アプリケーションの管理と保護
Trident

NetApp
March 11, 2026

目次

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認証とパスワードの設定

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。

Data Mover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。

Data Moverリポジトリパスワードシークレットの作成

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Data Moverに応じて、そのData Moverに対応するパスワードだけを含める必要があります。たとえば、Resticを使用していて、今後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の使用

```
---
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 を使用します

```
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、 "StorageGRID 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の**AppVault**キー生成例

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
```

StorageGRID S3

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

AppVaultの作成例

各プロバイダのAppVault定義の例を次に示します。

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。

- 必要に応じて、ResticおよびKopiaリポジトリ暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。詳細については、を参照してください Data Moverリポジトリのパスワード 。
- Amazon S3（AWS）AppVaultオブジェクトの場合、必要に応じて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、でプロバイダのキーを生成するときに作成されクラウドプロバイダのAppVaultキー生成例ます。
- S3 AppVaultオブジェクトの場合、必要に応じて、キーを使用して発信S3トラフィックの出力プロキシURLを指定できます `spec.providerConfig.S3.proxyURL`。

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`セクションを追加して `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

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGRID S3

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

Trident Protect CLI を使用した **AppVault** 作成例

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

- 必要に応じて、ResticおよびKopiaリポジトリ暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。詳細については、を参照してください Data Moverリポジトリのパスワード 。
- S3 AppVaultオブジェクトの場合は、引数を使用して送信S3トラフィックの出力プロキシURLをオプションで指定できます `--proxy-url <ip_address:port>`。

Google Cloud

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

Amazon S3（AWS）

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

Microsoft Azure

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

汎用 S3

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

ONTAP S3

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

StorageGRID S3

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

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

S3 プロバイダーの構成オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト	例
`providerConfig.s3.skipCertValidation`	SSL/TLS 証明書の検証を無効にします。	いいえ	「真」、「偽」
`providerConfig.s3.secure`	S3 エンドポイントとの安全な HTTPS 通信を有効にします。	正しいです	「真」、「偽」
`providerConfig.s3.proxyURL`	S3 への接続に使用するプロキシ サーバーの URL を指定します。	なし	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	SSL/TLS 検証用のカスタム ルート CA 証明書を提供します。	なし	「CN=MyCustomCA」
`providerConfig.s3.useIAM`	S3 バケットにアクセスするための IAM 認証を有効にします。 EKS ポッド ID に適用可能です。	いいえ	true false

AppVault情報の表示

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

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

AppVaultの削除

AppVaultオブジェクトはいつでも削除できます。

AppVaultオブジェクトを削除する前に、AppVault CRのキーを削除しないで `finalizers`ください。これを行うと、AppVaultバケット内のデータが残り、クラスタ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

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CRSが削除されていることを確認します。

Kubernetes CLIを使用した**AppVault**の削除

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

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

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

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

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

Trident Protectで管理するアプリケーションを定義する

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

AppVault CRの作成

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

アプリケーションの定義

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

CRを使用したアプリケーションの追加

手順

1. デスティネーションアプリケーションのCRファイルを作成します。
   a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `maria-app.yaml` ）。
   b. 次の属性を設定します。
      - * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。保護操作に必要な他のCRファイルがこの値を参照するため、選択した名前をメモします。
      - * spec.includedNamespaces*:(*required*)名前空間とラベルセレクタを使用して、アプリケーションが使用する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に含まれている必要があります。ラベルセレクタはオプションで、指定した各名前空間内のリソースをフィルタリングするために使用できます。
      - * spec.includedClusterScopedResources*:（_Optional _）この属性を使用して、アプリケーション定義に含めるクラスタスコープ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、およびラベルに基づいてこれらのリソースを選択できます。
        - *groupVersionKind *:(*required*)クラスタスコープリソースのAPIグループ、バージョン、および種類を指定します。
        - *labelSelector *:(*Optional*)ラベルに基づいてクラスタスコープ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"。

アプリケーションの作成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します。

```
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"`。

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

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"]
```

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
```

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て適用します。括弧内の値は、環境の情報に置き換えます。アプリケーション定義に名前空間とリソースを含めるには、例に示す引数をカンマで区切ったリストを使用します。

   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、フラグを使用し`--annotation`ます。

- アプリケーションを作成し、ファイルシステム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します

オンデマンド Snapshot はいつでも作成できます。

クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CRを使用したスナップショットの作成

手順

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

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

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

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

CLIを使用したスナップショットの作成

手順

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

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

オンデマンドバックアップの作成

アプリはいつでもバックアップできます。

> クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

作業を開始する前に

長時間実行されるs3バックアップ処理には、AWSセッショントークンの有効期限が十分であることを確認してください。バックアップ処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

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スナップショットの名前。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。

     YAMLの例：

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

3. 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> --data-mover <Kopia_or_Restic> -n <application_namespace>
```

オプションで、フラグを使用して、バックアップを非増分にするかどうかを指定できます `--full-backup`。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です。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。リストアに伴うリスクを最小限に抑えるために、フルバックアップを定期的に実行してから、フルバックアップの間に増分バックアップを実行することを推奨します。

サポートされているバックアップ注釈

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

アノテーション	を入力します	説明	デフォルト値
protect.trident.netapp.io/フルバックアップ	文字列	バックアップを非増分にするかどうかを指定します。設定 `true`非増分バックアップを作成します。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。	いいえ
protect.trident.netapp.io/スナップショット完了タイムアウト	文字列	スナップショット操作全体が完了するまでに許容される最大時間。	「60メートル」
protect.trident.netapp.io/ボリュームスナップショットの使用準備完了のタイムアウト	文字列	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	「30メートル」
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	文字列	ボリューム スナップショットの作成に許可される最大時間。	「5メートル」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。	「1200」（20分）

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。
- クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CRを使用し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**: (必須) 保持するバックアップの数。ゼロは、バックアップを作成しないことを示します (スナップショットのみ)。
   - **spec.backupReclaimPolicy**: (オプション) バックアップ CR が保持期間中に削除された場合にバックアップに何が起こるかを決定します。保持期間が過ぎると、バックアップは常に削除されます。可能な値（大文字と小文字が区別されます）:
     - `Retain` （デフォルト）
     - `Delete`
   - **spec.snapshotRetention**: (必須) 保持するスナップショットの数。ゼロはスナップショットを作成しないことを示します。
   - **spec.snapshotReclaimPolicy**: (オプション) 保持期間中にスナップショット CR が削除された場合にスナップショットに何が起こるかを決定します。保持期間が過ぎると、スナップショットは常に削除されます。可能な値（大文字と小文字が区別されます）:
     - `Retain`
     - `Delete`（デフォルト）
   - * spec.granularity*:スケジュールを実行する頻度。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`（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。
- **spec.runImmediately**：（オプション） `true`に設定すると、スケジュールの作成時に1回限りの即時ベースライン実行（保持設定に基づくバックアップやスナップショット）がトリガーされます。デフォルトは `false`です。これは以降の繰り返しには影響しません。

  バックアップとスナップショットのスケジュールの YAML の例:

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

  スナップショットのみのスケジュールの YAML の例:

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

即時実行のスケジュールの YAML の例：

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

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> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_to_run_schedule> \
  --granularity <frequency_to_run> \
  --hour <hour_of_day_to_run> \
  --minute <minute_of_hour_to_run> \
  --recurrence-rule <recurrence> \
  --snapshot-retention <how_many_snapshots_to_retain> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

次のフラグを使用すると、スケジュールをさらに制御できます。

- フルバックアップスケジュール: `--full-backup-rule` 非増分完全バックアップをスケジュールするためのフラグ。このフラグは、 `--granularity Daily`。可能な値:
  - `Always`: 毎日フル バックアップを作成します。
  - 特定の曜日: 1 つ以上の曜日をカンマで区切って指定します (例: `"Monday,Thursday"`）。有効な値: 月曜日、火曜日、水曜日、木曜日、金曜日、土曜日、日曜日。

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

- 即時ベースライン保護： `--run-immediately true`を使用すると、最初のスケジュールされた実行時間を待たずに、スケジュールが作成されるとすぐに初期バックアップまたはスナップショットを作成できます。デフォルトは `false`です。
- スナップショットのみのスケジュール: 設定 `--backup-retention 0` 0より大きい値を指定する `--snapshot-retention`。

サポートされているスケジュール注釈

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

アノテーション	を入力します	説明	デフォルト値
protect.trident.netapp.io/フルバックアップルール	文字列	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します。設定できるのは `Always` 継続的なフル 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、日単位の粒度を選択した場合、フルバックアップを実行する曜日を指定できます（例： `"Monday,Thursday"`）。有効な曜日の値は、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日、日曜日です。この注釈は、以下のスケジュールでのみ使用できます。 `granularity` に設定 `Daily`。	未設定（すべてのバックアップは増分です）
protect.trident.netapp.io/スナップショット完了タイムアウト	文字列	スナップショット操作全体が完了するまでに許容される最大時間。	「60メートル」
protect.trident.netapp.io/ボリュームスナップショットの使用準備完了のタイムアウト	文字列	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	「30メートル」
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	文字列	ボリューム スナップショットの作成に許可される最大時間。	「5メートル」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。	「1200」（20分）

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。

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。このポリシーのデフォルト設定は `Retain`偶発的なデータ損失を防ぐためです。ポリシーが変更されていない場合は、 `Delete`バックアップ データはオブジェクト ストレージに残り、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
```

バックアップ処理のステータスの確認

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

手順

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

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

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

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

作業を開始する前に

次の点を確認します。

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

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

結果

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

リストアアプリケーション

Trident Protectを使用してアプリケーションを復元する

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

- アプリケーションを復元すると、そのアプリケーションに設定されているすべての実行フックがアプリケーションとともに復元されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。
- qtreeボリュームでは、バックアップから別のネームスペースまたは元のネームスペースへの復元がサポートされています。ただし、スナップショットから別のネームスペースまたは元のネームスペース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ネームスペースへのリストア

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

- 既存のリソースがある別の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 を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

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.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペース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: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       destinationApplicationName: my-new-app-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （オプション）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。

   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> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

バックアップから元のネームスペースへのリストア

いつでもバックアップを元の名前空間に復元できます。In Placeリストアを実行すると、Trident Protectは、保護スケジュールと進行中の操作を自動的に管理し、無効な回復ポイントを防止します。

- 復元が開始される前に、アプリケーションの有効なすべての保護スケジュールが無効になります。これにより、アプリケーション リソースの復元中に、スケジュールされたバックアップまたはSnapshotが実行されなくなります。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。すでに無効になっているスケジュールは無効のままになります。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元は続行され、復元CRステータスに警告が記録されます。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

> 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。

   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> \
-n <application_namespace>
```

バックアップから別のクラスタへのリストア

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

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

作業を開始する前に

次の前提条件が満たされていることを確認します。

- 宛先クラスターにはTrident Protect がインストールされています。
- デスティネーションクラスタは、バックアップが格納されているソースクラスタと同じAppVaultのバケットパスにアクセスできます。
- 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。
- 長時間実行されるリストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。
  - 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント"ください。
  - AWSリソースのクレデンシャルの詳細については、を参照してください "AWSのドキュメント"。

手順

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

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

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

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

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

このコマンドを実行すると、AppVaultで使用可能なバックアップが表示されます。これには、元のクラス

タ、対応するアプリケーション名、タイムスタンプ、アーカイブパスが含まれます。

出力例：

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

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

CRの使用

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

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

     BackupRestore CRを使用できない場合は、手順2のコマンドを使用してバックアップの内容を表示できます。
   - **spec.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     例：

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

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

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

CLI を使用します

1. 次のコマンドを使用してアプリケーションをリストアし、括弧内の値を環境の情報に置き換えます。namespace-mapping引数では、コロンで区切られた名前空間を使用して、ソース名前空間をsource1:dest1、source2:dest2の形式で正しいデスティネーション名前空間にマッピングします。例：

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

Snapshotから別のネームスペースへのリストア

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

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

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

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.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペース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: 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）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。

   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> \
--destination-app-name <custom_app_name> \
-n <application_namespace>
```

Snapshotから元のネームスペースへのリストア

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。In Placeリストアを実行すると、Trident Protectは、保護スケジュールと進行中の操作を自動的に管理し、無効な回復ポイントを防止します：

- 復元が開始される前に、アプリケーションの有効なすべての保護スケジュールが無効になります。これにより、アプリケーション リソースの復元中に、スケジュールされたバックアップまたはSnapshotが実行されなくなります。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。すでに無効になっているスケジュールは無効のままになります。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元は続行され、復元CRステータスに警告が記録されます。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

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。

> 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 <namespace/snapshot_to_restore> \
-n <application_namespace>
```

リストア処理のステータスの確認

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

手順

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

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

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。

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。

> 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に関するドキュメント"。

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：

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

> 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、 "追加のTrident Protectヘルムチャート設定を構成する"。

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、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

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、システムのさまざまな動作を設定するためにサポートされているアノテーションの一覧を示します。ユーザーがアノテーションを明示的に設定しない場合、システムはデフォルト値を使用します。

アノテーション	を入力します	説明	デフォルト値
保護.trident.netapp.io/データムーバータイムアウト秒	文字列	データ ムーバー操作を停止できる最大時間 (秒単位)。	「300」
保護.trident.netapp.io/kopia-content-cache-size-limit-mb	文字列	Kopia コンテンツ キャッシュの最大サイズ制限 (メガバイト単位)。	「1000」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。すべての復元 CR タイプ (BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore) に適用されます。ストレージ バックエンドまたはクラスターでより多くの時間が必要になることが多い場合は、より高い値を使用します。	「1200」（20分）

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

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

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

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

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に関するドキュメント" 。

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：

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

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、 "追加のTrident Protectヘルムチャート設定を構成する"。

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、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、実行フックはアプリケーション上に存在し、関連するアクションの実行中に実行されます。
- リバース同期またはリバース再同期中、アプリケーション上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ターゲットアプリケーションになると、これらの実行フックは無効となり、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

レプリケーション関係の設定には、次の作業が含まれます。

- Trident Protect がアプリのスナップショット (アプリの Kubernetes リソースとアプリの各ボリュームのボリューム スナップショットが含まれます) を作成する頻度を選択します。

- レプリケーションスケジュールの選択（Kubernetesリソースと永続ボリュームデータを含む）
- Snapshotの作成時間の設定

手順

1. ソースクラスタで、ソースアプリケーションの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> -n
trident-protect
```

2. ソースクラスタで、ソースアプリケーションCRを作成します。

CRを使用し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: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

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

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

CLIを使用したソースアプリケーションの作成

a. ソースアプリケーションを作成します。例：

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

3. 必要に応じて、ソース 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します"。

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。

下記の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、7日間の保持期間を持つ日次スナップショットスケジュールを別途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日間利用可能になりますが、保持期間はユーザーの要件に応じ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、逆操作に使用できます。このアプローチにより、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、逆操作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、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**: (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 *：（*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：

```yaml
---
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. 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を使用してレプリケーションスケジュールを作成する

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

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

- 例： *

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

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

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

7. デスティネーションクラスタに、デスティネーションアプリケーション用のデスティネーションAppVault CRを作成します。ストレージプロバイダに応じて、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 trident-protect
```

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

> CR を使用する場合は、CR を適用する前に、宛先名前空間を手動で作成します。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

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.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 *:開始日をUTC時間と繰り返し間隔で定義します。

  YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-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）デスティネーションクラスタで、レプリケーション関係の状態とステータスを確認します。

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

デスティネーションクラスタへのフェイルオーバー

Trident Protect を使用すると、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. デスティネーションクラスタで、AppMirrorRelationship CRファイル（など）を編集し `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. オプション：ソースクラスタで、ソースアプリケーションのSnapshotを作成します。これにより、ソースクラスタからの最新の変更がキャプチャされます。
2. デスティネーションクラスタで、AppMirrorRelationship CRファイル（など）を編集し `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. 新しいデスティネーション（元のソースクラスタ）に両方のAppVault CRSが設定されていることを確認します。
4. 反対側のクラスタにレプリケーション関係を設定し、逆方向の値を設定します。

アプリケーションのレプリケーション方向を反転

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

この状況では、ソースとデスティネーションを交換しようとしています。

手順

1. ソースクラスタで、シャットダウンSnapshotを作成します。

CRを使用したシャットダウンスナップショットの作成

a. ソースアプリケーションの保護ポリシースケジュールを無効にします。

b. ShutdownSnapshot CRファイルを作成します。

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

   ii. 次の属性を設定します。

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

        YAMLの例：

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

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

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

CLIを使用したシャットダウンスナップショットの作成

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

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

2. ソースクラスタで、シャットダウンSnapshotが完了したら、シャットダウンSnapshotのステータスを取得します。

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

3. ソースクラスタで、次のコマンドを使用して* shutdownsnapshot.status.appArchivePath *の値を探し、ファイルパスの最後の部分（basenameとも呼ばれます。最後のスラッシュのあとのすべての部分）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、新しいデスティネーションクラスタから新しいソースクラスタへのフェイルオーバーを実行します。

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

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

結果

リバースレプリケーションが実行されると、次の処理が実行されます。

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

アプリケーションを元のソースクラスタにフェイルバックします

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

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

- フェイルオーバー状態から開始します。
- レプリケーション関係を逆再同期します。

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

- レプリケーションの方向を逆にします。

手順

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

レプリケーション関係を削除する

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

手順

1. 現在のディサイトクラスタで、AppMirrorRelationship CRを削除します。

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

Trident Protectを使用してアプリケーションを移行する

バックアップ データを復元することで、アプリケーションをクラスター間または異なるストレージ クラスに移行できます。

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

バックアップとリストアの処理

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

同じクラスタにクローニング

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

手順

1. 次のいずれかを実行します。
   a. "Snapshot を作成します"です。

b. "バックアップを作成します"です。

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

別のクラスタにクローニング

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

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

手順

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

あるストレージクラスから別のストレージクラスへのアプリケーションの移行

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、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 Protect実行フックを管理する

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

実行フックのタイプ

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

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

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

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

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

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

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

本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にしたら、作成されたSnapshotとバックアップをテストして整合性があることを確認します。これを行うには、アプリケーションを一時的な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。

スナップショット前の実行フックで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、および後続のリストア処理に含まれます。

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

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

実行フックは、実行中のアプリケーションの機能を低下させたり、完全に無効にしたりすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。実行フックが関連付けられている状態でバックアップまたはスナップショット操作を開始した後'キャンセルした場合でも'バックアップまたはスナップショット操作がすでに開始されてい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" 。

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

実行フックの例

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

実行フックの作成

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

CRの使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-hook.yaml`ます。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.applicationRef *:(*required*)実行フックを実行するアプリケーションのKubernetes名。
   - *spec.stage *:(*required*)実行フックが実行されるアクションのステージを示す文字列。有効な値：
     - 前
     - 投稿
   - *spec.action *:(*required*)指定された実行フックフィルタが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値：
     - 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> -n <application_namespace>
```

実行フックを手動で実行する

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

実行フックを手動で実行するには、次の2つの基本ステップがあります。

1. リソースのバックアップを作成します。リソースを収集してバックアップを作成し、フックの実行場所を決定します。
2. バックアップに対して実行フックを実行する

手順**1**：リソースのバックアップを作成する

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-resource-backup.yaml`ます。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.applicationRef * :（*required*）リソースのバックアップを作成するアプリケーションのKubernetes名。
   - * spec.appVaultRef * :（*required*）バックアップコンテンツが格納されているAppVaultの名前。
   - **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

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

     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 を使用します

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 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。この例のコマンドは、処理が完了するまで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-run.yaml` ます。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.applicationRef *:(*required*)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 *:(*required*)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 *:(*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.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。このフックランは、他のステージではフックを実行しません。有効な値：
     - 前
     - 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 を使用します

手順

1. 手動実行フック実行要求を作成します。

```
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. 実行フック実行のステータスを確認します。このコマンドは、処理が完了するまで繰り返し実行できます。

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

3. exechooksrunオブジェクトについて説明し、最終的な詳細とステータスを確認します。

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