Astra Controlプロビジョニングツールを使用

Astra Control Service

NetApp
April 24, 2024

目次

Astra Controlプロビジョニングツールを使用 1
ストレージバックエンドの暗号化の設定 1
Snapshotを使用したボリュームデータのリカバリ 8
SnapMirrorによるボリュームのレプリケート 10

Astra Controlプロビジョニングツールを使用

ストレージバックエンドの暗号化の設定

Astra Control Provisionerを使用すると、管理対象クラスタとストレージバックエンドの間のトラフィックの暗号化を有効にすることで、データアクセスセキュリティを強化できます。

Astra Control Provisionerは、次の2種類のストレージバックエンドでKerberos暗号化をサポートします。

- *オンプレミスのONTAP *- Astra Control Provisionerは、Red Hat OpenShiftおよびアップストリームのKubernetesクラスタからオンプレミスのONTAPボリュームへのNFSv3およびNFSv4接続でKerberos暗号化をサポートします。
- * Azure NetApp Files *- Astra Control Provisionerは、アップストリームのKubernetesクラスタからAzure NetApp Filesボリュームへの NFSv4.1接続でKerberos暗号化をサポートします。

作成、削除、サイズ変更、スナップショット、クローン、 読み取り専用のクローンを作成し、NFS暗号化を使用するボリュームをインポートします。

オンプレミスの**ONTAP**ボリュームで転送中の**Kerberos**暗号化を設定

管理対象クラスタとオンプレミスのONTAPストレージバックエンドの間のストレージトラフィックに対してKerberos暗号化を有効にすることができます。

> オンプレミスのONTAPストレージバックエンドを使用するNFSトラフィックに対するKerberos暗号化は、 `ontap-nas` ストレージドライバ。

作業を開始する前に

- 次のことを確認します。 "Astra Control Provisionerを有効にしました" 管理対象クラスタ。
- にアクセスできることを確認します。 `tridentctl` ユーティリティ。
- ONTAPストレージバックエンドへの管理者アクセス権があることを確認します。
- ONTAPストレージバックエンドから共有するボリュームの名前を確認しておきます。
- NFSボリュームのKerberos暗号化をサポートするようにONTAP Storage VMを準備しておく必要があります。を参照してください "データ LIF で Kerberos を有効にします" 手順については、を参照し
- Kerberos暗号化で使用するNFSv4ボリュームが正しく設定されていることを確認します。『NetApp NFSv4ドメイン設定』のセクション（13ページ）を参照してください。 "『NetApp NFSv4 Enhancements and Best Practices Guide』"。

ONTAPエクスポートポリシーを追加または変更する

既存のONTAPエクスポートポリシーにルールを追加するか、ONTAP Storage VMのルートボリュームおよびアップストリームのKubernetesクラスタと共有するONTAPボリュームに対してKerberos暗号化をサポートする新しいエクスポートポリシーを作成する必要があります。追加するエクスポートポリシールールまたは新規に作成するエクスポートポリシーでは、次のアクセスプロトコルとアクセス権限がサポートされている必要があります。

アクセスプロトコル

NFS、NFSv3、およびNFSv4の各アクセスプロトコルを使用してエクスポートポリシーを設定します。

詳細を確認

ボリュームのニーズに応じて、次の3つのバージョンのいずれかを設定できます。

- * Kerberos 5 *-（認証と暗号化）
- * Kerberos 5i *-（ID保護による認証と暗号化）
- * Kerberos 5p *-（IDおよびプライバシー保護による認証および暗号化）

適切なアクセス権限を指定してONTAPエクスポートポリシールールを設定します。たとえば、Kerberos 5i暗号化とKerberos 5p暗号化が混在しているNFSボリュームをクラスタにマウントする場合は、次のアクセス設定を使用します。

を入力します	読み取り専用アクセス	読み取り/書き込みアクセス	スーパーユーザアクセス
「 UNIX 」	有効	有効	有効
Kerberos 5i	有効	有効	有効
Kerberos 5p	有効	有効	有効

ONTAPエクスポートポリシーおよびエクスポートポリシールールの作成方法については、次のドキュメントを参照してください。

- "エクスポートポリシーを作成する"
- "エクスポートポリシーにルールを追加する"

ストレージバックエンドの作成

Kerberos暗号化機能を含むAstra Control Provisionerストレージバックエンド構成を作成できます。

このタスクについて

Kerberos暗号化を設定するストレージバックエンド構成ファイルを作成するときに、`spec.nfsMountOptions` パラメータ：

- `spec.nfsMountOptions: sec=krb5` （認証と暗号化）
- `spec.nfsMountOptions: sec=krb5i` （ID保護による認証と暗号化）
- `spec.nfsMountOptions: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

Kerberosレベルを1つだけ指定してください。パラメータリストで複数のKerberos暗号化レベルを指定した場合は、最初のオプションのみが使用されます。

手順

1. 管理対象クラスタで、次の例を使用してストレージバックエンド構成ファイルを作成します。括弧<>の値は、環境の情報で置き換えます。

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. 前の手順で作成した構成ファイルを使用して、バックエンドを作成します。

```
tridentctl create backend -f <backend-configuration-file>
```

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs
```

構成ファイルで問題を特定して修正したら、 create コマンドを再度実行できます。

ストレージクラスを作成する。

ストレージクラスを作成して、Kerberos暗号化を使用してボリュームをプロビジョニングできます。

このタスクについて

ストレージクラスオブジェクトを作成するときに、を使用して3つの異なるバージョンのKerberos暗号化のいずれかを指定できます。 `mountOptions` パラメータ：

- `mountOptions: sec=krb5` （認証と暗号化）
- `mountOptions: sec=krb5i` （ID保護による認証と暗号化）
- `mountOptions: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

Kerberosレベルを1つだけ指定してください。パラメータリストで複数のKerberos暗号化レベルを指定した場合は、最初のオプションのみが使用されます。ストレージバックエンド構成で指定した暗号化レベルがストレージクラスオブジェクトで指定したレベルと異なる場合は、ストレージクラスオブジェクトが優先されます。

手順

1. 次の例を使用して、StorageClass Kubernetesオブジェクトを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
parameters:
  backendType: "ontap-nas"
  storagePools: "ontapnas_pool"
  trident.netapp.io/nasType: "nfs"
allowVolumeExpansion: True
```

2. ストレージクラスを作成します。

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. ストレージクラスが作成されていることを確認します。

```
kubectl get sc ontap-nas-sc
```

次のような出力が表示されます。

```
NAME            PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになり

ました。以下の手順を参照してください。 "ボリュームのプロビジョニング"。

Azure NetApp Filesボリュームでの転送中Kerberos暗号化の設定

管理対象クラスタと単一のAzure NetApp Filesストレージバックエンドまたは Azure NetApp Filesストレージバックエンドの仮想プールの間のストレージトラフィックに対してKerberos暗号化を有効にすることができます。

作業を開始する前に

- 管理対象のRed Hat OpenShiftクラスタでAstra Control Provisionerが有効になっていることを確認します。を参照してください "Astra Control Provisionerを有効にする" 手順については、を参照し
- にアクセスできることを確認します。 `tridentctl` ユーティリティ。
- 要件を確認し、次の手順に従って、Kerberos暗号化用のAzure NetApp Filesストレージバックエンドの準備が完了していることを確認します。 "Azure NetApp Files のドキュメント"。
- Kerberos暗号化で使用するNFSv4ボリュームが正しく設定されていることを確認します。『NetApp NFSv4ドメイン設定』のセクション（13ページ）を参照してください。 "『NetApp NFSv4 Enhancements and Best Practices Guide』"。

ストレージバックエンドの作成

Kerberos暗号化機能を含むAzure NetApp Filesストレージバックエンド構成を作成できます。

このタスクについて

Kerberos暗号化を設定するストレージバックエンド構成ファイルを作成する場合は、次の2つのレベルのいずれかで適用するように定義できます。

- ストレージバックエンドレベル*を使用して `spec.kerberos` フィールド
- 仮想プールレベル*を使用して `spec.storage.kerberos` フィールド

仮想プールレベルで構成を定義する場合、ストレージクラスのラベルを使用してプールが選択されます。

どちらのレベルでも、次の3つのバージョンのKerberos暗号化のいずれかを指定できます。

- `kerberos: sec=krb5` （認証と暗号化）
- `kerberos: sec=krb5i` （ID保護による認証と暗号化）
- `kerberos: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

手順

1. 管理対象クラスタで、ストレージバックエンドを定義する必要がある場所（ストレージバックエンドレベルまたは仮想プールレベル）に応じて、次のいずれかの例を使用してストレージバックエンド構成ファイルを作成します。括弧<>の値は、環境の情報で置き換えます。

ストレージバックエンドレベルの例

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-anf-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-anf-secret
```

仮想プールレベルの例

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-anf-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-anf-secret
```

2. 前の手順で作成した構成ファイルを使用して、バックエンドを作成します。

```
tridentctl create backend -f <backend-configuration-file>
```

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs
```

構成ファイルで問題を特定して修正したら、 create コマンドを再度実行できます。

ストレージクラスを作成する。

ストレージクラスを作成して、Kerberos暗号化を使用してボリュームをプロビジョニングできます。

手順

1. 次の例を使用して、StorageClass Kubernetesオブジェクトを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "nfs"
  selector: "type=encryption"
```

2. ストレージクラスを作成します。

```
kubectl create -f sample-input/storage-class-anf-sc-nfs.yaml
```

3. ストレージクラスが作成されていることを確認します。

```
kubectl get sc anf-sc-nfs
```

次のような出力が表示されます。

```
NAME          PROVISIONER              AGE
anf-sc-nfs     csi.trident.netapp.io    15h
```

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになりました。以下の手順を参照してください。 "ボリュームのプロビジョニング"。

Snapshotを使用したボリュームデータのリカバリ

Astra Control Provisionerを使用すると、Snapshotからボリュームをインプレースですばやくリストアできます。 `TridentActionSnapshotRestore` （TASR）CR。このCRはKubernetesの必須アクションとして機能し、処理の完了後も維持されません。

Astra Control Provisionerは、上でのSnapshotのリストアをサポートします。 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `gcp-cvs`および `solidfire-san` ドライバ。

作業を開始する前に

バインドされたPVCと使用可能なボリュームSnapshotが必要です。

- PVCステータスがバインドされていることを確認します。

```
kubectl get pvc
```

- ボリュームSnapshotを使用する準備が完了していることを確認します。

```
kubectl get vs
```

手順

1. TASR CRを作成します。次に、PVC用のCRを作成する例を示します。 `pvc1` ボリュームSnapshot `pvc1-snapshot`。

```
cat tasr-pvc1-snapshot.yaml

apiVersion: v1
kind: TridentActionSnapshotRestore
metadata:
  name: this-doesnt-matter
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. スナップショットからリストアするにはCRを適用します。この例では、Snapshotからリストアします。 `pvc1`。

```
kubectl create -f tasr-pvc1-snapshot.yaml

tridentactionsnapshotrestore.trident.netapp.io/this-doesnt-matter
created
```

結果

Astra Control ProvisionerがSnapshotからデータをリストアします。Snapshotのリストアステータスを確認できます。

```
kubectl get tasr -o yaml

apiVersion: v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: this-doesnt-matter
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- ほとんどの場合、障害が発生したときにAstra Control Provisionerで処理が自動的に再試行されません。この操作を再度実行する必要があります。
- 管理者アクセス権を持たないKubernetesユーザは、アプリケーションネームスペースにTASR CRを作成するために、管理者から権限を付与されなければならない場合があります。

SnapMirrorによるボリュームのレプリケート

Astra Control Provisionerを使用すると、ディザスタリカバリ用にデータをレプリケートするために、一方のクラスタのソースボリュームとピアクラスタのデスティネーションボリュームの間にミラー関係を作成できます。名前空間カスタムリソース定義（CRD）を使用して、次の操作を実行できます。

- ボリューム（PVC）間のミラー関係を作成する
- ボリューム間のミラー関係の削除
- ミラー関係を解除する
- 災害時（フェイルオーバー）にセカンダリボリュームを昇格する
- クラスタからクラスタへのアプリケーションのロスレス移行の実行（計画的なフェイルオーバーまたは移行時）

レプリケーションの前提条件

作業を開始する前に、次の前提条件を満たしていることを確認してください。

ONTAP クラスタ

- * Astra Control Provisioner *：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、Astra Control Provisionerバージョン23.10以降が必要です。
- 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。を参照してください "ONTAPのSnapMirrorライセンスの概要" を参照してください。

ピアリング

- *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。を参照してください "クラスタと SVM のピアリングの概要" を参照してください。

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

- * Astra Control ProvisionerとSVM *：ピア関係にあるリモートSVMがデスティネーションクラスタのAstra Control Provisionerで使用可能である必要があります。

サポートされるドライバ

- ボリュームレプリケーションは、ONTAP-NASドライバとONTAP-SANドライバでサポートされます。

ミラー**PVC**の作成

以下の手順に従って、CRDの例を使用してプライマリボリュームとセカンダリボリュームの間にミラー関係を作成します。

手順

1. プライマリKubernetesクラスタで次の手順を実行します。
   a. を使用してStorageClassオブジェクトを作成します。 `trident.netapp.io/replication: true` パラメータ

      例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 以前に作成したStorageClassを使用してPVCを作成します。

例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. ローカル情報を含むMirrorRelationship CRを作成します。

例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Astra Control Provisionerは、ボリュームの内部情報とボリュームの現在のデータ保護（DP）の状態をフェッチし、MirrorRelationshipのstatusフィールドに値を入力します。

d. TridentMirrorRelationship CRを取得して、PVCの内部名とSVMを取得します。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. セカンダリKubernetesクラスタで次の手順を実行します。
   a. trident.netapp.io/replication: trueパラメータを使用してStorageClassを作成します。

      例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

   b. デスティネーションとソースの情報を含むMirrorRelationship CRを作成します。

      例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Astra Control Provisionerは、設定された関係ポリシー名（ONTAPの場合はデフォルト）を使用してSnapMirror関係を作成し、初期化します。

c. セカンダリ（SnapMirrorデスティネーション）として機能するStorageClassを作成してPVCを作成します。

例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Astra Control ProvisionerはTridentMirrorRelationship CRDを確認し、関係が存在しない場合はボリュームの作成に失敗します。関係が存在する場合は、Astra Control Provisionerによって、新しいFlexVolがMirrorRelationshipで定義されているリモートSVMとピア関係にあるSVMに配置されます。

ボリュームレプリケーションの状態

Trident Mirror Relationship（TMR）は、PVC間のレプリケーション関係の一端を表すCRDです。デスティネーションTMRの状態はで、Astra Control Provisionerに必要な状態が示されます。宛先TMRの状態は次のとおりです。

- 確立済み：ローカルPVCはミラー関係のデスティネーションボリュームであり、これは新しい関係です。
- 昇格：ローカルPVCはReadWriteでマウント可能であり、ミラー関係は現在有効ではありません。
- * reestablished *：ローカルPVCはミラー関係のデスティネーションボリュームであり、以前はそのミラー関係に含まれていました。
  - デスティネーションボリュームはデスティネーションボリュームの内容を上書きするため、ソースボリュームとの関係が確立されたことがある場合は、reestablished状態を使用する必要があります。
  - ボリュームが以前にソースとの関係になかった場合、再確立状態は失敗します。

計画外フェールオーバー時にセカンダリ**PVC**を昇格する

セカンダリKubernetesクラスタで次の手順を実行します。

- TridentMirrorRelationshipの_spec.state_フィールド を次のように更新します。 `promoted`。

計画的フェイルオーバー中にセカンダリ**PVC**を昇格

計画的フェイルオーバー（移行）中に、次の手順を実行してセカンダリPVCをプロモートします。

手順

1. プライマリKubernetesクラスタでPVCのSnapshotを作成し、Snapshotが作成されるまで待ちます。
2. プライマリKubernetesクラスタで、SnapshotInfo CRを作成して内部の詳細を取得します。

   例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. セカンダリKubernetesクラスタで、_TridentMirrorRelationship_CRの_spec.state_フィールドを_promoted_に更新し、_spec.promotedSnapshotHandle_をSnapshotのinternalNameにします。
4. セカンダリKubernetesクラスタで、TridentMirrorRelationshipのステータス（status.stateフィールド）がPromotedになっていることを確認します。

フェイルオーバー後にミラー関係をリストアする

ミラー関係をリストアする前に、新しいプライマリとして作成する側を選択します。

手順

1. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.remoteVolumeHandle_fieldの値が更新されていることを確認します。
2. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.mirror_fieldを `reestablished`。

その他の処理

Astra Control Provisionerでは、プライマリボリュームとセカンダリボリュームに対する次の処理がサポートされます。

新しいセカンダリ**PVC**へのプライマリ**PVC**の複製

プライマリPVCとセカンダリPVCがすでに存在していることを確認します。

手順

1. PersistentVolumeClaim CRDとTridentMirrorRelationship CRDを、確立されたセカンダリ（デスティネーション）クラスタから削除します。
2. プライマリ（ソース）クラスタからTridentMirrorRelationship CRDを削除します。
3. 確立する新しいセカンダリ（デスティネーション）PVC用に、プライマリ（ソース）クラスタに新しいTridentMirrorRelationship CRDを作成します。

ミラー、プライマリ、またはセカンダリ**PVC**のサイズ変更

PVCは通常どおりサイズ変更できます。データ量が現在のサイズを超えると、ONTAPは自動的に宛先フレフxolを拡張します。

PVCからのレプリケーションの削除

レプリケーションを削除するには、現在のセカンダリボリュームで次のいずれかの操作を実行します。

- セカンダリPVCのMirrorRelationshipを削除します。これにより、レプリケーション関係が解除されます。
- または、spec.stateフィールドを_promoted_に更新します。

（以前にミラーリングされていた）**PVC**の削除

Astra Control Provisionerは、ボリュームの削除を試行する前に、レプリケートされたPVCをチェックし、レプリケーション関係を解放します。

TMRの削除

ミラー関係の一方のTMRを削除すると、Astra Control Provisionerが削除を完了する前に、残りのTMRが_promoted_stateに移行します。削除対象として選択したTMRがすでに_promoted_stateである場合は、既存のミラー関係は存在せず、TMRが削除され、Astra Control ProvisionerがローカルPVCを_ReadWrite_に昇格します。この削除により、ONTAP内のローカルボリュームのSnapMirrorメタデータが解放されます。このボリュームを今後ミラー関係で使用する場合は、新しいミラー関係を作成するときに、レプリケーション状態が_established_volumeである新しいTMRを使用する必要があります。

ONTAPがオンラインのときにミラー関係を更新

ミラー関係は、確立後にいつでも更新できます。を使用できます `state: promoted` または `state: reestablished` 関係を更新するフィールド。
デスティネーションボリュームを通常のReadWriteボリュームに昇格する場合は、_promotedSnapshotHandle_を使用して、現在のボリュームのリストア先となる特定のSnapshotを指定できます。

ONTAPがオフラインの場合にミラー関係を更新

CRDを使用すると、Astra ControlからONTAPクラスタに直接接続せずにSnapMirror更新を実行できます。次のTridentActionMirrorUpdateの形式例を参照してください。

例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` TridentActionMirrorUpdate CRDの状態を反映します。*Succeeded*、*In Progress*、_Failed_ のいずれかの値を指定できます。