Trident Protectでアプリケーションを保護する

Trident

NetApp
January 15, 2026

目次

Trident Protectでアプリケーションを保護する

Tridentプロテクトについて学ぶ

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

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

次の手順

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

- "Tridentプロテクトの要件"

Trident Protectをインストールする

Tridentプロテクトの要件

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

Trident Protect Kubernetes クラスターの互換性

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

- Amazon Elastic Kubernetes サービス (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes サービス (AKS)
- レッドハット オープンシフト
- SUSE ランチャー
- 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ーに、実行中のスナップショット コントローラーと関連する CRD が設定されていることを確認します。スナップショットコントローラをインストールするには、 "これらの指示" 。

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。

- クラスターに接続されている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ます。
- 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 はこれらの名前空間を作成しません。カスタム リソースに存在しない名前空間を指定すると、操作は失敗します。

NASエコノミーボリュームの要件

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

一部のアプリケーションは、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>`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
```

Tridentバックエンド設定オプションを設定することで、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ことができます。 `snapshotDir`に `true`。既存のボリュームは影響を受けません。

KubeVirt VM によるデータ保護

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

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

Tridentプロテクト **24.10**

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

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

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

Trident Protect 24.10.1以降

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

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

SnapMirrorレプリケーションの要件

NetApp SnapMirrorレプリケーションは、次のONTAPソリューションのTrident Protect で使用できます。

- オンプレミスのNetApp FAS、 AFF、 ASAクラスター
- NetApp ONTAP Select
- NetAppCloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

SnapMirrorレプリケーションのONTAPクラスタ要件

SnapMirrorレプリケーションを使用する予定の場合は、 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使用するソース Kubernetes クラスターと宛先 Kubernetes クラスターの両方に存在する必要があります。Trident Protect は、次のドライバーによってサポートされるストレージ クラスを使用して、 NetApp SnapMirrorテクノロジ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 (最低でもONTAPバージョン 9.15.1 が必要)
- ライセンス: データ保護バンドルを使用するONTAP SnapMirror非同期ライセンスは、ソースと宛先の両方のONTAPクラスタで有効にする必要があります。参照 "ONTAPにおけるSnapMirrorライセンスの概要"詳細についてはこちらをご覧ください。

  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。参照"ONTAP Oneに含まれるライセンス"詳細についてはこちらをご覧ください。

  SnapMirror非同期保護のみがサポートされます。

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 バックエンドをピアリングする必要があります。参照 "クラスタとSVMのピアリングの概要"詳細についてはこちらをご覧ください。

  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ます。

- * NetApp Tridentと SVM*: ピアリングされたリモート SVM は、宛先クラスタ上のNetApp Tridentで使用できる必要があります。
- 管理対象バックエンド: レプリケーション関係を作成するには、 Trident Protect でONTAPストレージ バックエンドを追加および管理する必要があります。

SnapMirrorレプリケーションのためのTrident / ONTAP構成

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

SnapMirrorレプリケーションのKubernetesクラスタ要件

Kubernetes クラスターが次の要件を満たしていることを確認します。

- **AppVault** のアクセシビリティ: アプリケーション オブジェクトのレプリケーションでは、ソース クラスターと宛先クラスターの両方に、AppVault の読み取りと書き込みを行うためのネットワーク アクセスが必要です。
- 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 許可リストを構成して、WAN を介した両方のクラスターと AppVault 間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構成する前に、インフラストラクチャ チームとこれらのネットワーク要件を確認してください。

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

Trident Protectをインストールする

NetAppからTrident Protectをインストールする

手順

1. Trident Helm リポジトリを追加します。

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

2. Helm を使用してTrident Protect をインストールします。交換する `<name-of-cluster>`クラスター名。このクラスター名はクラスターに割り当てられ、クラスターのバックアップとスナップショットを識別するために使用されます。

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2506.0 --create
-namespace --namespace trident-protect
```

プライベートレジストリから**Trident Protect**をインストールする

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

手順

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

```
netapp/controller:25.06.0
netapp/restic:25.06.0
netapp/kopia:25.06.0
netapp/trident-autosupport:25.06.0
netapp/exechook:25.06.0
netapp/resourcebackup:25.06.0
netapp/resourcerestore:25.06.0
netapp/resourcedelete:25.06.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例えば：

```
docker pull netapp/controller:25.06.0
```

```
docker tag netapp/controller:25.06.0 <private-registry-
url>/controller:25.06.0
```

```
docker push <private-registry-url>/controller:25.06.0
```

2. Trident Protect システム名前空間を作成します。

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

3. レジストリにログインします。

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

4. プライベート レジストリ認証に使用するプル シークレットを作成します。

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

5. Trident Helm リポジトリを追加します。

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

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

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

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

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2506.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

Trident Protect CLIプラグインをインストールする

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

Trident Protect CLIプラグインをインストールする

コマンドライン ユーティリティを使用する前に、クラスタ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 またはARM CPU を使用しているかどうかに応じて、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/25.06.0/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/25.06.0/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/25.06.0/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/25.06.0/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、次のように入力してプラグインを使用できるようになります。 `tridentctl-protect`または `tridentctl protect`どこからでも。

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後、特定のコマンドの自動補完を有効にすることができます。

Bashシェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/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：

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

4. 次の行を `~/.bashrc`ホームディレクトリ内のファイル:

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

Z シェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/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：

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

4. 次の行を `~/.zprofile`ホームディレクトリ内のファイル:

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

手順

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

   次の例の構成ファイルは、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. 値を適用する `resourceLimits.yaml`ファイル：

```
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 をインストールした後、構成ファイルを使用して、 Trident Protect コンテナーの 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 クラスター内のポッドのセキュリティ制限を定義します。

手順

1. という名前のファイルを作成します `sccconfig.yaml`。
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	説明	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リソースは次の場合にのみ作成されます。 `scc.create`設定されている `true`Helm インストール プロセスでは OpenShift 環境が識別されます。 OpenShift上で動作していない場合、または `scc.create`設定されている `false`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。	トライデントプロテクトジョブ
優先度	SCC の優先度を定義します。優先度の高い SCC は、優先度の低い SCC よりも先に評価されます。	1

3. の値を適用します `sccconfig.yaml`ファイル：

```
helm upgrade 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。

追加の**Trident Protect**ヘルムチャート設定を構成する

特定の要件に合わせて、 AutoSupport設定と名前空間フィルタリングをカスタマイズできます。次の表は、使用可能な構成パラメータを示しています。

パラメータ	タイプ	説明
自動サポートプロキシ	string	NetApp AutoSupport接続用のプロキシ URL を構成します。これを使用して、サポート バンドルのアップロードをプロキシ サーバー経由でルーティングします。例： `http://my.proxy.url` 。

パラメータ	タイプ	説明
autoSupport.insecure	ブーリアン	設定すると、 AutoSupportプロキシ接続のTLS検証をスキップします。 `true` 。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` ）
自動サポートが有効	ブーリアン	毎日のTrident Protect AutoSupportバンドルのアップロードを有効または無効にします。に設定すると `false`、スケジュールされた毎日のアップロードは無効になっていますが、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` ）
スキップ名前空間注釈の復元	string	バックアップおよび復元操作から除外する名前空間注釈のコンマ区切りリスト。注釈に基づいて名前空間をフィルタリングできます。
スキップ名前空間ラベルの復元	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。

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 フラグを使用して構成できます。

YAMLファイルを使用する

手順

1. 設定ファイルを作成し、名前を付けます `values.yaml`。
2. 作成したファイルに、カスタマイズする構成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"
```

3. 入力したら `values.yaml`正しい値を持つファイルの場合は、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. 次のコマンドを `--set`個々のパラメータを指定するためのフラグ:

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

Trident Protectポッドを特定のノードに制限する

Kubernetes nodeSelector ノード選択制約を使用すると、ノード ラベルに基づいて、どのノードがTrident Protect ポッドを実行できるかを制御できます。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ています。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. という名前のファイルを作成します `nodeSelectorConfig.yaml`。
2. ファイルに nodeSelector オプションを追加し、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変更します。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. 値を適用する `nodeSelectorConfig.yaml`ファイル：

```
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。

Trident Protectの管理

Trident Protectの認証とアクセス制御を管理する

Trident Protect は、ロールベースのアクセス制御 (RBAC) の Kubernetes モデルを使用します。デフォルトでは、 Trident Protect は単一のシステム名前空間とそれに関連付けられたデフォルトのサービス アカウントを提供します。組織内に多数のユーザーや特定のセキュリティ ニーズがある場合は、 Trident Protect の RBAC 機能を使用して、リソースや名前空間へのアクセスをより細かく制御できます。

クラスタ管理者は常にデフォルトのリソースにアクセスできます。 `trident-protect`名前空間だけでなく、他のすべての名前空間のリソースにもアクセスできます。リソースとアプリケーションへのアクセスを制御するには、追加の名前空間を作成し、それらの名前空間にリソースとアプリケーションを追加する必要があります。

デフォルトでは、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。 `trident-protect`名前空間。アプリケーション名前空間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名前空間にアプリケーション データ管理 CR を作成します)。

> 特権のあるTrident Protect カスタム リソース オブジェクトには、管理者のみがアクセスできる必要があります。これには次のものが含まれます。
>
> - **AppVault**: バケット認証情報データが必要
> - **AutoSupportBundle**: メトリック、ログ、その他の機密性の高いTrident Protect データを収集します
> - **AutoSupportBundleSchedule**: ログ収集スケジュールを管理します
>
> 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" 。

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" 。

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、組織にはクラスター管理者、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。クラスター管理者は、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 グループの名前空間を作成します。

```
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

作成する新しい名前空間にはそれぞれデフォルトのサービス アカウントが付属しますが、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. マーケティング サービス アカウントのシークレットを作成します。

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

ステップ**4: 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するための**RoleBinding**オブジェクトを作成する

Trident Protect をインストールすると、デフォルトの ClusterRole オブジェクトが作成されます。RoleBinding オブジェクトを作成して適用することで、この ClusterRole をサービス アカウントにバインドできます。

手順

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

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

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

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

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、クラスター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. ユーザーに AppVault へのアクセスを許可する AppVault とシークレットの組み合わせ YAML ファイルを作成して適用します。たとえば、次のCRはユーザーにAppVaultへのアクセスを許可します。 `eng-user` ：

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

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

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

3. RoleBinding CR を作成して適用し、権限をユーザー eng-user にバインドします。例えば：

```
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 オブジェクト情報を取得しようとします。

   ```
   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. ユーザーが現在アクセス権限を持っている 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、アプリケーション データ管理操作のために承認された AppVault オブジェクトを使用でき、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リソースを監視する

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 によって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 と併用すると、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する方法が提供されます。

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 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、Promethus、および Alertmanager をインストールして構成する必要があります。

kube-state-metricsをインストールする

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

手順

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

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

2. Prometheus ServiceMonitor CRD をクラスターに適用します。

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

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

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

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

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

# Collect metrics from all namespaces
namespaces: ""

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

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

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

5. kube-state-metricsを設定して、 Trident Protectが使用するカスタムリソースのメトリックを生成するには、以下の手順に従ってください。 "kube-state-metrics カスタムリソースのドキュメント" 。

Prometheusをインストールする

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

Alertmanagerをインストールする

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

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設定ファイルを編集する(`prometheus.yaml`) をクリックし、kube-state-metrics サービス情報を追加します。例えば：

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設定ファイルを編集する(`prometheus.yaml`) に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構成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、アラートを送信する場所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に設定されたときにトリガーされる重要なアラートを定義します。 `Error` 5秒以上。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:

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は、電子メール、PagerDuty、Microsoft Teams、その他の通知サービスなどの他のチャネルに通知を送信するように設定できます。 `alertmanager.yaml`ファイル。

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`お使いの環境で使用されている Slack Webhook URL にキーを設定します。

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

CRを使用し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. 次の属性を構成します。
   - **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、スケジュールに従って生成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 バンドルを生成後にNetAppサポート サイトにアップロードするかどうかを制御します。指定しない場合は、デフォルトは `false`。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が開始する日時を指定する RFC 3339 形式の日付文字列。指定しない場合は、デフォルトで 24 時間前になります。指定できる最も早いウィンドウ日付は 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. 入力したら `trident-protect-support-bundle.yaml`正しい値を持つファイルには、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。その `trigger-type`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。デフォルト設定は Manual。

   例えば：

   ```
   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. 待つ `status.generationState`到達する `Completed`州。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
   ```

   見つける `kubectl cp`出力からコマンドを取得して実行し、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プロテクトのアップグレード

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

> 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。

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

手順

1. Trident Helm リポジトリを更新します。

```
helm repo update
```

2. Trident Protect CRD をアップグレードします。

> 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 チャートに含まれるようになったため、この手順は必須です。

a. このコマンドを実行すると、CRDの管理を `trident-protect-crds`に `trident-protect`:

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

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

> アンインストールしないでください `trident-protect-crds`Helm を使用してチャートを作成しないでください。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プロテクトのアップグレード:

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

アプリケーションの管理と保護

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

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

アプリケーションに対してデータ保護操作を実行する際は、AppVault CR を手動で作成するか、コマンドラインから作成する必要があります。AppVaultCR は環境によって異なりますので、このページの例を参考にして AppVault CR を作成してください。

AppVault CR がTrident Protect がインストールされているクラスター上にあることを確認します。CRが存在しない場合、または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の認証とパスワードを設定する

AppVault CR を作成する前に、選択した AppVault とデータ ムーバーがプロバイダーおよび関連リソースに対して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 または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ときに、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。秘密を指定しない場合、 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`秘密を指定するための引数。

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ときに、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Trident Protect 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入力する必要があります。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にデータを復元する際に必要になります。クラスターまたは `trident-protect`名前空間が削除されると、パスワードなしでバックアップやスナップショットを復元できなくなります。

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

Amazon S3、Generic S3などのS3互換ストレージにアクセスする場合、"ストレージグリッドS3" 、 または "ONTAP S3"Trident Protect を使用する場合は、提供したユーザー認証情報にバケットにアクセスするために必要な権限があることを確認する必要があります。以下は、 Trident Protect によるアクセスに必要な最小限の権限を付与するポリシーの例です。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ーに適用できます。

```
{
  "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" 。

Amazon S3 (AWS) 認証用の EKS ポッド ID

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 クラスターで 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"。
- 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`

ポッドアイデンティティを有効にし、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 ポッドアイデンティティのドキュメント" 。

AppVault 構成 EKS Pod Identity を使用する場合は、AppVault CR を次のように構成します。 `useIAM: true` 明示的な資格情報の代わりにフラグを使用します:

```
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
```

クラウドプロバイダー向けの AppVault キー生成の例

AppVault CR を定義するときは、IAM 認証を使用していない限り、プロバイダー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資格情報を含める必要があります。資格情報のキーを生成する方法はプロバイダーによって異なります。以下は、いくつかのプロバイダーのコマンド ライン キー生成の例です。次の例を使用して、各クラウド プロバイダーの資格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
```

ストレージグリッド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

次の CR 例を使用して、各クラウド プロバイダーの 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。参照[データムーバーリポジトリのパスワード] 詳細についてはこちらをご覧ください。
- 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、プロバイダーのキーを生成するときに作成されます。クラウドプロバイダー向けの AppVault キー生成の例 。
- S3 AppVaultオブジェクトの場合、オプションで、アウトバウンドS3トラフィックの出力プロキシURLを次のように指定できます。 `spec.providerConfig.S3.proxyURL`鍵。

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を使用するEKS環境では、 `providerCredentials`セクションを追加 `useIAM: true`の下で `s3`代わりに構成してください。

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

```yaml
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
```

ストレージグリッド**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。参照[データムーバーリポジトリのパスワード] 詳細についてはこちらをご覧ください。
- S3 AppVaultオブジェクトの場合、オプションで、アウトバウンドS3トラフィックの出力プロキシURLを次のように指定できます。 `--proxy-url <ip_address:port>`口論。

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
```

ストレージグリッド**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
```

AppVault情報を表示する

Trident Protect CLI プラグインを使用して、クラスター上に作成した 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. オプションとして、各リソースのAppVaultPathを表示するには、フラグを使用します。 `--show-paths` 。

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

AppVaultを削除する

AppVault オブジェクトはいつでも削除できます。

> 取り外さないでください `finalizers`AppVault オブジェクトを削除する前に、AppVault CR にキーを追加します。これを実行すると、AppVault バケットにデータが残り、クラスター内に孤立したリソースが存在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およびバックアップ CR が削除されていることを確認します。

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、 `appvault-name`削除する AppVault オブジェクトの名前:

```
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、 `appvault-name`削除する AppVault オブジェクトの名前:

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

Trident Protectで管理するアプリケーションを定義する

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

AppVault CRを作成する

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

アプリケーションを定義する

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。アプリケーション CR を手動で作成するか、 Trident Protect CLI を使用して、管理対象のアプリケーションを定義できます。

CRを使用してアプリケーションを追加する

手順

1. 宛先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. 次の属性を構成します。
      - **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) 名前空間とラベル セレクターを使用して、アプリケーションが使用する名前空間とリソースを指定します。アプリケーション名前空間はこのリストの一部である必要があります。ラベル セレクターはオプションであり、指定された各名前空間内のリソースをフィルター処理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、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 KubeVirt 環境などの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。このアプリケーションが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。指定されていても、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、注釈は無視されます。指定されていない場合は、アプリケーションは"グローバルTrident Protectフリーズ設定"。

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) 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かを指定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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。

       - **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

> 両方が `resourceFilter`そして `labelSelector`使用される、 `resourceFilter`最初に実行し、その後 `labelSelector`結果のリソースに適用されます。

例えば：

```
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"]
```

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

```
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、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 KubeVirt 環境などの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`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。設定すると `false`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、アプリケーションのアプリケーション定義に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フリーズ設定"。

CLIを使用してアプリケーションを作成するときにアノテーションを指定するには、 `--annotation`フラグ。

- 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の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のローカル 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">
```

使用できます `--resource-filter-include`そして `--resource-filter-exclude`リソースを含めるか除外するかのフラグ `resourceSelectionCriteria`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、名前空間などです。

```
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"]}]'
```

Trident Protectを使用してアプリケーションを保護する

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

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

オンデマンドスナップショットを作成する

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。

> クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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、名前を付けます `trident-protect-snapshot-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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 に何が起こるかを定義します。つまり、 `Retain`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. 入力したら `trident-protect-snapshot-cr.yaml`正しい値を持つファイルには、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。例えば：

```
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。

> クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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 バックアップ操作に十分であることを確認します。バックアップ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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-backup-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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) バックアップが要求から解放されたとき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、一時的なスナップショットが作成され、バックアップされます。
   - **metadata.annotations.protect.trident.netapp.io/full-backup**: (オプション) この注釈は、バックアップを非増分にするかどうかを指定するために使用されます。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。ただし、この注釈が `true`、バックアップは非増分になります。指定しない場合は、バックアップはデフォルトの増分バックアップ設定に従います。復元に伴うリスクを最小限に抑えるために、定期的に完全バックアップを実行し、完全バックアップの間に増分バックアップを実行するのがベストプラクティスです。

     YAMLの例:

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

3. 入力したら `trident-protect-backup-cr.yaml`正しい値を持つファイルには、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。例えば：

```
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`バックアップを非増分にするかどうかを指定するフラグ。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。復元に伴うリスクを最小限に抑えるために、定期的に完全バックアップを実行し、完全バックアップの間に増分バックアップを実行するのがベストプラクティスです。

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することによっ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、増分以外の完全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。定期的に完全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すると、復元に関連するリスクを軽減できます。

- スナップショットのスケジュールを作成するには、以下を設定します。 `backupRetention`ゼロにし、 `snapshotRetention`ゼロより大きい値に設定します。設定 `snapshotRetention`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スナップショットが作成されますが、それらは一時的なものであり、バックアップが完了するとすぐに削除されます。
- クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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、名前を付けます `trident-protect-schedule-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: 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.snapshotRetention**: 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.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`（指定する必要があります `spec.minute`）
     - `Daily`（指定する必要があります `spec.minute`そして `spec.hour`）
     - `Weekly`（指定する必要があります `spec.minute, spec.hour`、 そして `spec.dayOfWeek`）
     - `Monthly`（指定する必要があります `spec.minute, spec.hour`、 そして `spec.dayOfMonth`）
     - `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「」に設定されている場合、このフィールドは必須です。 `Monthly` 。値は文字列として提供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「」に設定されている場合、このフィールドは必須です。 `Weekly` 。値は文字列として提供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「」に設定されている場合、このフィールドは必須です。 `Daily` 、 `Weekly` 、 または `Monthly`。値は文字列として提供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分 (0 - 59)。粒度が「」に設定されている場合、このフィールドは必須です。 `Hourly` 、 `Daily` 、 `Weekly` 、 または `Monthly`。値は文字列として提供する必要があります。
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (オプション) このアノテーションは、完全バックアップをスケジュールするためのルールを指定するために使用されます。設定できるのは `always`継続的な完全バックアップを実現するか、要件に応じてカスタマイズしま

す。たとえば、毎日の粒度を選択した場合は、完全バックアップを実行する曜日を指定できます。

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
  annotations:
    protect.trident.netapp.io/full-backup-rule: "Monday,Thursday"
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"
```

3. 入力したら `trident-protect-schedule-cr.yaml`正しい値を持つファイルには、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。例えば：

   使用できます `tridentctl-protect create schedule --help`このコマンドの詳細なヘルプ情報を表示します。

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

   設定できるのは `--full-backup-rule`フラグを `always`継続的な完全バックアップを実現するか、要件に応じてカスタマイズします。たとえば、毎日の粒度を選択した場合は、完全バックアップを実行する曜日を指定できます。例えば、 `--full-backup-rule "Monday,Thursday"`月曜日と木曜日に完全バックアップをスケジュールします。

   スナップショットのみのスケジュールの場合は、 `--backup-retention 0` 0より大きい値を指定する `--snapshot-retention`。

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された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されたバックアップまたはオンデマンドのバックアップを削除します。

回収ポリシーが設定されていることを確認する `Delete`オブジェクト ストレージからすべてのバックアップ データを削除します。ポリシーのデフォルト設定は `Retain`偶発的なデータ損失を回避するため。政策が変更されない場合 `Delete`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復元を有効にする

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

開始する前に

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。
- あなたが持っている `azure-netapp-files`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 にアップグレードする前に作成された場合は、 Tridentで次の操作を実行します。

   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は、通常のバックアップ機能が選択され、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、バックアップ対象のボリュームのサイズの一時ボリュームを作成するために十分なスペースが容量プールにあることを確認してください。

結果

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

アプリケーションを復元する

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。復元後の実行フックが存在する場合、復元操作の一部として自動的に実行されます。
- qtree ボリュームでは、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がサポートされています。ただし、qtree ボリュームでは、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復元設定を使用する"。

バックアップから別の名前空間に復元する

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

既存のリソースを含む別の名前空間にバックアップを復元しても、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復元するには、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復元します。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンド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、名前を付けます `trident-protect-backup-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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.namespaceMapping**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

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

3. (オプション) 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。

   Trident Protect は、選択したリソースとの関係に基づいて、いくつかのリソースを自動的に選択します。たとえば、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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。

- **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-backup-restore-cr.yaml`正しい値を持つファイルには、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。その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。例えば：

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

バックアップから元の名前空間に復元する

いつでもバックアップを元の名前空間に復元できます。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンド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、名前を付けます `trident-protect-backup-ipr-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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。

     例えば：

```
---
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) 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。

   Trident Protect は、選択したリソースとの関係に基づいて、いくつかのリソースを自動的に選択します。たとえば、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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。
       - **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-backup-ipr-cr.yaml`正しい値を持つファイルには、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。その `backup` 引数は、名前空間とバックアップ名を次の形式で使用します。 `<namespace>/<name>` 。例えば：

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

元のクラスターに問題がある場合は、バックアップを別のクラスターに復元できます。

> 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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンドを参照してください。

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。

- 宛先クラスターにはTrident Protect がインストールされています。
- 宛先クラスターは、バックアップが保存されているソース クラスターと同じ AppVault のバケット パスにアクセスできます。
- 実行時に、ローカル環境が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。 `tridentctl-protect get appvaultcontent`指示。ネットワーク制限によりアクセスできない場合は、代わりに宛先クラスターのポッド内からTrident Protect CLI を実行します。
- 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。
  - 参照 "AWS API ドキュメント"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、こちらをご覧ください。
  - 参照 "AWSのドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

手順

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

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

> アプリケーションの復元を目的とした名前空間が宛先クラスターに存在することを確認します。

2. 宛先クラスターから利用可能な AppVault のバックアップ コンテンツを表示します。

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

このコマンドを実行すると、AppVault 内の使用可能なバックアップ（元のクラスター、対応するアプリケーション名、タイムスタンプ、アーカイブ パスなど）が表示されます。

出力例:

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

3. AppVault 名とアーカイブ パスを使用して、アプリケーションを宛先クラスターに復元します。

CRを使用する

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 `trident-protect-backup-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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}'
     ```

     BackupRestore CR が使用できない場合は、手順 2 に記載されているコマンドを使用してバックアップの内容を表示できます。

   - **spec.namespaceMapping**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     例えば：

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

3. 入力したら `trident-protect-backup-restore-cr.yaml`正しい値を持つファイルには、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。例えば：

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

スナップショットから別の名前空間に復元する

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

SnapshotRestoreは、 `spec.storageClassMapping`属性ですが、ソース ストレージ クラスと宛先ストレージ 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。復元しようとすると、 `StorageClass`異なるストレージ バックエンドを使用する場合、復元操作は失敗します。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-snapshot-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.namespaceMapping**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

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

3. (オプション) 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。

   Trident Protect は、選択したリソースとの関係に基づいて、いくつかのリソースを自動的に選択します。たとえば、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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。

- **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-restore-cr.yaml`正しい値を持つファイルには、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名前空間に復元します。
   - 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>` 。
   - その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。

例えば：

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

スナップショットから元の名前空間に復元する

いつでもスナップショットを元の名前空間に復元できます。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-snapshot-ipr-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

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

3. (オプション) 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。

> Trident Protect は、選択したリソースとの関係に基づいて、いくつかのリソースを自動的に選択します。たとえば、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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。
       - **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-ipr-cr.yaml`正しい値を持つファイルには、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。例えば：

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

復元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、または失敗した復元操作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**復元設定を使用する

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。

復元およびフェイルオーバー操作中の名前空間の注釈とラベル

復元およびフェイルオーバー操作中に、宛先名前空間のラベルと注釈は、ソース名前空間のラベルと注釈と一致するように作成されます。宛先名前空間に存在しないソース名前空間のラベルまたは注釈が追加され、既存のラベルまたは注釈はソース名前空間の値と一致するように上書きされます。宛先名前空間にのみ存在するラベルまたは注釈は変更されません。

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、復元されたポッドが OpenShift のセキュリティ コンテキスト制約 (SCC) によって定義された適切な権限とセキュリティ構成に準拠し、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 コンテキスト制約のドキュメント" 。

Kubernetes環境変数を設定することで、宛先名前空間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。 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`復元またはフェイルオーバー操作を実行する前に。例えば：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

復元またはフェイルオーバー操作を実行する場合、 `restoreSkipNamespaceAnnotations` そして `restoreSkipNamespaceLabels` 復元またはフェイルオーバー操作から除外されます。これらの設定が Helm の初期インストール時に構成されていることを確認してください。詳細については、 "AutoSupportと名前空間フィルタリングオプションを構成する"。

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

例

次の例は、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。

復元またはフェイルオーバー操作の前に

次の表は、復元またはフェイルオーバー操作前のサンプルのソース名前空間と宛先名前空間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
名前空間 ns-1 (ソース)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true"	• 環境=本番環境 • コンプライアンス=HIPAA • 名前=ns-1
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "true" • 注釈.three/キー: "false"	• 役割=データベース

復元操作後

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
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true" • 注釈.three/キー: "false"	• 名前=ns-2 • コンプライアンス=HIPAA • 環境=本番環境 • 役割=データベース

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

その `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 クラスター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。これは、異なるストレージ クラスを持つクラスター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注釈

このセクションでは、システム内のさまざまな動作を構成するためにサポートされているアノテーションを一覧表示します。ユーザーが注釈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	説明	デフォルト値
protect.trident.netapp.io/データムーバータイムアウト秒	string	データ ムーバー操作を停止できる最大時間 (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」

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

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

復元およびフェイルオーバー操作中の名前空間の注釈とラベル

復元およびフェイルオーバー操作中に、宛先名前空間のラベルと注釈は、ソース名前空間のラベルと注釈と一致するように作成されます。宛先名前空間に存在しないソース名前空間のラベルまたは注釈が追加され、既存のラベルまたは注釈はソース名前空間の値と一致するように上書きされます。宛先名前空間にのみ存在するラベルまたは注釈は変更されません。

> 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、復元されたポッドが OpenShift のセキュリティ コンテキスト制約 (SCC) によって定義された適切な権限とセキュリティ構成に準拠し、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 コンテキスト制約のドキュメント" 。

Kubernetes環境変数を設定することで、宛先名前空間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。 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`復元またはフェイルオーバー操作を実行する前に。例えば：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

復元またはフェイルオーバー操作を実行する場合、 `restoreSkipNamespaceAnnotations` そして `restoreSkipNamespaceLabels` 復元またはフェイルオーバー操作から除外されます。これらの設定が Helm の初期インストール時に構成されていることを確認してください。詳細については、 "AutoSupportと名前空間フィルタリングオプションを構成する"。

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

例

次の例は、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。

復元またはフェイルオーバー操作の前に

次の表は、復元またはフェイルオーバー操作前のサンプルのソース名前空間と宛先名前空間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
名前空間 ns-1 (ソース)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true"	• 環境=本番環境 • コンプライアンス=HIPAA • 名前=ns-1
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "true" • 注釈.three/キー: "false"	• 役割=データベース

復元操作後

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
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true" • 注釈.three/キー: "false"	• 名前=ns-2 • コンプライアンス=HIPAA • 環境=本番環境 • 役割=データベース

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

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注意する必要がある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 クラスターから宛先クラスター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宛先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、以下を参照してください。"Trident Protect実行フックを管理する" 。

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

手順

1. ソース クラスターで、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、以下の例を変更します。"AppVault カスタムリソース"あなたの環境に合わせて:

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. 次の属性を構成します。

- **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.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構成を保存します。プロバイダーの bucketName とその他の必要な詳細を選択します。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。参照"AppVault カスタムリソース"他のプロバイダーとの AppVault CR の例。
- **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資格情報への参照を格納します。
  - **spec.providerCredentials.valueFromSecret**: (必須) 資格情報の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。たとえば、 NetApp ONTAP S3、汎用 S3、Google Cloud、Microsoft Azure などです。有効な値は次のとおりです。
  - AWS
  - 紺碧
  - gcp
  - ジェネリックS3
  - オンタップS3
  - ストレージグリッドS3

c. 入力したら `trident-protect-appvault-primary-source.yaml`正しい値を持つファイルには、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. 括弧内の値を環境の情報に置き換えて、AppVault を作成します。

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

2. ソース クラスターで、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. 次の属性を構成します。

- **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) 名前空間と関連ラベルの配列。ここにリストされている名前空間内に存在するリソースを指定するには、名前空間名を使用し、オプションでラベルを使用して名前空間の範囲を絞り込みます。アプリケーション名前空間はこの配列の一部である必要があります。

  YAMLの例:

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

c. 入力したら `trident-protect-app-source.yaml`正しい値を持つファイルには、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。例えば：

```
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、宛先クラスター上のアプリケーションのベースとして使用されます。この手順を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"。

4. ソース クラスターで、レプリケーション スケジュール CR を作成します。

以下に示すスケジュールに加えて、ピア接続された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、保持期間が 7 日間の個別の毎日のスナップショット 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、保持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。この方法では、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. 次の属性を構成します。

   - **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**: に設定する必要があります `Custom`。
   - **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. 入力したら `trident-protect-schedule.yaml`正しい値を持つファイルには、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. 宛先クラスタで、ソースクラスタに適用したAppVault CRと同一の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. 宛先クラスター上の宛先アプリケーション用の宛先 AppVault CR を作成します。ストレージプロバイダに応じて、以下の例を変更します。"AppVault カスタムリソース"あなたの環境に合わせて:
   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. 次の属性を構成します。
      - **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.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構成を保存します。選択してください `bucketName`およびプロバイダーに関するその他の必要な詳細情報。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。参照"AppVault カスタムリソース"他のプロバイダーとの AppVault CR の例。
      - **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資格情報への参照を格納します。
        - **spec.providerCredentials.valueFromSecret**: (必須) 資格情報の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。たとえば、 NetApp ONTAP S3、汎用 S3、Google Cloud、Microsoft Azure などです。有効な値は次のとおりです。
      - AWS
      - 紺碧
      - gcp
      - ジェネリックS3
      - オンタップS3
      - ストレージグリッドS3

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

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

8. 宛先クラスターで、AppMirrorRelationship CR ファイルを作成します。

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. 次の属性を構成します。

- **metadata.name:** (必須) AppMirrorRelationship カスタム リソースの名前。
- **spec.destinationAppVaultRef**: (必須) この値は、宛先クラスター上の宛先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) 宛先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 で定義されている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) クラスター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、クラスター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.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. 入力したら `trident-protect-relationship.yaml`正しい値を持つファイルには、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> -n <application_namespace>
```

例：

```
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) 宛先クラスターで、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
```

宛先クラスタへのフェイルオーバー

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

手順

1. 宛先クラスタで、AppMirrorRelationship CRファイルを編集します（例： `trident-protect-relationship.yaml` ）の*spec.desiredState*の値を次のように変更します。 `Promoted` 。
2. CR ファイルを保存します。
3. CR を適用します。

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

4. (オプション) フェールオーバー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、宛先クラスター上の実行中のアプリケーションに加えられた変更はすべて破棄されます。

このプロセスでは、レプリケーションを再確立する前に、宛先クラスター上のアプリを停止します。

フェイルオーバー中に宛先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. 宛先クラスタで、AppMirrorRelationship CRファイルを編集します（例： `trident-protect-relationship.yaml` ）spec.desiredStateの値を次のように変更します。 `Established` 。
3. CR ファイルを保存します。
4. CR を適用します。

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

5. フェールオーバーされたアプリケーションを保護するために宛先クラスターに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。

フェイルオーバーされ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、宛先アプリケーションがソース アプリケーションになり、ソースが宛先になります。フェイルオーバー中に宛先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. 元の宛先クラスターで、AppMirrorRelationship CR を削除します。これにより、宛先がソースになります。新しい宛先クラスター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ーに適用して、レプリケーション関係を設定します。
3. 新しい宛先 (元のソース クラスター) が両方の AppVault CR で構成されていることを確認します。
4. 反対方向の値を設定して、反対側のクラスター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宛先ストレージ バックエンドに移動します。Trident Protect は、ソース アプリケーションを停止し、宛先アプリにフェールオーバーする前にデータを宛先に複製します。

この状況では、ソースと宛先が入れ替わっています。

手順

1. ソース クラスターで、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. 次の属性を構成します。

   - **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. 入力したら `trident-protect-shutdownsnapshot.yaml`正しい値を持つファイルには、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スナップショットを作成する

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。例えば：

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

2. ソース クラスターで、シャットダウン スナップショットが完了したら、シャットダウン スナップショット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 クラスターで、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、最後のスラッシュの後のすべて) を記録します。

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

4. 次の変更を加えて、新しい宛先クラスターから新しいソース クラスターへのフェイルオーバーを実行します。

> フェイルオーバー手順のステップ2では、 `spec.promotedSnapshot` AppMirrorRelationship CR ファイルのフィールドに追加し、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. 逆再同期の手順を実行します。[フェイルオーバーされたレプリケーション関係を逆再同期する] 。
6. 新しいソース クラスター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、複製されます。
- SnapMirror関係が解除され、宛先ボリュームは読み取り/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複製されたボリューム データを使用して、シャットダウン前のスナップショットから復元されます。
- 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、次の一連の操作によって、フェイルオーバー操作後の「フェイルバック」を実現できます。元のレプリケーション方向を復元するこのワークフローでは、 Trident Protect は、レプリケーション方向を反転する前に、アプリケーションの変更を元のソース アプリケーションに複製 (再同期) します。

このプロセスは、宛先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宛先クラスター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. 実行する[フェイルオーバーされたレプリケーション関係を逆再同期する]手順。
2. 実行する[アプリケーションのレプリケーション方向を逆にする]手順。

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ーで、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構成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。復元後の実行フックが存在する場合、復元操作の一部として自動的に実行されます。

バックアップと復元操作

次のシナリオでバックアップおよび復元操作を実行するには、特定のバックアップおよび復元タスクを自動化できます。

同じクラスターにクローンする

アプリケーションを同じクラスターに複製するには、スナップショットまたはバックアップを作成し、データを同じクラスターに復元します。

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. 同じクラスターで、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"。

別のクラスターにクローンする

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。宛先クラスターにTrident Protect がインストールされていることを確認します。

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" 。

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 が宛先クラスターで設定されていることを確認します。
3. 宛先クラスタでは、"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

バックアップを宛先ストレージ 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する) 次のようにします。

```
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、名前を付けます `trident-protect-snapshot-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 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**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     ```
     ---
     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) 使用 `include or exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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すると、それらは OR 演算として一致し、各要素内のフィールド (グループ、種類、バージョン) は AND 演算として一致します。
       - **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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-restore-cr.yaml`正しい値を持つファイルには、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名前空間に復元します。
   - 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>` 。
   - その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。

     例えば：

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

Trident Protect実行フックを管理する

実行フックは、管理対象アプリ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。

実行フックの種類

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操作前実行フックは、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 によるファイルシステムのフリーズ設定の詳細"。
3. データ保護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、事前スナップショット）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。

実行フック スクリプトを実稼働環境で有効にする前に、必ずテストする必要があります。'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。実稼働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、その後のすべての復元操作に含まれます。

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 96 KB 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スナップショット操作を開始したが、その後キャンセルした場合でも、バックアップまたはスナップショット操作がすでに開始されている場合は、フックは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ーを追加して、フックが一致するコンテナーを管理できます。フィルターは、すべてのコンテナーで同じコンテナー イメージを使用するが、各イメージを異なる目的で使用する可能性があるアプリケーション (Elasticsearch など) に役立ちます。フィルターを使用すると、実行フックが必ずしもすべての同一コンテナーではなく一部のコンテナーで実行される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ーを作成すると、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 フィルターを設定できます。

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。フィルターの正規表現では正規表現 2 (RE2) 構文が使用されますが、一致リストからコンテナーを除外するフィルターの作成

はサポートされていません。Trident Protectが実行フックフィルタでサポートする正規表現の構文については、以下を参照してください。 "正規表現2（RE2）構文のサポート" 。

> 復元またはクローン操作後に実行される実行フックに名前空間フィルターを追加し、復元またはクローンのソースと宛先が異なる名前空間にある場合、名前空間フィルターは宛先の名前空間にのみ適用されます。

実行フックの例

訪問 "NetApp Verda GitHub プロジェクト"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ます。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。

実行フックを作成する

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、名前を付けます `trident-protect-hook.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **metadata.name**: (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。有効な値は次のとおりです。
     - プレ
     - 投稿
   - **spec.action**: (必須) 指定された実行フック フィルターが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値は次のとおりです。
     - 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。指定しない場合はデフォルト値は true になります。
   - **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はデフォルト値は 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) 実行フック フィルターを構成する基準キー値のペアのオプション リスト。実行フックごとに最大 10 個のフィルターを追加できます。
   - **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。有効な値は次のとおりです。
     - コンテナイメージ
     - コンテナ名
     - ポッド名
     - ポッドラベル
     - 名前空間名
   - **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。例えば：

```
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、所有者、管理者、またはメンバーの権限が必要です。

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、名前を付けます `trident-protect-resource-backup.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **metadata.name**: (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**: バックアップ コンテンツが保存される AppVault 内のパス。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。例えば：

```
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、名前を付けます `trident-protect-hook-run.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **metadata.name**: (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ます。
   - **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ます。
   - **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ます。

     ```
     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 フィルターが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値は次のとおりです。
     - 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。有効な値は次のとおりです。
     - 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場合があります。

Trident Protect を削除するには、次の手順を実行します。

手順

1. Trident Protect CR ファイルを削除します。

   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を削除します。

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

3. Trident Protect 名前空間を削除します。

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