatchers[].labelSelectors**：（オプション） "Kubernetesドキュメント"で定義されたリソースのKubernetes metadata.nameフィールド内のラベルセレクタ文字列。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"]
```

+ . `trident-protect-snapshot-ipr-cr.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-snapshot-ipr-cr.yaml
```

**CLI を使用する**

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを元の名前空間に復元します。次に例を示します。

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

リストア処理のステータスを確認する

コマンド ラインを使用して、進行中、完了、または失敗した復元処理のステータスを確認できます。

手順

1. 復元操作のステータスを取得するには、次のコマンドを使用します。括弧内の値は、ご使用の環境の情報に置き換えてください：

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

高度な **Trident Protect** リストア設定を使用する

## 注釈、名前空間設定、ストレージ オプションなどの詳細設定を使用して、特定の要件を満たすように復元操作をカスタマイズできます。

リストアおよびフェイルオーバー処理中のネームスペースのアノテーションとラベル

復元およびフェイルオーバー処理中に、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は、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、既存のラベルまたはアノテーションはソースネームスペースの値と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またはアノテーションは変更されません。

Red Hat OpenShiftを使用する場合は、OpenShift環境における名前空間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。名前空間アノテーションにより、リストアされたポッドがOpenShiftセキュリティコンテキスト制約（SCC）で定義された適切な権限とセキュリティ設定に準拠し、権限の問題なく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、"OpenShiftセキュリティコンテキスト制約ドキュメント"を参照してください。

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 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`を設定することで、デスティネーションネームスペース内の特定のアノテーションが上書きされるのを防ぐことができます。次に例を示します。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

リストアまたはフェイルオーバー操作を実行すると、 `restoreSkipNamespaceAnnotations`および `restoreSkipNamespaceLabels`で指定された名前空間アノテーションとラベルは、リストアまたはフェイルオーバー操作から除外されます。これらの設定は、Helmの初回インストール時に必ず構成してください。詳細については、"追加のTrident Protectヘルムチャート設定を構成する"を参照してください。

Helm を使用して `--create-namespace`フラグを指定してソースアプリケーションをインストールした場合、`name`ラベルキーには特別な処理が適用されます。リストアまたはフェイルオーバープロセス中に、 Trident Protect はこのラベルをデスティネーションネームスペースにコピーしますが、ソースからの値がソースネームスペースと一致する場合は、値をデスティネーションネームスペースの値に更新します。この値がソースネームスペースと一致しない場合は、変更されずにデスティネーションネームスペースにコピーされます。

例

次の例は、それぞれ異なる注釈とラベルを持つソース名前空間と宛先名前空間を示しています。操作の前後の宛先名前空間の状態や、宛先名前空間で注釈とラベルがどのように結合または上書きされるかを確認できます。

リストアまたはフェイルオーバー処理の前に

次の表は、リストアまたはフェイルオーバー処理前のサンプル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ています：

| 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） | • annotation.one/key："updatedvalue"<br>• annotation.two/key："true" | • environment=production<br>• コンプライアンス＝hipaa<br>• name=ns-1 |
| ネームスペースns-2（宛先） | • annotation.one/key："true"<br>• annotation.three/key："false" | • ロール＝database |

リストア処理後

次の表は、復元またはフェイルオーバー操作後の宛先名前空間の例の状態を示しています。いくつかのキーが追加され、いくつかは上書きされ、 `name`ラベルは宛先名前空間と一致するように更新されました：

| 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-2（宛先） | • annotation.one/key："updatedvalue"<br>• annotation.two/key："true"<br>• annotation.three/key："false" | • name=ns-2<br>• コンプライアンス＝hipaa<br>• environment=production<br>• ロール＝database |

サポートされているフィールド

このセクションでは、復元操作に使用できる追加のフィールドについて説明します。

ストレージクラスのマッピング

`spec.storageClassMapping`属性は、ソース アプリケーションに存在するストレージクラスからターゲット クラスタ上の新しいストレージクラスへのマッピングを定義します。異なるストレージクラスを持つクラスタ間でアプリケーションを移行する場合や、`BackupRestore`操作のストレージ

バックエンドを変更する場合にこれを使用できます。

例：

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

サポートされているアノテーション

このセクションでは、システムにおける様々な動作を設定するためにサポートされている注釈を一覧表示します。ユーザーがアノテーションを明示的に設定しない場合、システムはデフォルト値を使用します。

| 注釈 | タイプ | 概要 | デフォルト値 |
|---|---|---|---|
| protect.trident.netapp.io/data-mover-timeout-sec | string | データ ムーバー処理が停止する最大時間（秒単位）。 | "300" |
| protect.trident.netapp.io/kopia-content-cache-size-limit-mb | string | Kopia コンテンツ キャッシュの最大サイズ制限（メガバイト単位）。 | "1000" |
| protect.trident.netapp.io/pvc-bind-timeout-sec | string | 新しく作成されたPersistentVolumeClaims（PVC）が`Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。環境すべての復元CRタイプ（BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore）に適用されます。ストレージバックエンドまたはクラスターの処理に時間がかかることが多い場合は、より大きな値を使用してください。 | 「1200」（20分） |

カスタムリソースを使用して復元する際にリソースを変更します

リソース変換機能を使用すると、リソースの復元中にリソースを変更できます。これは、復元されたバージョンがオリジナルと異なる必要がある場合に役立ちます。たとえば、アプリケーションを別のクラスタに復元する際にConfigMapのホスト名を更新する場合などです。

リソース変更の仕組み

`SnapshotRestore`、 `BackupRestore`、`AppMirrorRelationship`、およびその他のリストア リソースの`transformations`フィールドを使用すると、リストア プロセス中に Kubernetes リソースを変更できます。これは、リストアされたアプリケーションをオリジナルとは異なる方法

で構成する必要がある場合（ホスト名、レジストリ
URL、リソース制限、環境変数の変更など）に便利です。

リソース変換では、 "RFC 6902 JSON Patch"操作と "RFC 6901 JSONポインタ"パスを使用して、Kubernetes リソース内の特定のフィールドをターゲットにして変更します。

以下に、リソース変換を含む復元オブジェクトの基本的な構造を示します。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: my-restore
  namespace: target-namespace
spec:
  appVaultRef: my-vault
  appArchivePath: /path/to/snapshot
  namespaceMapping:
    - source: source-ns
      destination: target-ns
  transformations:
    - resource:
        kind: ConfigMap       # Required: resource kind
        group: ""             # Optional: API group (empty for core
resources)
        version: ""           # Optional: API version
        name: ""              # Optional: specific resource name
      operations:
        - op: replace         # Operation type
          path: "/data/key"   # JSON Pointer path
          value: "new-value"  # New value (for add/replace/test)
```

サポートされているリソース

以下の条件に一致するリソースに対して、リソース変換を使用できます：

- **kind** （必須）：Kubernetes リソースの種類（例： `ConfigMap`、 `Deployment`、 `Pod`）
- グループ（オプション）：APIグループ（例： `apps`、 `route.openshift.io`）—コアリソースの場合は省略
- **version** （オプション）：APIバージョン（例： `v1`、 `v1beta1`）
- 名前（オプション）：名前で指定した特定のリソースにのみ適用します

PersistentVolumeClaimsと名前空間の変更は現在サポートされていません。

サポートされている操作

リソースを変更するには、以下の操作を使用できます： `add`、 `copy`、 `move`、 `remove`、 `replace`、および `test`。

リソースに値を追加する

`add`操作を使用して、指定されたパスに新しいフィールドまたは値を追加します。オブジェクトや配列にデータを追加できます。

**CR**を使用する

次の例では、ノードセレクタを `Deployment`リソースに追加します：

```
transformations:
  - resource:
      kind: Deployment
    operations:
      - op: add
        path: "/spec/template/spec/nodeSelector"
        value:
          "topology.kubernetes.io/zone": "us-east-1a"
          disktype: "ssd"
```

**CLI** を使用する

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'apps,v1,Deployment:add{"path":"/spec/template/spec/nodeSelector","valu
e":{"topology.kubernetes.io/zone":"us-east-1a","disktype":"ssd"}}'
```

リソース内の値をコピーする

`copy`操作を使用して、同じリソース内のあるパスから別のパスに値をコピーします。ソースは変更されません。

**CR**を使用する

次の例では、 `ConfigMap`オブジェクトのデータキーを複製します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: copy
        from: "/data/source-key"
        path: "/data/backup-key"
```

**CLI** を使用する

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation ',v1,ConfigMap:copy{"from":"/data/source-
key","path":"/data/backup-key"}'
```

リソース内で値を移動する

`move`操作を使用して、同じリソース内であるパスから別のパスに値を移動します。ソースが削除され、値が宛先に配置されます。

**CR**を使用する

次の例では、 `ConfigMap`オブジェクトのデータキーの名前を変更します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: move
        from: "/data/OLD_KEY"
        path: "/data/NEW_KEY"
```

**CLI** を使用する

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
```

```
  --transformation
',v1,ConfigMap:move{"from":"/data/OLD_KEY","path":"/data/NEW_KEY"}'
```

リソースから値を削除する

`remove`操作を使用して、指定されたパスにあるフィールドまたは値を削除します。

**CR**を使用する

次の例では、`ConfigMap`リソースから注釈を削除します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: remove
        path: "/metadata/annotations/kubectl.kubernetes.io~1last-
applied-configuration"
```

上記の例のパスでは、`~1`は`/`のJSONポインターのエスケープシーケンスです。

**CLI** を使用する

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:remove{"path":"/metadata/annotations/kubectl.kubernetes.
io~1last-applied-configuration"}'
```

リソース内の値を置き換える

`replace`操作を使用して、指定されたパスにあるリソース内の既存の値を置き換えます。JSONパスは既に存在している必要があります。

**CR**を使用する

次の例では、`Route`オブジェクトのホスト名を変更します：

```
transformations:
  - resource:
      kind: Route
      group: route.openshift.io
    operations:
      - op: replace
        path: "/spec/host"
        value: "prod.example.com"
```

**CLI を使用する**

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'route.openshift.io,v1,Route:replace{"path":"/spec/host","value":"prod.
example.com"}'
```

リソースの変更をテストする

`test`操作を使用して、他の操作が適用される前に、パス上の値が期待値と一致することを確認します。テストが失敗した場合、変更全体がロールバックされます。

**CRを使用する**

次の例では、 `environment`が `staging`に設定されている場合にのみ `database-host`を更新します。

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: test
        path: "/data/environment"
        value: "staging"
      - op: replace
        path: "/data/database-host"
        value: "prod-db.example.com"
```

**CLI を使用する**

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:test{"path":"/data/environment","value":"staging"},repla
ce{"path":"/data/database-host","value":"prod-db.example.com"}'
```

## NetApp SnapMirrorとTrident Protectを使用したアプリケーションのレプリケート

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、同じクラスタ上または異なるクラスタ間で、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データとアプリケーションの変更を複製できます。

### リストアおよびフェイルオーバー処理中のネームスペースのアノテーションとラベル

復元およびフェイルオーバー処理中に、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は、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、既存のラベルまたはアノテーションはソースネームスペースの値と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またはアノテーションは変更されません。

> Red Hat OpenShiftを使用する場合は、OpenShift環境における名前空間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。名前空間アノテーションにより、リストアされたポッドがOpenShiftセキュリティコンテキスト制約（SCC）で定義された適切な権限とセキュリティ設定に準拠し、権限の問題なく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、"OpenShiftセキュリティコンテキスト制約ドキュメント"を参照してください。

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`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`を設定することで、デスティネーションネームスペース内の特定のアノテーションが上書きされるのを防ぐことができます。次に例を示します。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

> リストアまたはフェイルオーバー操作を実行すると、`restoreSkipNamespaceAnnotations`および`restoreSkipNamespaceLabels`で指定された名前空間アノテーションとラベルは、リストアまたはフェイルオーバー操作から除外されます。これらの設定は、Helmの初回インストール時に必ず構成してください。詳細については、"追加のTrident Protectヘルムチャート設定を構成する"を参照してください。

Helm を使用して `--create-namespace`フラグを指定してソースアプリケーションをインストールした場合、`name`ラベルキーには特別な処理が適用されます。リストアまたはフェイルオーバープロセス中に、 Trident Protect はこのラベルをデスティネーションネームスペースにコピーしますが、ソースからの値がソースネームスペースと一致する場合は、値をデスティネーションネームスペースの値に更新します。この値がソースネームスペースと一致しない場合は、変更されずにデスティネーションネームスペースにコピーされます。

例

次の例は、それぞれ異なる注釈とラベルを持つソース名前空間と宛先名前空間を示しています。操作の前後の宛先名前空間の状態や、宛先名前空間で注釈とラベルがどのように結合または上書きされるかを確認できます。

リストアまたはフェイルオーバー処理の前に

次の表は、リストアまたはフェイルオーバー処理前のサンプル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ています：

| 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） | • annotation.one/key："updatedvalue"<br>• annotation.two/key："true" | • environment=production<br>• コンプライアンス＝hipaa<br>• name=ns-1 |
| ネームスペースns-2（宛先） | • annotation.one/key："true"<br>• annotation.three/key："false" | • ロール＝database |

リストア処理後

次の表は、復元またはフェイルオーバー操作後の宛先名前空間の例の状態を示しています。いくつかのキーが追加され、いくつかは上書きされ、 `name`ラベルは宛先名前空間と一致するように更新されました：

| 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-2（宛先） | • annotation.one/key："updatedvalue"<br>• annotation.two/key："true"<br>• annotation.three/key："false" | • name=ns-2<br>• コンプライアンス＝hipaa<br>• environment=production<br>• ロール＝database |

Trident Protectを設定して、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ことができます。"Trident Protectを使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください"。

フェイルオーバーおよびリバース操作中の実行フック

AppMirror関係を使用してアプリケーションを保護する場合、フェイルオーバーおよびリバース操作中に知っておくべき実行フックに関連する特定の動作があります。

- フェイルオーバー中、実行フックはソースクラスタからデスティネーション クラスタに自動的にコピーされます。手動で再作成する必要はありません。フェイルオーバー後、実行フックがアプリケーション上に存在し、関連するアクション中に実行されます。

- リバースまたはリバース再同期中に、アプリケーションの既存の実行フックはすべて削除されます。ソース アプリケーションがデスティネーション アプリケーションになると、これらの実行フックは無効になり、実行されないように削除されます。

実行フックの詳細については、"Trident Protect 実行フックを管理する"を参照してください。

レプリケーション関係を設定する

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 で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）を取得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）
- スナップショットを作成する時間を設定する

手順

1. ソースクラスタで、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、"AppVault カスタムリソース"の例を環境に合わせて変更します：

**CR** を使用して **AppVault** を作成する

a. 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-appvault-primary-source.yaml`）。

b. 次の属性を設定します：

- **metadata.name**：（必須）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するため、選択した名前をメモしておいてください。
- **spec.providerConfig**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な設定を保存します。bucketNameおよびプロバイダーに必要なその他の詳細を選択してください。レプリケーション関係に必要な他のCRファイルがこれらの値を参照するため、選択した値をメモしておいてください。他のプロバイダーを使用したAppVault CRの例については、"AppVault カスタムリソース"を参照してください。
- **spec.providerCredentials**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な認証情報への参照を保存します。
  - **spec.providerCredentials.valueFromSecret**：（必須）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**：（必須）選択するシークレットの有効なキー。
    - **name**：（必須）このフィールドの値を含むシークレットの名前。同じ名前空間に存在する必要があります。
  - **spec.providerCredentials.secretAccessKey**：（必須）プロバイダーにアクセスするために使用するアクセス キー。**name** は **spec.providerCredentials.valueFromSecret.name** と一致する必要があります。
- **spec.providerType**：（必須）バックアップを提供するものを決定します。例：NetApp ONTAP S3、汎用S3、Google Cloud、またはMicrosoft Azure。可能な値：
  - aws
  - azure

- gcp
- generic-s3
- ontap-s3
- storagegrid-s3

c. `trident-protect-appvault-primary-source.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-appvault-primary-source.yaml -n
trident-protect
```

**CLI**を使用して**AppVault**を作成する

a. AppVaultを作成します。括弧内の値は、実際の環境の情報に置き換えてください：

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

2. ソース クラスタで、ソース アプリケーション CR を作成します：

**CR**を使用してソースアプリケーションを作成する

a. 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-app-source.yaml`）。

b. 次の属性を設定します：

- **metadata.name**：（必須）アプリケーションのカスタム リソースの名前。レプリケーション関係に必要な他の CR ファイルはこの値を参照するため、選択した名前をメモしておいてください。
- **spec.includedNamespaces**：（必須）ネームスペースと関連付けられたラベルの配列。ネームスペース名を使用し、必要に応じてラベルでネームスペースの範囲を絞り込んで、ここにリストされているネームスペース内に存在するリソースを指定します。アプリケーション ネームスペースは、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. `trident-protect-app-source.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-app-source.yaml -n my-app-
namespace
```

**CLI** を使用してソースアプリケーションを作成する

a. ソース アプリケーションを作成します。次に例を示します。

```
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 クラスターでソース アプリケーションのスナップショットを取得します。このスナップショットは、デスティネーション クラスタ上のアプリケーションの基盤として使用されます。この手順をスキップした場合は、最新のスナップショットを取得するために、次にスケジュールされたスナップショットが実行されるまで待つ必要があります。オンデマンド スナップショットを作成するには、"オンデマンドスナップショットを作成する"を参照してください。
4. ソース クラスタで、レプリケーション スケジュール CR を作成します：

> ⓘ 下記のスケジュールに加えて、ピアリングされたONTAPクラスタ間で共通のスナップショットを維持するために、7日間の保存期間を持つ個別の日次スナップショットスケジュールを作成することをお勧めします。これにより、スナップショットは最大7日間利用できるようになりますが、保存期間はユーザーの要件に基づいてカスタマイズできます。
>
> フェイルオーバーが発生した場合、システムはこれらのスナップショットを最大7日間使用して逆の操作を行うことができます。このアプローチでは、すべてのデータではなく、最後のスナップショット以降に行われた変更のみが転送されるため、逆のプロセスがより高速かつ効率的になります。
>
> アプリケーションの既存のスケジュールがすでに必要な保持要件を満たしている場合は、追加のスケジュールは必要ありません。

**CR** を使用してレプリケーションスケジュールを作成する

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、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**：（必須）スケジュールカスタムリソースの名前。
- **spec.appVaultRef**：（必須）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**：（必須）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**：（必須）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**：`Custom`に設定する必要があります。
- **spec.recurrenceRule**：UTC 時間での開始日と繰り返し間隔を定義します。
- **spec.snapshotRetention**：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
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

i. `trident-protect-schedule.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-schedule.yaml -n my-app-
namespace
```

**CLI** を使用してレプリケーションスケジュールを作成する

a. 括弧内の値を環境の情報に置き換えて、レプリケーション スケジュールを作成します：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

例：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. デスティネーション クラスタで、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を作成し、名前を付けます（例： `trident-protect-appvault-primary-destination.yaml`）。
6. CR を適用します：

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

7. 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用のデスティネーションAppVault CRを作成します。ストレージプロバイダに応じて、"AppVault カスタムリソース"の例を環境に合わせて変更します：
   a. 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-appvault-secondary-destination.yaml`）。
   b. 次の属性を設定します：
      - **metadata.name**：（必須）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するため、選択した名前をメモしておいてください。
      - **spec.providerConfig**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な設定を保存します。 `bucketName`および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ファイルがこれらの値を参照するため、選択した値をメモしておいてください。他のプロバイダーを使用したAppVault CRの例については、"AppVault カスタムリソース"を参照してください。
      - **spec.providerCredentials**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な認証情報への参照を保存します。
        - **spec.providerCredentials.valueFromSecret**：（必須）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**：（必須）選択するシークレットの有効なキー。
          - **name**：（必須）このフィールドの値を含むシークレットの名前。同じ名前空間に存在する必要があります。
        - **spec.providerCredentials.secretAccessKey**：（必須）プロバイダーにアクセスするために使用するアクセス キー。**name** は **spec.providerCredentials.valueFromSecret.name** と一致する必要があります。
      - **spec.providerType**：（必須）バックアップを提供するものを決定します。例：NetApp ONTAP S3、汎用S3、Google Cloud、またはMicrosoft Azure。可能な値：
        - aws
        - azure

- gcp
- generic-s3
- ontap-s3
- storagegrid-s3

c. `trident-protect-appvault-secondary-destination.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-appvault-secondary-destination.yaml
-n trident-protect
```

8. デスティネーション クラスタで、AppMirrorRelationship CR ファイルを作成します。

CR を使用する場合は、CR を適用する前に、デスティネーション ネームスペースを手動で作成します。Trident Protect は、CLI を使用する場合にのみネームスペースを自動的に作成します。

**CR**を使用して**AppMirrorRelationship**を作成する

a. 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-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**：（必須）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**：（必須）デスティネーション名前空間とソース名前空間は、それぞれのアプリケーション CR で定義されたアプリケーション名前空間と一致する必要があります。
- **spec.sourceAppVaultRef**：（必須）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**：（必須）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.sourceApplicationUID**：（必須）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 UID と一致する必要があります。
- **spec.storageClassName**：（オプション）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。ストレージクラスは、ソース環境とピアリングされているONTAPストレージVMにリンクされている必要があります。ストレージクラスが指定されていない場合は、クラスタのデフォルトのストレージクラスがデフォルトで使用されます。
- **spec.storageClassMapping**: （オプション） 同じ AMR 内の個々の PVC のソースストレージクラスをデスティネーションストレージクラスにマッピングします。各エントリには `source`および `destination`ストレージクラス名が必要です。設定すると、`storageClassMapping`は `storageClassName`を完全にオーバーライドします。一致するソースエントリのない PVC はクラスタのデフォルトのストレージクラスを使用し、`storageClassName`は使用しません。

AppMirrorRelationship CR で `storageClassMapping`を使用する際は、以下の制限事項に注意してください。

- `storageClassMapping`は `storageClassName`を完全にオーバーライドします。両方のフィールドが設定されている場合、 `storageClassName`は無視されます。
- `storageClassMapping`に一致するエントリがないソースストレージクラスを持つ PVC は、 `storageClassName`ではなく、クラスターのデフォルトストレージクラスを使用します。
- 各エントリーには、 `source`と `destination`の値が必要です。
- デスティネーション ストレージクラスを変更すると、既存のデスティネーション PVC の削除と再作成が強制される場合があります。

- **spec.recurrenceRule**：UTC 時間での開始日と繰り返し間隔を定義します。

  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-vsim-2
```

c. `trident-protect-relationship.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-relationship.yaml -n my-app-
namespace
```

**CLI** を使用して **AppMirrorRelationship** を作成する

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

：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --source-app-vault <my_vault_name> --recurrence
-rule <rule> --namespace-mapping <ns_mapping> --source-app-id
<source_app_UID> --source-app <my_source_app_name> --storage
-class <storage_class_name> --storageclass-mapping
<src_class:dst_class> -n <application_namespace>
```

`--storageclass-mapping`を使用して、個々のソースストレージクラスをデスティネーションストレージクラスにマッピングします。カンマで区切られた1つ以上の `source:destination`ペアを指定します（例： `"class-a:class-b,class-c:class-d"）。`*:destination-class`をワイルドカードとして使用して、マッピングされていないすべてのPVCを単一のデスティネーションストレージクラスにマッピングします。`--storageclass-mapping`が設定されている場合、`--storage-class`は無視されます。

単一のデスティネーション ストレージクラスを使用した例(**--storage-class**）：

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. （オプション）デスティネーション クラスタで、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 クラスタへのフェイルオーバー

Trident Protect を使用すると、レプリケートされたアプリケーションをデスティネーション クラスタにフェイルオーバーできます。この手順では、レプリケーション関係を停止し、デスティネーション クラスタでアプリをオンラインにします。Trident Protect は、ソース クラスタ上のアプリが稼働中であっても停止しません。

手順

1. デスティネーション クラスタで、AppMirrorRelationship CR ファイル（例： `trident-protect-relationship.yaml`）を編集し、**spec.desiredState** の値を `Promoted`に変更します。

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）fail over されたアプリケーションで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
```

フェイルオーバーしたレプリケーション関係を再同期する

再同期操作により、レプリケーション関係が再確立されます。再同期操作を実行すると、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 クラスタ上の実行中のアプリケーションに対して行われ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. オプション：ソース クラスターで、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。これにより、ソース クラスターからの最新の変更が確実にキャプチャされます。
2. デスティネーション クラスタで、AppMirrorRelationship CR ファイル（例： `trident-protect-relationship.yaml`）を編集し、spec.desiredState の値を `Established`に変更します。
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、それらを削除します。スケジュールが残っていると、ボリューム スナップショットが失敗します。

フェイルオーバーしたレプリケーション関係を逆再同期する

フェー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、それらを削除します。
2. 元々関係を設定するために使用した CR ファイルを反対側のクラスタに適用して、レプリケーション関係を設定します。
3. 新しいデスティネーション（元のソースクラスタ）が両方のAppVault CRで設定されていることを確認します。
4. 逆方向の値を設定して、反対側のクラスタにレプリケーション関係を設定します。

アプリケーションのレプリケーション方向を逆にする

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デスティネーション 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、デスティネーション アプリにフェイルオーバーする前にデータをデスティネーションに複製します。

この状況では、ソースとデスティネーションが入れ替わっています。

手順

1. ソース クラスタで、シャットダウン スナップショットを作成します：

**CR**を使用してシャットダウンスナップショットを作成する

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、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**：（必須）カスタム リソースの名前。
- **spec.AppVaultRef**：（必須）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**：（必須）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. `trident-protect-shutdownsnapshot.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-shutdownsnapshot.yaml -n my-app-namespace
```

**CLI** を使用してシャットダウン **Snapshot** を作成する

a. 括弧内の値を環境の情報に置き換えて、シャットダウン スナップショットを作成します。次に例を示します。

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

2. ソース クラスタで、シャットダウン Snapshot が完了したら、シャットダウン 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. ソース クラスタで、次のコマンドを使用して **shutdownsnapshot.status.appArchivePath** の値を見つけ、ファイル パスの最後の部分（ベース名とも呼ばれます。最後のスラッシュの後のすべてです）を記録します：

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

4. 新しいデスティネーション クラスタから新しいソース クラスタへのフェイルオーバーを実行します。次の変更を加えます：

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

5. [フェイルオーバーしたレプリケーション関係を逆再同期する]で逆再同期の手順を実行します。
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

逆レプリケーションにより、次のアクションが発生します：

- 元のソース アプリの Kubernetes リソースのスナップショットが取得されます。
- 元のソース アプリのポッドは、アプリの 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 はそのまま残ります）。
- 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レプリケートされたボリューム データを使用して、シャットダウン前の Snapshot からリストアされます。
- レプリケーションは逆方向に再確立されます。

#### 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する

Trident Protect を使用すると、次の一連の操作を使用して、フェイルオーバー操作後に「フェイルバック」を実現できます。このワークフローでは、元のレプリケーション方向を復元するために、Trident Protect は、レプリケーションの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を完了した関係から開始され、次の手順が含まれます：

- 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。フェイルオーバー手順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. [フェイルオーバーしたレプリケーション関係を逆再同期する]の手順を実行します。
2. [アプリケーションのレプリケーション方向を逆にする]の手順を実行します。

#### 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、関係のない 2 つの別個のアプリケーションが作成されます。

手順

1. 現在の宛先クラスタで、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 データを復元することで、アプリケーションをクラスター間または異なる

ストレージ クラスに移行できます。

> アプリケーションを移行すると、アプリケーション用に設定されたすべての実行フックも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存在する場合、リストア処理の一部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およびリストア処理を実行するには、特定のバックアップおよびリストアタスクを自動化できます。

同じクラスタにクローンする

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. "スナップショットを作成する"。
   b. "バックアップを作成する"。
2. 同じクラスタで、スナップショットを作成したかバックアップを作成したかに応じて、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復元する"。
   b. "バックアップからデータを復元する"。

別のクラスタにクローンする

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。Trident Protectがデスティネーション クラスタにインストールされていることを確認してください。

> 異なるクラスタ間でアプリケーションを複製するには、"SnapMirrorレプリケーション"を使用します。

手順

1. "バックアップを作成する"。
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復元する"。

アプリケーションをあるストレージ クラスから別のストレージ クラスに移行する

バックアップをデスティネーション ストレージ クラスに復元することで、アプリケーションを 1 つのストレージ クラスから別のストレージ クラスに移行できます。

たとえば（復元 CR からシークレットを除外）：

```
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**を使用してスナップショットを復元する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、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.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

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

   - **spec.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     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. オプションとして、復元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、特定のラベルでマークされ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須）`include or exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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演算で一致し、各要素内のフィールド（group、kind、version）はAND演算で一致します。
       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
       - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
       - **resourceMatchers[].names**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前。
       - **resourceMatchers[].namespaces**：（オプション）フィルタリングするリソースのKubernetes metadata.name フィールド内の名前空間。
       - **resourceMatchers[].labelSelectors**：（オプション）"Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`

         次に例を示します。

```
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. `trident-protect-snapshot-restore-cr.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-snapshot-restore-cr.yaml
```

**CLI** を使用してスナップショットを復元する

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを別のネームスペースにリストアします。
   - その `snapshot`引数は、名前空間とスナップショット名を次の形式で使用します `<namespace>/<name>`。
   - `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デスティネーション名前空間にマッピングします（形式は `source1:dest1,source2:dest2`）。

     次に例を示します。

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

## KubeVirt 仮想マシンを Trident Protect で保護する

Trident Protectを使用すると、OpenShift Virtualization上で実行されているKubeVirt仮想マシン（VM）をバックアップおよびリストアできます。1つ以上のネームスペースにまたがるすべてのVMを一度に保護することも、`includedVirtualMachines`フィールドを使用して特定のVMを名前で指定して保護することもできます。Trident Protectは、バックアップ時に関連するすべてのVMリソースを自動的に収集します。リストア時には、アーカイブ全体を復旧することも、特定のVMを選択して個別にリストアすることもできます。

### KubeVirt 保護モード

Trident Protect は、保護する KubeVirt VM を定義する 2 つの方法をサポートしています。アプリケーションを定義する際に、いずれかのモードを選択してください。

| モード | フィールド | 概要 |
| --- | --- | --- |
| 名前空間ベース | `includedNamespaces` | 1つ以上の名前空間内の、一致するすべてのリソースを保護します。名前空間エントリで `labelSelector`を使用して、動的に含める仮想マシンを制御します。 |

| モード | フィールド | 概要 |
| --- | --- | --- |
| VMスコープ | `includedVirtualMachines` | 名前空間と名前で特定の KubeVirt VMを保護します。VMが複数の名前空間に分散している場合、またはすべての名前空間リソースを保護したくない場合は、このモードを使用してください。 |

必ず1つのフィールド（`includedNamespaces`または `includedVirtualMachines`）を`ApplicationSpec`に指定してください。同じアプリケーション内で両方を使用することはできません。

どちらのモードも追加のフィルタリングをサポートしています。

- `resourceFilter`を使用して、特定のリソースを含めるか除外するかを指定します。
- `includedClusterScopedResources`を使用して、 `StorageClass`オブジェクトなど、互換性のあるクラスタスコープのリソースを含めます。

> Trident Protect は、データ保護操作中に KubeVirt VM ファイルシステムをフリーズおよびアンフリーズして、一貫性を確保できます。設定の詳細については、"KubeVirt VM でデータを保護"を参照してください。

バックアップとスナップショット中に収集されたリソース

KubeVirt VM を含むアプリケーションのバックアップまたはスナップショットを作成する場合、Trident Protect は選択された VM リソースを収集し、それらに依存するリソースを自動的に検出します。

コレクションの動作は、次の2つの場合に適用されます。

- VM スコープのアプリケーションで `includedVirtualMachines`を使用するもの。
- 名前空間ベースのアプリケーションで、KubeVirt `VirtualMachine`リソースを含むもの。

**VM**スコープのコレクション

VMスコープのアプリケーションの場合、Trident Protectは、 `includedVirtualMachines`にリストされている各`VirtualMachine`から開始し、そのVMのリストアに必要な依存リソースを検出します。

| リソースタイプ | **Scope** | 発見方法 |
| --- | --- | --- |
| `VirtualMachine` | 名前空間付き | `includedVirtualMachines`リストからの完全一致。 |
| `VirtualMachineInstance` | 名前空間付き | VMが実行中の場合は、VMと同じ名前空間と名前を使用します。 |
| `VirtualMachineInstancetype` | 名前空間付き | 参照元 `spec.instancetype`（ `kind: VirtualMachineInstancetype`の場合）。 |
| `VirtualMachineClusterInstancetype` | クラスタ | `kind`が未設定の場合、または `kind: VirtualMachineClusterInstancetype`の場合に、`spec.instancetype`によって参照されます。 |
| `VirtualMachinePreference` | 名前空間付き | 参照元 `spec.preference`（ `kind: VirtualMachinePreference`の場合）。 |

| リソースタイプ | Scope | 発見方法 |
| --- | --- | --- |
| `VirtualMachineClusterPreference` | クラスタ | `kind`が未設定の場合、または `kind: VirtualMachineClusterPreference`の場合に、 `spec.preference` によって参照されます。 |
| `DataVolume` | 名前空間付き | 参照元 `spec.template.spec.volumes[].dataVolume.name`。 |
| `PersistentVolumeClaim` | 名前空間付き | DataVolume と同じ名前を使用する DataVolume バッキング PVC、 `persistentVolumeClaim.claimName`内の直接 PVC 参照、 `memoryDump.claimName`内のメモリダンプ PVC、および `ephemeral.persistentVolumeClaim.claimName`内の一時的な PVC から検出されます。 |
| `PersistentVolume` | クラスタ | 各収集済みPVCのバッキングPVから、 `spec.volumeName`を使用して検出されます。 |
| `Secret` | 名前空間付き | ボリュームシークレットから検出 `secret.secretName` ；containerDiskイメージのプルシークレット；CloudInitNoCloudおよびCloudInitConfigDriveのユーザーデータとネットワークデータのシークレット参照；Sysprepシークレット；KernelBootコンテナイメージのプルシークレット；およびAccessCredentialsのSSHまたはパスワードシークレット。 |
| `ConfigMap` | 名前空間付き | ボリューム ConfigMaps および Sysprep ConfigMaps の `configMap.name`から検出されました。 |
| `ServiceAccount` | 名前空間付き | `serviceAccount.serviceAccountName`のボリューム参照から検出されました。 |
| `NetworkAttachmentDefinition` | 名前空間付き | ネットワーク名に `/`が含まれていない場合、 `spec.template.spec.networks[].multus.networkName`内のMultusネットワークから検出されます。同一ネームスペースのNetworkAttachmentDefinition参照のみが収集されます。ネームスペース間の参照はスキップされます。 |
| `Pod` | 名前空間付き | ラベルセレクター `kubevirt.io=virt-launcher,vm.kubevirt.io/name=<vmName>`を使用して、 `virt-launcher` Podに対して検出されました。 |
| ストレージクラス、Tridentボリューム、およびTridentバックエンド | クラスタ | 仮想マシンに対して検出された PVC に基づいて、名前空間ベースのアプリケーションと同様の方法で収集されます。 |

VM スコープの名前空間リソースがすべて収集された後、Trident Protect は標準 `resourceFilter` を適用します（設定されている場合）。その後、クラスタスコープのリソース収集は通常どおり実行されます。

名前空間ベースのコレクション

名前空間ベースのアプリケーションの場合、Trident Protect は選択された名前空間をスキャンし、一致するすべてのリソースを収集します。KubeVirt `VirtualMachine` オブジェクトが見つかった場合、Trident Protect は、VMスコープのアプリケーションと同様に、インスタンスタイプ、設定、および `virt-launcher` Pod などの依存リソースを自動的に収集します。

これは、アプリケーションが定義されている場合でも、VM 固有のリソースがバックアップまたはスナップショットに含まれることを意味します `includedNamespaces`。

名前空間ベースのアプリケーション定義を使用して仮想マシンを保護する

名前空間ベースのアプリケーションを作成する場合、Trident Protect は選択された名前空間をスキャンし、KubeVirt VM を含むすべてのリソースを収集します。見つかった仮想マシンごとに、Trident Protect はその仮想マシンのバックアップと復元に必要な依存リソースも収集します。

名前空間ベースのアプリケーションCRの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
```

アプリケーションを作成したら、バックアップとスナップショットに標準の Trident Protect ワークフローを使用します：

- スナップショットYAMLの例については、"オンデマンドスナップショットを作成する"を参照してください。
- バックアップYAMLの例については、"オンデマンド バックアップを作成する"を参照してください。

名前空間ベースの**VM**アプリケーションのリストア動作

名前空間ベースのアプリケーションは選択された名前空間からリソースを収集するため、バックアップまたはスナップショットにはKubeVirtランタイムオブジェクトとKubeVirtが管理するメタデータが含まれる可能性があります。これらのオブジェクトの中には、ソースクラスタの実行時状態を表しているため、直接復元すべきでないものもあります。

復元中、Trident Protect は KubeVirt 対応の変換を適用して、ソースクラスターのメタデータを削除し、一時的な KubeVirt リソースをスキップします。これにより、KubeVirt は復元された環境で VM をクリーンに再作成できます。

| リソース | 変換 |
| --- | --- |
| VirtualMachine | `kubevirt.io/`で始まるラベルと注釈を削除します。 |
| VirtualMachineInstance | `kubevirt.io/`で始まるラベルと注釈を削除します。 |
| VirtualMachineInstanceMigration | 無視され、復元もされません。 |
| VirtualMachineSnapshot | 無視され、復元もされません。 |
| VirtualMachineSnapshotContent | 無視され、復元もされません。 |
| Pod (`virt-launcher`) | ポッドに `kubevirt.io=virt-launcher`ラベルがある場合はスキップされます。 |

Trident Protect は、依存関係の問題を回避するために、KubeVirt リソースを定められた順序で復元します。CDI DataVolumes (`cdi.kubevirt.io`) は、VM オブジェクトが作成される前に VM ディスクデータが利用可能になるように、最初に復元されます。その後、VirtualMachines (`kubevirt.io`) が VirtualMachineInstances より先に復元されます。

ラベルセレクタを使用した動的**VM**保護

各仮想マシン名を列挙せずに動的に仮想マシンを保護したい場合は、ラベルセレクタを使用した名前空間ベースのアプリケーション定義を使用してください。

VirtualMachine CR のみにラベルが必要です。DataVolumes、PVC、シークレット、ConfigMaps などの依存リソースは自動的に収集されます。

手順

1. 仮想マシン CR に共有ラベルを付ける：

```
kubectl label virtualmachine <vm_name 1> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_2> app=<label_value> -n
<vm_namespace>
```

2. アプリケーション CR を `includedNamespaces`および `labelSelector`で作成します：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
      labelSelector:
        matchLabels:
          app: <label_value>
```

3. 標準の Trident Protect ワークフローを使用してバックアップとリストアを作成します。

追加情報：

- ラベルを変更することで、いつでも仮想マシンを保護対象に追加または削除できます。

```
kubectl label virtualmachine <vm_name_to_add> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_to_remove> app
```

```
=<different_label_value> -n <vm_namespace>
```

- ラベルセレクタは複数の名前空間にわたって使用することもできます。

```
spec:
  includedNamespaces:
    - namespace: <vm_namespace_1>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
    - namespace: <vm_namespace_2>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
```

特定の仮想マシンを**includedVirtualMachines**で保護する

名前付きセットのVirtualMachinesを1つまたは複数の名前空間にわたって保護し、アーカイブから特定のVMを選択的にリストアする場合は、 `includedVirtualMachines`を使用します。

アプリケーションCRとリストアCRの両方で同じ `includedVirtualMachines`セレクター形式を使用します。

このセレクターは、さまざまな状況で使用できます。

- アプリケーションCRでは、どのVMが保護されるかを定義します。
- リストアCRでは、バックアップまたはスナップショット アーカイブからどのVMをリストアするかを定義します。

CRの復元時、 `includedVirtualMachines`は任意です。このフィールドを含めない場合、アーカイブ全体が復元されます。指定した場合、リストに記載されている仮想マシンとその依存リソースのみが復元されます。

以下のYAMLは、共有セレクタ構造を示しています。

```
includedVirtualMachines:
  - namespace: <vm_namespace_1>
    names:
      - <vm_name_1>
      - <vm_name_2>
  - namespace: <vm_namespace_2>
    names:
      - <vm_name_3>
```

複数のエントリー `includedVirtualMachines`は異なる名前空間を参照できるため、マルチ名前

空間のVM対応アプリケーションの定義とリストアが可能になります。

| フィールド | タイプ | 概要 |
|---|---|---|
| namespace | string | 対象となる仮想マシンオブジェクトを含むKubernetesネームスペース。 |
| 名前 | 文字列のリスト | その名前空間内の1つ以上の仮想マシン名。 |

**VM**ベースのアプリケーションの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace_1>
      names:
        - <vm_name_1>
        - <vm_name_2>
    - namespace: <vm_namespace_2>
      names:
        - <vm_name_3>
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

特定の**KubeVirt VM**をリストアする

リストアCRで `includedVirtualMachines`を使用して、バックアップ、スナップショット、またはレプリケートされたスナップショットアーカイブから選択したKubeVirt VMのみをリストアできます。

`includedVirtualMachines`がリストア CR で指定されている場合、Trident Protect はリストされている VirtualMachines とその依存リソース（ `VirtualMachineInstance`、 `DataVolume`、 `PersistentVolumeClaim`、 `PersistentVolume`、 `Secret`、 `ConfigMap`、 `ServiceAccount`、およびその他の VM 関連オブジェクト）のみをリストアします。

リストア CR で `includedVirtualMachines`を指定しない場合、Trident Protect はリストア CR の設定に従ってアーカイブ全体をリストアします。

サポートされている復元**CR**と**CLI**コマンド

以下の表は、 `includedVirtualMachines`をサポートするリストアCRと、それらに対応するCLIコマンドの一覧です。

| CRを復元する | CLIコマンド | 概要 |
| --- | --- | --- |
| BackupRestore | tridentctl-protect create backuprestore | バックアップから、新規または異なる名前空間マッピングに復元します。サポート namespaceMapping`および `storageClassMapping。 |
| BackupInplaceRestore | tridentctl-protect create backupinplacerestore | バックアップから元の名前空間または複数の名前空間に復元します。 |
| SnapshotRestore | tridentctl-protect create snapshotrestore | スナップショットから、新規または異なる名前空間マッピングに復元します。サポート namespaceMapping`および `storageClassMapping。 |
| SnapshotInplaceRestore | tridentctl-protect create snapshotinplacerestore | スナップショットから元の名前空間または複数の名前空間に復元します。 |
| ReplicateSnapshotRestore | YAMLのみを使用するか、お使いの環境でサポートされているCLIコマンドが利用可能な場合はそれを使用してください | レプリケートされたスナップショットから復元します。namespaceMapping、 storageClassMapping、および `inPlaceRestore`フラグを使用したIn Placeリストアをサポートします。 |

Trident CLIには、リストア コマンドに `includedVirtualMachines`用のフラグはありません。`tridentctl-protect`特定のVMをリストアするには、 `kubectl apply`を使用してリストアCR YAMLファイルを直接適用する必要があります。YAMLを一から作成するか、CLIで `--dry-run --output yaml`を使用して開始ファイルを生成し、 `spec`の下に `includedVirtualMachines`フィールドを追加してから適用します。

**VM**スコープのリストア動作

復元CRの `includedVirtualMachines`フィールドは、アプリケーションCRと同じセレクタ形式を使用します。

```
includedVirtualMachines:
  - namespace: <source_vm_namespace>
    names:
      - <vm_name_1>
      - <vm_name_2>
```

CR を復元する場合、 `includedVirtualMachines`の名前空間の値は、バックアップ、スナップショット、またはレプリケートされたスナップショット アーカイブに存在するソース名前空間を参照します。

使用する場合は namespaceMapping、Trident Protect はまずソースアーカイブから要求された仮想マシンを選択し、次にリストア中に名前空間マッピングを適用します。使用する場合は storageClassMapping 、Trident Protect は、リストア変換フェーズ中にストレージクラスのマッピングを適用します。

CLIを使用してリストア用YAMLを生成する

`tridentctl-protect`を使用してベースリストアCRを生成し、YAMLを編集して`includedVirtualMachines`を追加できます。

```
tridentctl-protect create <restore_command> <restore_name> \
  <restore_options> \
  --dry-run --output yaml > <restore_name>.yaml
```

生成されたYAMLを編集し、`spec`の下に`includedVirtualMachines`フィールドを追加してから、ファイルを適用します：

```
kubectl apply -f <restore_name>.yaml
```

例：VMベースのアプリケーションを定義する

以下のアプリケーションは、2つの名前空間にまたがる3つの特定の仮想マシンを保護します。

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: production-vms
  namespace: prod
spec:
  includedVirtualMachines:
    - namespace: prod
      names:
        - app-server
        - database-server
    - namespace: monitoring
      names:
        - prometheus-vm
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

例：バックアップを作成する

VMベースのアプリケーションのバックアップを作成するには、標準のバックアップCRを使用します。バックアップCRには、VM固有のフィールドは必要ありません。

```
apiVersion: protect.trident.netapp.io/v1
```

```
kind: Backup
metadata:
  name: daily-backup
  namespace: prod
spec:
  applicationRef: production-vms
  appVaultRef: s3-vault
```

例：バックアップから単一の仮想マシンをリストアする

次の例では、バックアップ アーカイブから `database-server`のみをリストアします。アーカイブ内の他のVM（ `app-server`や `prometheus-vm`など）はスキップされます。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: restore-db-only
  namespace: prod-vms-restore
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  namespaceMapping:
    - source: prod
      destination: prod-dr
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f restore-db-only.yaml
```

`database-server`とその依存リソース（DataVolumes、PVC、PV、Secrets、ConfigMaps、および関連するKubeVirtリソースなど）のみがリストアされます。

例：バックアップから単一の **VM**を**In Place**リストアする

次の例では、 `database-server`のみを元の名前空間にバックアップ アーカイブからリストアします。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: ipr-db-only
```

```
  namespace: prod
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f ipr-db-only.yaml
```

例：スナップショットから単一の仮想マシンを復元する

次の例では、スナップショット アーカイブから `ubuntu-vm-blue-a627be38`のみをリストアします。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: blue-vm-snap-restore
  namespace: blue
spec:
  appVaultRef: my-appvault
  appArchivePath: snapshots/protectctl-blue-vm/snap-1
  includedVirtualMachines:
    - namespace: blue
      names:
        - ubuntu-vm-blue-a627be38
```

```
kubectl apply -f blue-vm-snap-restore.yaml
```

**VM**リストアと他のフィルターを組み合わせる

`includedVirtualMachines`を `resourceFilter`、 `namespaceMapping`、および `storageClassMapping`と同じリストアCRで組み合わせて使用することで、リストア対象の内容を細かく調整できます。

追加すると `resourceFilter`、Trident Protect はまず `includedVirtualMachines`にリストされている VM とその依存リソースを選択します。次に、リソースフィルターを適用して、結果に含めるものと除外するものをさらに絞り込みます。

```
spec:
```

```
  includedVirtualMachines:
    - namespace: <vm_namespace>
      names:
        - <vm_name>
  resourceFilter:
    resourceSelectionCriteria: Include
    resourceMatchers:
      - kinds: ["PersistentVolumeClaim"]
        names: ["<pvc_name>"]
```

`BackupRestore`、 `SnapshotRestore`、および `ReplicateSnapshotRestore`の場合、 `namespaceMapping`と `storageClassMapping`は `includedVirtualMachines`と連携して動作します。 `includedVirtualMachines`の名前空間参照は、アーカイブに表示されているソース名前空間に対応します。名前空間とストレージクラスのマッピングは、フィルタリングが完了した後のリストア変換フェーズ中に適用されます。

復元動作の詳細については、"Trident Protectを使用してアプリケーションを復元する"を参照してください。

**CLI**を使用して**VM**スコープのアプリケーションを作成する

`--virtual-machines`フラグを使用して、名前空間全体ではなく特定のKubeVirt VMにアプリケーションのスコープを絞り込みます。 `--virtual-machines`フラグは `--namespaces`と相互に排他的です。

手順

1. 以下のコマンドのいずれかを実行し、括弧内の値を環境情報に置き換えてアプリケーションを作成してください。 `--virtual-machines`フラグのフォーマットは `<namespace>(<vm_name>)`です。複数の仮想マシンを指定する場合は、括弧内にカンマ区切りの名前を使用してください。
   - 名前空間内の単一の仮想マシン：

     ```
     tridentctl-protect create application <application_name> \
       -n <application_namespace> \
       --virtual-machines "<vm_namespace>(<vm_name>)"
     ```

   - 名前空間をまたがる複数の仮想マシン：

     ```
     tridentctl-protect create application <application_name> \
       -n <application_namespace> \
       --virtual-machines "<vm_namespace_1>(<vm_name_1>,<vm_name_2>
     ),<vm_namespace_2>(<vm_name_3>)"
     ```

関連情報

- "管理用のアプリケーションを定義する"
- "アプリケーションを保護する"
- "Trident Protectを使用してアプリケーションを復元する"
- "詳細リストア設定を使用する"
- "Trident Protect CLI プラグインをインストールします"
- "AppVaultオブジェクトを使用してバケットを管理します"

## 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、スナップショットが完了した後にトランザクションを再開できます。これにより、アプリケーションの一貫性のあるスナップショットが保証されます。

### 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。

- 事前スナップショット
- スナップショット後
- バックアップ前
- バックアップ後
- 復元後
- フェイルオーバー後

### 実行順序

データ保護操作が実行されると、実行フック イベントが次の順序で発生します：

1. 適用可能なカスタム事前操作実行フックは、該当するコンテナ上で実行されます。必要な数だけカスタム操作前フックを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されておらず、構成もできません。
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を使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください"。
3. データ保護操作が実行されます。
4. 該当する場合、凍結されたファイルシステムは凍結解除されます。
5. 適用可能なカスタム操作後実行フックは、適切なコンテナで実行されます。必要な数だけカスタム操作後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ておらず、構成もできません。

同じタイプ（たとえば、pre-snapshot）の実行フックを複数作成する場合、そ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ます。たとえば、さまざまな種類のフックがすべて含まれる構成の実行順序は次のとおりです：

1. スナップショット前のフックが実行されました
2. スナップショット後のフックが実行されました
3. バックアップ前のフックが実行されました
4. バックアップ後のフックが実行されました

上記の順序の例は、既存のスナップショットを使用しないバックアップを実行する場合にのみ適用されます。

本番環境で実行フックスクリプトを有効にする前に、必ず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後、結果のスナップショットとバックアップをテストして、一貫性があることを確認します。これを行うには、アプリを一時的な名前空間に複製し、スナップショットまたはバックアップを復元してから、アプリをテストします。

スナップショット前の実行フックによって Kubernetes リソースが追加、変更、または削除された場合、それらの変更はスナップショットまたはバックアップと、その後のすべてのリストア処理に含まれます。

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、アクションを実行するためにスクリプトを使用する必要があります。複数の実行フックが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、多くの場合、実行対象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ため、カスタム実行フックの実行にかかる時間を常に最小限に抑えるようにする必要があります。関連する実行フックを使用してバックアップまたは Snapshot 操作を開始したが、その後キャンセルした場合でも、バックアップまたは Snapshot 操作がすでに開始されている場合は、フックは引き続き実行できます。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では、バックアップが完了したと想定することはできません。

実行フックフィルター

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するアプリケーションで役立ちますが、各イメージを異なる目的で使用する場合があります（Elasticsearchなど）。フィルタを使用すると、実行フックを一部の同一コンテナでのみ実行し、必ずしもすべての同一コンテナで実行するとは限らないシナリオを作成できます。単一の実行フックに対して複数のフィルタを作成した場合、それらは

論理AND演算子で結合されます。実行フックごとに最大10個のアクティブなフィルタを設定できます。

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。フィルタ用の正規表現は正規表現2（RE2）構文を使用しますが、この構文ではコンテナを一致リストから除外するフィルタを作成することはできません。Trident Protect が実行フックフィルタの正規表現でサポートする構文については、 "正規表現2（RE2）構文のサポート"を参照してください。

リストアまたはクローン処理の後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

`https://github.com/NetApp/Verda["NetApp Verda GitHubプロジェクト "]`にアクセスして、`Apache Cassandra` や `Elasticsearch` などの一般的なアプリの実際の実行フックをダウンロードしてください。また、例を参照したり、独自のカスタム実行フックを構成するためのアイデアを入手したりすることもできます。

Verdaリポジトリのスクリプトは現状のまま提供されており、NetAppによる公式サポートの対象外です。NetAppのサポート対象は、Trident Protect内の実行フックフレームワークのみです。

実行フックを作成する

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**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-hook.yaml`という名前を付けます。
2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：
   - **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.applicationRef**：（必須）実行フックを実行するアプリケーションの Kubernetes 名。
   - **spec.stage**：（必須）アクション中に実行フックを実行するステージを示す文字列。可能な値：
     - プレ
     - 投稿
   - **spec.action**：（必須）指定された実行フックフィルタが一致した場合に、実行フックが実行するアクションを示す文字列。可能な値：
     - Snapshot
     - バックアップ

        - リストア
        - フェイルオーバー
    - **spec.enabled**：（オプション）この実行フックが有効か無効かを示します。指定がない場合、デフォルト値はtrueです。
    - **spec.hookSource**：（必須）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**：（オプション）実行フックの実行が許可される時間（分）を定義する数値。最小値は 1 分で、指定されていない場合のデフォルト値は 25 分です。
    - **spec.arguments**: （オプション）実行フックに指定できる引数の YAML リスト。
    - **spec.matchingCriteria**：（オプション）条件キーと値のペアのオプションのリスト。各ペアは実行フック フィルターを構成します。実行フックごとに最大 10 個のフィルターを追加できます。
    - **spec.matchingCriteria.type**：（オプション）実行フックのフィルタータイプを識別する文字列。可能な値：
        - ContainerImage
        - ContainerName
        - PodName
        - PodLabel
        - NamespaceName
    - **spec.matchingCriteria.value**：（オプション）実行フックのフィルター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: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
```

```
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. 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** を使用する

手順

1. 実行フックを作成し、括弧内の値を環境の情報に置き換えます。次に例を示します。

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

実行フックを手動で実行する

テスト目的で、または失敗後にフックを手動で再実行する必要がある場合は、実行フックを手動で実行できます。実行フックを手動で実行するには、Owner、Admin、またはMemberの権限が必要です。

実行フックを手動で実行するには、次の 2 つの基本的な手順を実行します：

1. リソースのバックアップを作成します。これはリソースを収集してそれらのバックアップを作成し、フックが実行される場所を決定します
2. バックアップに対して実行フックを実行する

▼ ステップ**1**：リソースのバックアップを作成する

**CR**を使用する

手順

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

2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：

   - **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.applicationRef**：（必須）リソース バックアップを作成するアプリケーションの Kubernetes 名。
   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

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

     YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ResourceBackup
metadata:
  name: example-resource-backup
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
```

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-resource-backup.yaml
```

**CLI** を使用する

手順

1. 括弧内の値を環境の情報に置き換えてバックアップを作成します。次に例を示します。

```
tridentctl protect create resourcebackup <my_backup_name> --app
<my_app_name> --appvault <my_appvault_name> -n <my_app_namespace>
--app-archive-path <app_archive_path>
```

2. バックアップのステータスを表示します。操作が完了するまで、この例のコマンドを繰り返し使用できます：

```
tridentctl protect get resourcebackup -n <my_app_namespace>
```

```
<my_backup_name>
```

3. バックアップが成功したことを確認します：

```
kubectl describe resourcebackup <my_backup_name>
```

▼ ステップ**2**：実行フックを実行する

**CR**を使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-hook-run.yaml`という名前を付けます。
2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：
   - **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.applicationRef**：（必須）この値が、手順 1 で作成した ResourceBackup CR のアプリケーション名と一致していることを確認してください。
   - **spec.appVaultRef**：（必須）この値が、手順1で作成したResourceBackup CRのappVaultRefと一致していることを確認してください。
   - **spec.appArchivePath**：この値が、手順1で作成したResourceBackup CRのappArchivePathと一致していることを確認してください。

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

   - **spec.action**：（必須）指定された実行フックフィルタが一致した場合に、実行フックが実行するアクションを示す文字列。可能な値：
     - Snapshot
     - バックアップ
     - リストア
     - フェイルオーバー
   - **spec.stage**：（必須）アクション中に実行フックを実行するステージを示す文字列。このフック実行では、他のステージのフックは実行されません。可能な値：
     - プレ
     - 投稿

     YAMLの例：

```
---
```

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

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-run.yaml
```

**CLI** を使用する

手順

1. 手動実行フックの実行リクエストを作成します：

```
tridentctl protect create exechooksrun <my_exec_hook_run_name> -n
<my_app_namespace> --action snapshot --stage <pre_or_post> --app
<my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. 実行フックの実行ステータスを確認します。操作が完了するまでこのコマンドを繰り返し実行できます：

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. 最終的な詳細とステータスを確認するには、exechooksrun オブジェクトを記述します：

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

# 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かもしれません。

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：

   バージョン 25.06 以降ではこの手順は必要ありません。

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

2. Trident Protectを削除：

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

3. Trident Protect ネームスペースを削除します：

   ```
   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権を有し、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権については、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。