Trident保護でアプリケーションを保護

Trident

NetApp
January 17, 2025

目次

Trident保護でアプリケーションを保護

Trident protectの詳細

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ニングツールを基盤とするステートフルな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にわたるコンテナ化されたワークロードの管理、保護、移動を簡易化します。また、APIとCLIを使用した自動化機能も提供します。

Trident保護を使用してアプリケーションを保護するには、カスタムリソース（CRS）を作成するか、Trident保護CLIを使用します。

次の手順

インストールする前に、Trident保護の要件について確認できます。

- "Trident保護の要件"

Tridentプロテクトのインストール

Trident保護の要件

まず、運用環境、アプリケーションクラスタ、アプリケーション、ライセンスの準備状況を確認します。Trident保護を導入して運用するには、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は**Kubernetes**の互換性を保護

Trident Protectは、次のようなフルマネージドおよび自己管理型の幅広いKubernetes製品と互換性があります。

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

Tridentはストレージバックエンドの互換性を保護

Trident保護は、次のストレージバックエンドをサポートします。

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

ストレージバックエンドが次の要件を満たしていることを確認します。

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

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リストア処理をサポートします。Snapshot、クローン、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保護で使用するNASエコノミーボリュームごとに、スナップショットディレクトリを有効にする必要があります。

> 一部のアプリケーションは、Snapshotディレクトリを使用するボリュームと互換性がありません。これらのアプリケーションでは、ONTAPストレージシステムで次のコマンドを実行して、snapshotディレクトリを非表示にする必要があります。
>
> ```
> nfs modify -vserver <svm> -v3-hide-snapshot enabled
> ```

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

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

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

KubeVirt VMによるデータ保護

Trident protect 24.10および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、データ保護処理中のファイルシステムのフリーズおよびフリーズ解除を有効または無効にすることができます。

すべてのTrident保護バージョンで、OpenShift環境で自動フリーズ機能を有効または無効にするには、アプリケーションの名前空間に特権権限を付与する必要があります。例：

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

Trident保護24.10

Trident protect 24.10では、データ保護処理中にKubeVirt VMファイルシステムの一貫した状態が自動的に保証されません。Trident protect 24.10を使用してKubeVirt VMデータを保護する場合は、データ保護処理の前にファイルシステムのフリーズ/フリーズ解除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であることが保証されます。

データ保護処理中のVMファイルシステムのフリーズおよびフリーズ解除を管理するようにTrident protect 24.10を設定するには、"仮想化の設定"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以降では、Trident protectでは、データ保護処理中にKubeVirtファイルシステムが自動的にフリーズおよびフリーズ解除されます。必要に応じて、次のコマンドを使用してこの自動動作を無効にできます。

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

SnapMirrorレプリケーションの要件

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

- NetApp ASA
- NetApp AFF
- NetApp FAS
- NetApp ONTAP Select の略
- NetApp Cloud Volumes ONTAP の略
- NetApp ONTAP 対応の Amazon FSX

SnapMirrorレプリケーション用のONTAPクラスタの要件

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

- * Astra Control ProvisionerまたはTrident *：Astra Control ProvisionerまたはTridentが、ONTAPをバックエンドとして利用するソースとデスティネーションの両方のKubernetesクラスタに存在している必要があります。Trident保護では、次のドライバに基づくストレージクラスを使用したNetApp SnapMirrorテクノロジによるレプリケーションがサポートされます。
  - 「 ONTAP - NAS 」
  - 「 ontap - san 」
- ライセンス：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデスティネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

SnapMirrorレプリケーションのピアリングに関する考慮事項

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

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

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

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

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

Trident保護を使用するには、ソースとデスティネーションの両方のクラスタのレプリケーションをサポートするストレージバックエンドを少なくとも1つ設定する必要があります。ソースクラスタとデスティネーションクラスタが同じである場合は、耐障害性を最大限に高めるために、デスティネーションアプリケーションでソースアプリケーションとは別のストレージバックエンドを使用する必要があります。

Trident保護のインストールと設定

ご使用の環境がTrident保護の要件を満たしている場合は、次の手順に従ってクラスタにTrident保護をインストールします。NetAppからTrident protectを取得するか、独自のプライベートレジストリからインストールできます。プライベートレジストリからインストールすると、クラスタがインターネットにアクセスできない場合に役立ちます。

デフォルトでは、Trident protectは、クラスタと管理対象アプリケーションに関するログ、指標、トポロジ情報など、NetAppサポートケースをオープンする際に役立つサポート情報を収集します。Trident PROTECTは、これらのサポートバンドルを日次スケジュールでNetAppに送信します。Trident protectのインストール時に、必要に応じてこのサポートバンドル収集を無効にすることができます。いつでも手動で行うことができ"サポートバンドルの生成"ます。

Trident protect from NetAppのインストール

手順

1. Trident Helmリポジトリを追加します。

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

2. TridentプロテクトCRDをインストールします。

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

3. 次のいずれかのコマンドを使用して、Helmを使用してTrident protectをインストールします。をクラスタ名に置き換えます `<name_of_cluster>`。クラスタに割り当てられ、クラスタのバックアップとスナップショットの識別に使用されます。

   - Trident保護を通常どおりインストールします。

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

   - Trident protectをインストールし、Trident protect AutoSupportサポートバンドルのスケジュールされた毎日のアップロードを無効にします。

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

Trident protectをプライベートレジストリからインストールする

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

手順

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

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exechook:24.10.1
netapp/resourcebackup:24.10.1
netapp/resourcerestore:24.10.1
netapp/resourcedelete:24.10.1
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例：

```
docker pull netapp/controller:24.10.1
```

```
docker tag netapp/controller:24.10.1 <private-registry-
url>/controller:24.10.1
```

```
docker push <private-registry-url>/controller:24.10.1
```

2. Trident protect systemネームスペースを作成します。

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

3. レジストリにログインします。

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

4. プライベートレジストリ認証に使用するプルシークレットを作成します。

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

5. Trident Helmリポジトリを追加します。

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

6. 次のTrident保護設定を含むという名前のファイルを作成します `protectValues.yaml`。

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

7. TridentプロテクトCRDをインストールします。

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

8. 次のいずれかのコマンドを使用して、Helmを使用してTrident protectをインストールします。をクラスタ名に置き換えます `<name_of_cluster>`。クラスタに割り当てられ、クラスタのバックアップとスナップショットの識別に使用されます。
   - Trident保護を通常どおりインストールします。

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

   - Trident protectをインストールし、Trident protect AutoSupportサポートバンドルのスケジュールされた毎日のアップロードを無効にします。

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

Trident保護CLIプラグインのインストール

Tridentユーティリティの拡張機能であるTrident protectコマンドラインプラグインを使用すると、Trident protectカスタムリソース（CRS）を作成して操作できます `tridentctl`。

Trident保護CLIプラグインのインストール

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

Linux AMD64 CPU用プラグインのダウンロード

手順

1. Trident保護CLIプラグインをダウンロードします。

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

Linux ARM64 CPU用プラグインのダウンロード

手順

1. Trident保護CLIプラグインをダウンロードします。

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

Mac AMD64 CPU用プラグインのダウンロード

手順

1. Trident保護CLIプラグインをダウンロードします。

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

Mac ARM64 CPU用プラグインのダウンロード

手順

1. Trident保護CLIプラグインをダウンロードします。

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

1. バイナリの実行権限を有効にします。

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

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

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

3. 必要に応じて、バイナリをホームディレクトリ内の場所にコピーできます。この場合は、PATH変数に場所を追加する必要があります。

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

Trident CLIプラグインのヘルプを表示

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

手順

1. ヘルプ機能を使用して、使用方法に関するガイダンスを表示します。

```
tridentctl protect help
```

コマンドの自動補完を有効にする

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

Bashシェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします。

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

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

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

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

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

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

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

Zシェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします。

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

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

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

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

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

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

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

結果

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

Trident保護を管理します。

Trident保護の許可とアクセス制御を管理します。

Trident保護では、KubernetesモデルのRole-Based Access Control（RBAC；ロールベースアクセス制御）が使用されます。デフォルトでは、Trident保護は単一のシステムネームスペースとそれに関連付けられたデフォルトのサービスアカウントを提供します。多数のユーザがいる組織や、特定のセキュリティニーズがある組織では、Trident保護のRBAC機能を使用して、リソースやネームスペースへのアクセスをより細かく制御できます。

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

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

> ⓘ 管理者のみが、次のような特権Trident保護カスタムリソースオブジェクトへのアクセス権を持つ必要があります。
>
> - * AppVault *：バケット資格情報データが必要です。
> - * AutoSupportBundle *：指標、ログ、その他の機密性の高いTridentデータを収集します。
> - * AutoSupportBundleSchedule *：ログ収集スケジュールを管理します。
>
> RBACを使用して、権限付きオブジェクトへのアクセスを管理者に制限することを推奨します。

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

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

例：**2**つのユーザグループのアクセスを管理する

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

ープがそれぞれのネームスペースに割り当てられたリソースのみにアクセスできる環境を作成します。

手順1：各グループのリソースを含むネームスペースを作成する

ネームスペースを作成すると、リソースを論理的に分離し、それらのリソースにアクセスできるユーザをより細かく制御できます。

手順

1. engineeringグループの名前空間を作成します。

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

2. marketingグループの名前空間を作成します。

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

ステップ2：各ネームスペースのリソースとやり取りするための新しいサービスアカウントを作成する

作成する新しい名前空間にはそれぞれデフォルトのサービスアカウントが付属していますが、将来必要に応じてPrivilegesをグループ間でさらに分割できるように、ユーザーのグループごとにサービスアカウントを作成する必要があります。

手順

1. engineeringグループのサービスアカウントを作成します。

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

2. マーケティンググループのサービスアカウントを作成します。

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

ステップ3：新しいサービスアカウントごとにシークレットを作成する

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

手順

1. エンジニアリングサービスアカウントのシークレットを作成します。

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

2. マーケティングサービスアカウントのシークレットを作成します。

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

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

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

手順

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

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

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

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

手順**5**：権限のテスト

権限が正しいことをテストします。

手順

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

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

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

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

手順**6**：**AppVault**オブジェクトへのアクセスを許可する

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

手順

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

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

2. 役割CRを作成して適用し、クラスタ管理者がネームスペース内の特定のリソースへのアクセスを許可できるようにします。例：

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

3. RoleBinding CRを作成して適用し、権限をeng-userというユーザにバインドします。例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 権限が正しいことを確認します。

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

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

   次のような出力が表示されます。

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

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

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

次のような出力が表示されます。

```
yes
```

結果

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

Trident保護サポートバンドルの生成

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、指標、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NetAppサポートサイト（NSS）にサポートバンドルをアップロードできます。

CRを使用したサポートバンドルの作成

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-support-bundle.yaml`）。
2. 次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.triggerType *：（*required*）サポートバンドルをすぐに生成するかスケジュールするかを指定します。スケジュールされたバンドル生成は12AM UTCに行われます。有効な値：
     - スケジュール済み
     - 手動
   - * spec.uploadEnabled *：（_Optional _）サポートバンドルの生成後にNetAppサポートサイトにアップロードするかどうかを制御します。指定しない場合、デフォルトはになります `false`。有効な値：
     - 正しいです
     - false（デフォルト）
   - **spec.dataWindowStart**:(*Optional*)サポートバンドルに含まれるデータのウィンドウを開始する日時を指定する、RFC 3339形式の日付文字列。指定しない場合は、デフォルトで24時間前になります。指定できる最も早い期間の日付は7日前です。

     YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

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

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

CLIを使用したサポートバンドルの作成

手順

1. サポートバンドルを作成し、角かっこ内の値を環境からの情報に置き換えます。は `trigger-type`、バンドルをすぐに作成するか、スケジュールによって作成時間が指定されているかを決定し、または `Scheduled`を指定できます `Manual`。デフォルト設定はです `Manual`。

例：

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

Trident保護のアップグレード

Trident protectを最新バージョンにアップグレードすると、新機能やバグ修正を利用できます。

Trident保護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保護CRDをアップグレードします。

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

3. アップグレードTrident保護：

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

アプリケーションの管理と保護

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

Trident保護用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、Snapshot、リストア処理、SnapMirrorレプリケーションなど、保護処理でバケットを使用するために必要な設定が含まれています。AppVaultsを作成できるのは管理者のみです。

キー生成とAppVault定義の例

AppVault CRを定義するときは、プロバイダがホストするリソースにアクセスするための資格情報を含める必

要があります。クレデンシャルのキーの生成方法は、プロバイダによって異なります。次に、いくつかのプロバイダのコマンドラインキー生成の例を示します。次に、各プロバイダのAppVault定義の例を示します。

キー生成の例

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

Google Cloud

```
kubectl create secret generic <secret-name> --from-file=credentials
=<mycreds-file.json> -n trident-protect
```

Amazon S3（AWS）

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

Microsoft Azure

```
kubectl create secret generic <secret-name> --from-literal=accountKey
=<secret-name> -n trident-protect
```

汎用 S3

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

ONTAP S3

```
kubectl create secret generic <secret-name> --from-literal=accessKeyID
=<objectstorage-accesskey> --from-literal=secretAccessKey=<generic-s3-
trident-protect-src-bucket-secret> -n trident-protect
```

StorageGRID S3

```
kubectl create secret generic  <secret-name> --from-literal=
accessKeyID=<objectstorage-accesskey> --from-literal=secretAccessKey
=<generic-s3-trident-protect-src-bucket-secret> -n trident-protect
```

AppVault CRの例

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

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Amazon S3（AWS）

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

汎用 S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

StorageGRID S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-
971f-ac4a83621922
  namespace: trident-protect
spec:
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Trident保護CLIを使用した**AppVault**の作成例

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

Google Cloud

```
tridentctl protect create vault GCP my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3（AWS）

```
tridentctl protect create vault AWS <vault-name> --bucket <bucket-name>
--secret  <secret-name>  --endpoint <s3-endpoint>
```

Microsoft Azure

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

汎用 **S3**

```
tridentctl protect create vault GenericS3 <vault-name> --bucket
<bucket-name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

ONTAP S3

```
tridentctl protect create vault OntapS3 <vault-name> --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

StorageGRID S3

```
tridentctl protect create vault StorageGridS3 s3vault --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

AppVaultブラウザを使用してAppVault情報を表示する

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

手順

1. AppVaultオブジェクトの内容を表示します。

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

出力例：

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

   テーブルの最初の列に表示されるクラスタ名は、Trident protect helmのインストールでクラスタ名が指定されている場合にのみ使用できます。例： `--set clusterName=production1`。

AppVaultの削除

AppVaultオブジェクトはいつでも削除できます。

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

作業を開始する前に

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

Kubernetes CLIを使用した**AppVault**の削除

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

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

Trident CLIを使用した**AppVault**の削除

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

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

Trident保護で管理アプリケーションを定義

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

AppVault CRの作成

アプリケーションでデータ保護処理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、Trident保護がインストールされているクラスタにAppVault CRを配置する必要があります。AppVault CRはお使いの環境に固有です。AppVault CRSの例については、"AppVaultカスタムリソース。"

アプリケーションの定義

Trident保護で管理するアプリケーションをそれぞれ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保護CLIを使用して、管理対象のアプリケーションを定義できます。

CRを使用したアプリケーションの追加

手順

1. デスティネーションアプリケーションのCRファイルを作成します。

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

   b. 次の属性を設定します。

      - * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。保護操作に必要な他のCRファイルがこの値を参照するため、選択した名前をメモします。
      - * spec.includedNamespaces*:(*required*)名前空間ラベルまたは名前空間名を使用して、アプリケーションリソースが存在する名前空間を指定します。アプリケーション名前空間は、このリストの一部である必要があります。

        YAMLの例：

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

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

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

CLIを使用したアプリケーションの追加

手順

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

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

Trident保護を使用したアプリケーションの保護

自動保護ポリシーまたはアドホックベースでスナップショットとバックアップを作成することで、Trident protectで管理されているすべてのアプリケーションを保護できます。

データ保護処理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保護を設定できます。"Trident protectを使用したファイルシステムのフリーズ設定の詳細"です。

オンデマンドスナップショットを作成します

オンデマンド Snapshot はいつでも作成できます。

CRを使用したスナップショットの作成

手順

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

```
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  reclaimPolicy: Delete
```

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

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

CLIを使用したスナップショットの作成

手順

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

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

オンデマンドバックアップの作成

アプリはいつでもバックアップできます。

CRを使用したバックアップの作成

手順

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

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

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

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

CLIを使用したバックアップの作成

手順

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

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

データ保護スケジュールを作成

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

CRを使用したスケジュールの作成

手順

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

```
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"
```

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

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

CLIを使用してスケジュールを作成する

手順

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

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

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

Snapshot を削除します

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

手順

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

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

バックアップを削除します

不要になったスケジュール済みまたはオンデマンドのバックアップを削除します。

手順

1. バックアップに関連付けられているバックアップCRを削除します。

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

バックアップ処理のステータスの確認

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

手順

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

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

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

Trident protectをインストールしている場合はNetApp、Trident 24.06より前に作成されたazure-lun-filesストレージクラスを使用するストレージバックエンドに対して、スペース効率に優れたバックアップおよびリストア機能を有効にすることができます。この機能はNFSv4ボリュームで機能し、容量プールから追加のスペースを消費することはありません。

作業を開始する前に

次の点を確認します。

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

構成手順用に展開

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

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

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

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

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

   応答：

   ```
   snapshotDirectory: "true"
   ```

   +
   Snapshotディレクトリが有効になっていない場合、Trident保護は通常のバックアップ機能を選択します。この機能は、バックアッププロセス中に一時的に容量プールのスペースを消費します。この場合は、バックアップするボリュームと同じサイズの一時ボリュームを作成するための十分なスペースが容量プールに確保されていることを確認してください。

結果

これで、Trident保護を使用したアプリケーションのバックアップとリストアが可能になります。各PVCは、他のアプリケーションでバックアップおよびリストアに使用することもできます。

Trident保護を使用したアプリケーションのリストア

Trident保護を使用すると、Snapshotまたはバックアップからアプリケーションをリストアできます。同じクラスタにアプリケーションをリストアする場合、既存の Snapshot からのリストアは高速です。

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

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

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

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

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

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

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

フラグを指定してHelmを使用してソースアプリケーションをインストールした場合は `--create-namespace`、ラベルキーに特別な処理が行わ `name`れます。Trident保護では、リストアまたはフェイルオーバーのプロセスでこのラベルがデスティネーションネームスペースにコピーされますが、ソースの値がソースネームスペースと一致する場合はデスティネーションネームスペースの値に更新されます。この値がソースネームスペースと一致しない場合、変更なしでデスティネーションネームスペースにコピーされます。

例

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

リストアまたはフェイルオーバー処理の前

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

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true"	• 環境=本番 • コンプライアンス= HIPAA • 名前= ns-1
ネームスペースns-2（デスティネーション）	• Annotation.one/key："true" • annotation.three/key："false"	• ロール=データベース

リストア処理後

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

ネームスペース	アノテーション	ラベル
ネームスペースns-2（デスティネーション）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true" • annotation.three/key："false"	• 名前= ns-2 • コンプライアンス= HIPAA • 環境=本番 • ロール=データベース

バックアップから別のネームスペースへのリストア

BackupRestore CRを使用して別のネームスペースにバックアップをリストアすると、Trident保護によってアプリケーションが新しいネームスペースにリストアされますが、リストアされたアプリケーションはTrident保護によって自動的に保護されません。リストアされたアプリケーションを保護するには、Trident保護によって保護されるように、リストアされたアプリケーションのアプリケーションCRを作成する必要があります。

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

CRの使用

手順

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

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

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

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

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

種類、バージョン）はAND演算として照合されます。

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

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

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

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

CLI を使用します

手順

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

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

バックアップから元のネームスペースへのリストア

バックアップはいつでも元のネームスペースにリストアできます。

CRの使用

手順

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

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

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

     例：

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

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

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

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

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

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

CLI を使用します

手順

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

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

Snapshotから別のネームスペースへのリストア

カスタムリソース（CR）ファイルを使用して、スナップショットから別のネームスペースまたは元のソースネームスペースにデータをリストアできます。SnapshotRestore CRを使用して別のネームスペースにSnapshotをリストアすると、Trident保護によって新しいネームスペースにアプリケーションがリストアされますが、リストアされたアプリケーションはTrident保護によって自動的に保護されません。リストアされた

アプリケーションを保護するには、Trident保護によって保護されるように、リストアされたアプリケーションのアプリケーションCRを作成する必要があります。

CRの使用

手順

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

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

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

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

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

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

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

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

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

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

CLI を使用します

手順

1. スナップショットを別のネームスペースにリストアし、括弧内の値を環境の情報に置き換えます。

   ◦ `snapshot`引数では、という形式のネームスペースとSnapshot名を使用し `<namespace>/<name>`ます。

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

  例：

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

Snapshotから元のネームスペースへのリストア

Snapshotはいつでも元のネームスペースにリストアできます。

CRの使用

手順

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

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

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

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

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

例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

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

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

CLI を使用します

手順

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

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

リストア処理のステータスの確認

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

手順

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

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

NetApp SnapMirrorとTridentによる保護を使用してアプリケーションをレプリケート

Trident保護を使用すると、NetApp SnapMirrorテクノロジの非同期レプリケーション機能を使用して、ストレージバックエンド間、同じクラスタ上、または異なるクラスタ間でデータやアプリケーションの変更をレプリケートできます。

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

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

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

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

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

フラグを指定してHelmを使用してソースアプリケーションをインストールした場合は `--create-namespace`、ラベルキーに特別な処理が行わ `name`れます。Trident保護では、リストアまたはフェイルオーバーのプロセスでこのラベルがデスティネーションネームスペースにコピーされますが、ソースの値がソースネームスペースと一致する場合はデスティネーションネームスペースの値に更新されます。この値がソースネームスペースと一致しない場合、変更なしでデスティネーションネームスペースにコピーされます。

例

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

リストアまたはフェイルオーバー処理の前

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

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true"	• 環境=本番 • コンプライアンス= HIPAA • 名前= ns-1
ネームスペースns-2（デスティネーション）	• Annotation.one/key："true" • annotation.three/key："false"	• ロール=データベース

リストア処理後

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

ネームスペース	アノテーション	ラベル
ネームスペースns-2（デスティネーション）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true" • annotation.three/key："false"	• 名前= ns-2 • コンプライアンス= HIPAA • 環境=本番 • ロール=データベース

データ保護処理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保護を設定できます。"Trident protectを使用したファイルシステムのフリーズ設定の詳細"です。

レプリケーション関係を設定

レプリケーション関係の設定には、次の作業が含まれます。

- Trident protectでアプリケーションスナップショットを作成する頻度を選択します（アプリケーションのKubernetesリソースと、アプリケーション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）。
- レプリケーションスケジュールの選択（Kubernetesリソースと永続ボリュームデータを含む）
- Snapshotの作成時間の設定

手順

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

CRを使用した**AppVault**の作成

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

b. 次の属性を設定します。

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

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

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

CLIを使用した**AppVault**の作成

a. AppVaultを作成し、括弧内の値を環境からの情報に置き換えます。

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

2. ソースアプリケーションCRを作成します。

CRを使用したソースアプリケーションの作成

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

b. 次の属性を設定します。

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

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

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

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

CLIを使用したソースアプリケーションの作成

a. ソースアプリケーションを作成します。例：

```
tridentctl protect create app maria --namespaces maria -n my-app-namespace
```

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

CRを使用したスナップショットの作成

a. ソースアプリケーションのレプリケーションスケジュールを作成します。

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

   ii. 次の属性を設定します。

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

     YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-
46a3-420a-b351-45795e1b5e34
  applicationRef: maria
  backupRetention: "0"
  enabled: true
  granularity: custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

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

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

CLIを使用したスナップショットの作成

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

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

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

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

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

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

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

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

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

CRを使用した**AppMirrorRelationship**の作成

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

b. 次の属性を設定します。

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

  YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: maria
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

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

す。

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

CLIを使用した**AppMirrorRelationship**の作成

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

```
tridentctl protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --recurrence-rule <rule> --source-app
<my_source_app> --source-app-vault <my_source_app_vault>
```

8. （オプション）レプリケーション関係の状態とステータスを確認します。

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

デスティネーションクラスタへのフェイルオーバー

Trident保護を使用すると、レプリケートされたアプリケーションをデスティネーションクラスタにフェイルオーバーできます。この手順 はレプリケーション関係を停止し、デスティネーションクラスタでアプリケーションをオンラインにします。Trident protectが動作していた場合、ソースクラスタ上のアプリは停止しません。

手順

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

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

4. （オプション）フェイルオーバーされたアプリケーション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
```

フェイルオーバーされたレプリケーション関係を再同期します。

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

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

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

手順

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

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

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

フェイルオーバーされたレプリケーション関係の逆再同期

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

手順

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

アプリケーションのレプリケーション方向を反転

レプリケーション方向を反転すると、Trident保護によってアプリケーションがデスティネーションストレージバックエンドに移動され、元のソースストレージバックエンドに引き続きレプリケートされます。Trident protectは、ソースアプリケーションを停止し、デスティネーションアプリケーションにフェイルオーバーする前にデータをデスティネーションにレプリケートします。

この状況では、ソースとデスティネーションを交換しようとしています。

手順

1. シャットダウンスナップショットを作成します。

CRを使用したシャットダウンスナップショットの作成

a. ソースアプリケーションの保護ポリシースケジュールを無効にします。

b. ShutdownSnapshot CRファイルを作成します。

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

ii. 次の属性を設定します。

- * metadata.name*:（*required*）カスタムリソースの名前。
- *spec.AppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVaultのmetadata.nameフィールドと一致する必要があります。
- *spec.ApplicationRef *:(*required*)この値は、ソースアプリケーションCRファイルのmetadata.nameフィールドと一致する必要があります。

YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: maria
```

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

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

CLIを使用したシャットダウンスナップショットの作成

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

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

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

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

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

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

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

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

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

結果

リバースレプリケーションが実行されると、次の処理が実行されます。

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

アプリケーションを元のソースクラスタにフェイルバックします

Trident保護を使用すると、フェイルオーバー処理後に次の一連の処理を使用して「フェイルバック」を実現できます。このワークフローでは、元のレプリケーション方向を復元するために、Trident保護は、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戻します（再同期）。

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

- フェイルオーバー状態から開始します。

- レプリケーション関係を逆再同期します。

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

- レプリケーションの方向を逆にします。

手順

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

レプリケーション関係を削除する

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

手順

1. AppMirrorRelationship CRを削除します。

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

Trident保護を使用したアプリケーションの移行

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

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

バックアップとリストアの処理

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

同じクラスタにクローニング

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

手順

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

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

別のクラスタにクローニング

アプリケーションを別のクラスタにクローニング（クラスタ間クローニングを実行）するには、ソースクラスタでバックアップを作成し、別のクラスタにリストアします。デスティネーションクラスタにTrident protectがインストールされていることを確認します。

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

手順

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

あるストレージクラスから別のストレージクラスへのアプリケーションの移行

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

たとえば、次のようになります（リストアCRのシークレットを除く）。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    destination: "${destinationNamespace}"
    source: "${sourceNamespace}"
  storageClassMapping:
    destination: "${destinationStorageClass}"
    source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

CRを使用したスナップショットのリストア

手順

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

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

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

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

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

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

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

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

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

CLIを使用したスナップショットのリストア

手順

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

     例：

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

Trident保護実行フックの管理

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

実行フックのタイプ

Trident PROTECTは、実行可能な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ます。

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

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

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

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

1. スナップショット前フックが実行されます

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

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

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

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

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

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

実行フックフィルタ

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

実行フックに追加する各フィルタは、正規表現を使用してクラスタ内のコンテナを照合します。フックがコンテナと一致すると、そのコンテナに関連付けられたスクリプトがフックによって実行されます。フィルタの正規表現では、正規表現2（RE2）構文を使用します。この構文では、一致リストからコンテナを除外するフィルタの作成はサポートされていません。実行フックフィルタの正規表現に対してTrident保護がサポートする構文については、を参照してください "正規表現2（RE2）構文のサポート"。

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

実行フックの例

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

実行フックの作成

Trident protectを使用して、アプリのカスタム実行フックを作成できます。実行フックを作成するには、Owner 、 Admin 、または Member のいずれかの権限が必要です。

CRの使用

手順

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

     YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: 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 を使用します

手順

1. 実行フックを作成し、括弧内の値を環境からの情報に置き換えます。例：

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

Tridentプロテクトのアンインストール

製品の試用版からフルバージョンにアップグレードする場合は、Trident保護コンポーネントの削除が必要になることがあります。

Trident保護を削除するには、次の手順を実行します。

手順

1. Trident保護CRファイルを削除します。

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

2. Trident保護を削除します。

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

3. Trident保護ネームスペースを削除します。

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

著作権に関する情報

商標に関する情報