SnapMirrorによるボリュームのレプリケート

Astra Trident

NetApp
March 05, 2026

目次

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。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

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。詳細については、を参照してください "クラスタと SVM のピアリングの概要" 。

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

- * Astra Control Provisioner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. パラメータを指定してStorageClassオブジェクトを作成し `trident.netapp.io/replication: true` ます。

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をに更新します `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。フィールドまたはフィールドを使用して関係を更新できます `state: promoted` `state: reestablished` 。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。

Astra Control Provisionerを有効にする

Tridentバージョン23.10以降には、Astra Control Provisionerを使用するオプションが用意されています。このオプションを使用すると、ライセンスを取得したAstra Controlユーザは、高度なストレージプロビジョニング機能にアクセスできます。Astra Control Provisionerは、Astra Trident CSIベースの標準機能に加えて、この拡張機能を提供します。この手順を使用して、Astra Control Provisionerを有効にしてインストールできます。

Astra Control Serviceのサブスクリプションには、Astra Control Provisionerの使用ライセンスが自動的に含まれます。

今後のAstra Controlの更新では、Astra Control ProvisionerがAstra Tridentの代わりにストレージプロビジョニングツールおよびオーケストレータとして使用され、Astra Controlでは必須となります。そのため、Astra ControlのユーザはAstra Control Provisionerを有効にすることを強く推奨します。Astra Tridentは引き続きオープンソースであり、NetAppの新しいCSIやその他の機能でリリース、メンテナンス、サポート、更新されます。

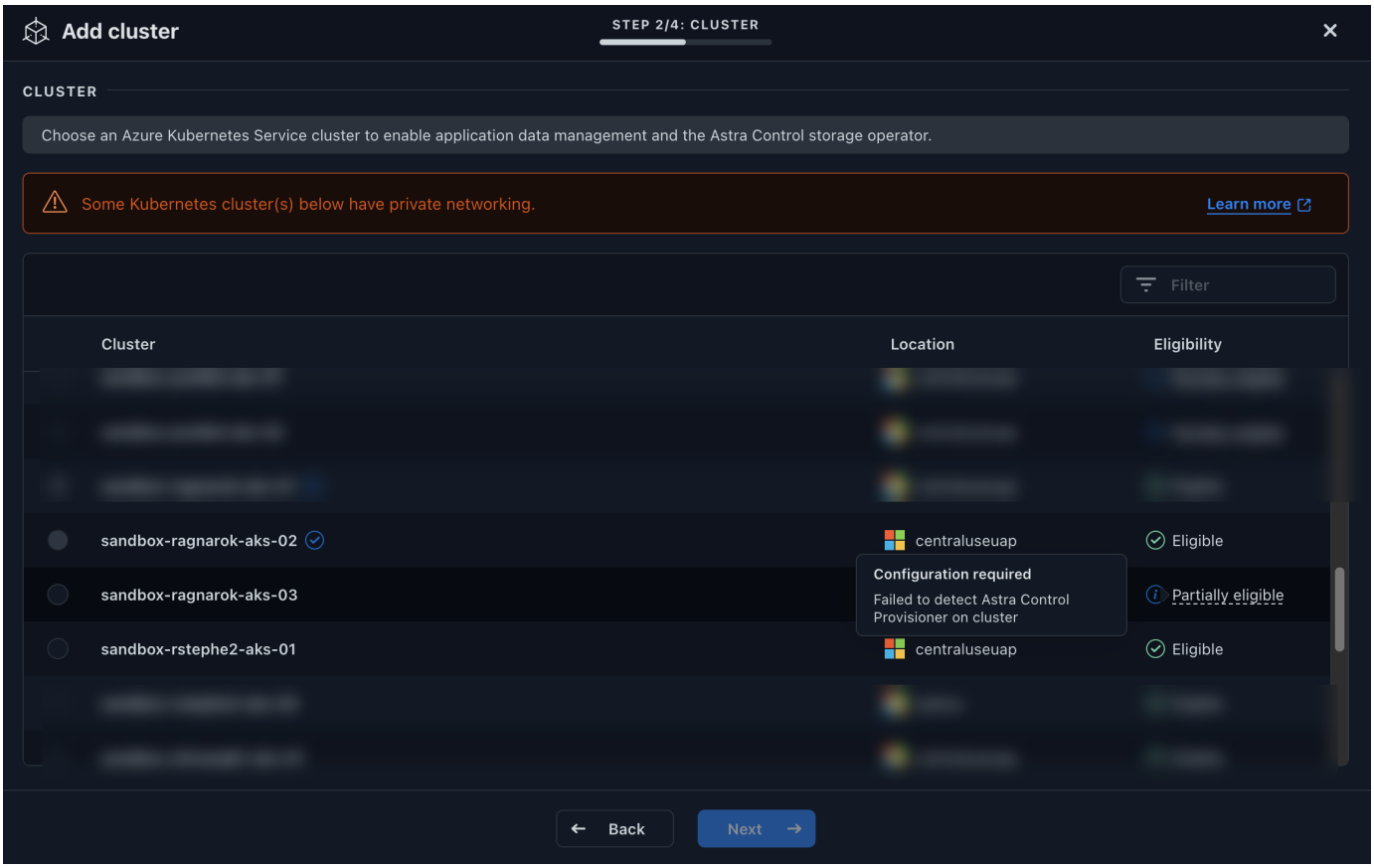
Astra Control Provisionerを有効にする必要があるかどうかを確認する方法を教えてください。

Astra Tridentがインストールされていないクラスタを Astra Control Serviceに追加すると、そのクラスタはとマークされ `Eligible`ます。設定が完了すると "Astra Controlへのクラスタの追加"、Astra Control Provisionerが自動的に有効になります。

クラスタがマークされていない場合は `Eligible`、次のいずれかの理由でマークされ `Partially eligible` ます。

- 古いバージョンのAstra Tridentを使用している
- Provisionerオプションが有効になっていないAstra Trident 23.10を使用している
- このクラスタタイプでは自動有効化が許可されていません

ケースの場合 `Partially eligible` は、以下の手順に従ってクラスタのAstra Control Provisionerを手動で有効にしてください。



Astra Control Provisionerを有効にする前に

Astra Control Provisionerが設定されていないAstra Tridentをすでに使用していて、Astra Control Provisionerを有効にする場合は、まず次の手順を実行します。

- * Astra Tridentがインストールされている場合は、バージョンが4リリース期間内であることを確認してください*：Astra Tridentがバージョン24.02の4リリース期間内であれば、Astra Control Provisionerを使用してAstra Trident 24.02への直接アップグレードを実行できます。たとえば、Astra Trident 23.04から24.02に直接アップグレードできます。
- クラスタに**AMD64**システムアーキテクチャがあることを確認する：Astra Control Provisionerイメージは AMD64とARM64の両方のCPUアーキテクチャで提供されますが、Astra Controlでサポートされるのは AMD64のみです。

手順

1. NetApp Astra Controlイメージのレジストリにアクセスします。
   a. Astra Control Service UIにログオンし、Astra ControlアカウントIDを記録します。
      i. ページの右上にある図のアイコンを選択します。
      ii. [API access*]を選択します。
      iii. アカウントIDを書き留めます。
   b. 同じページから* APIトークンの生成*を選択し、APIトークン文字列をクリップボードにコピーしてエディターに保存します。

c. 任意の方法でAstra Controlレジストリにログインします。

```
docker login cr.astra.netapp.io -u <account-id> -p <api-token>
```

```
crane auth login cr.astra.netapp.io -u <account-id> -p <api-token>
```

2. (カスタムレジストリのみ)イメージをカスタムレジストリに移動するには、次の手順に従います。レジストリを使用していない場合は、のTrident Operatorの手順に従います 次のセクション。

> 次のコマンドには、Dockerの代わりにPodmanを使用できます。Windows環境を使用している場合は、PowerShellを推奨します。

Docker です

a. Astra Control Provisionerのイメージをレジストリから取得します。

> プルされたイメージは複数のプラットフォームをサポートせず、Linux AMD64など、イメージをプルしたホストと同じプラットフォームのみをサポートします。

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0
--platform <cluster platform>
```

例：

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0
--platform linux/amd64
```

b. 画像にタグを付けます。

```
docker tag cr.astra.netapp.io/astra/trident-acp:24.02.0
<my_custom_registry>/trident-acp:24.02.0
```

c. イメージをカスタムレジストリにプッシュします。

```
docker push <my_custom_registry>/trident-acp:24.02.0
```

クレーン

a. Astra Control Provisionerのマニフェストをカスタムレジストリにコピーします。

```
crane copy cr.astra.netapp.io/astra/trident-acp:24.02.0
<my_custom_registry>/trident-acp:24.02.0
```

3. 元のAstra Tridentインストール方法でを使用していたかどうかを確認します。
4. 以前使用したインストール方法を使用して、Astra TridentでAstra Control Provisionerを有効にします。

Astra Trident運用者

a. "Astra Tridentインストーラをダウンロードして展開"です。

b. Astra Tridentをまだインストールしていない場合、または元のAstra Trident環境からオペレータを削除した場合は、次の手順を実行します。

   i. CRDを作成します。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.1
6.yaml
```

   ii. Tridentネームスペースを作成 (`kubectl create namespace trident`）またはTridentネームスペースが残っていることを確認し (`kubectl get all -n trident`ます。ネームスペースが削除されている場合は、もう一度作成します。

c. Astra Tridentを24.02.0に更新：

> クラスタでKubernetes 1.24以前を実行している場合は、を使用します `bundle_pre_1_25.yaml`。クラスタでKubernetes 1.25以降を実行している場合は、を使用します `bundle_post_1_25.yaml`。

```
kubectl -n trident apply -f trident-installer/deploy/<bundle-
name.yaml>
```

d. Astra Tridentが実行されていることを確認します。

```
kubectl get torc -n trident
```

応答：

```
NAME      AGE
trident   21m
```

e. [pull-secrets]]シークレットを使用するレジストリがある場合は、Astra Control Provisionerイメージの取得に使用するシークレットを作成します。

```
kubectl create secret docker-registry <secret_name> -n trident
--docker-server=<my_custom_registry> --docker-username=<username>
--docker-password=<token>
```

f. TridentOrchestrator CRを編集し、次の編集を行います。

```
kubectl edit torc trident -n trident
```

i. Astra Tridentイメージのカスタムのレジストリの場所を設定するか、Astra Controlレジストリまたはから取得し (`tridentImage: <my_custom_registry>/trident:24.02.0` `tridentImage: netapp/trident:24.02.0`ます）。

ii. Astra Control Provisionerを有効にし (`enableACP: true`ます）。

iii. Astra Control Provisionerイメージのカスタムのレジストリの場所を設定するか、Astra Controlレジストリまたはから取得し (`acpImage: <my_custom_registry>/trident-acp:24.02.0` `acpImage: cr.astra.netapp.io/astra/trident-acp:24.02.0`ます。

iv. この手順で以前に確立した場合は 画像プルシークレット 、ここで設定でき (`imagePullSecrets: - <secret_name>`ます）。前の手順で設定した名前と同じシークレット名を使用します。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  tridentImage: <registry>/trident:24.02.0
  enableACP: true
  acpImage: <registry>/trident-acp:24.02.0
  imagePullSecrets:
  - <secret_name>
```

g. ファイルを保存して終了します。導入プロセスが自動的に開始されます。

h. operator、deployment、およびReplicaSetsが作成されていることを確認します。

```
kubectl get all -n trident
```

Kubernetes クラスタには、オペレータのインスタンスが * 1 つしか存在しないようにしてください。Astra Tridentオペレータを複数の環境に導入することは避けてください。

i. コンテナが実行されていて、ステータスがになっていることを確認し `trident-acp` `acpVersion 24.02.0` `Installed`ます。

```
kubectl get torc -o yaml
```

応答：

```
status:
  acpVersion: 24.02.0
  currentInstallationParams:
    ...
    acpImage: <registry>/trident-acp:24.02.0
    enableACP: "true"
    ...
  ...
  status: Installed
```

Tridentctl

a. "Astra Tridentインストーラをダウンロードして展開"です。

b. "既存のAstra Tridentがある場合は、そのTridentをホストしているクラスタからアンインストール"です。

c. Astra Control Provisionerを有効にしてAstra Tridentをインストール (`--enable-acp=true`）：

```
./tridentctl -n trident install --enable-acp=true --acp
-image=mycustomregistry/trident-acp:24.02
```

d. Astra Control Provisionerが有効になっていることを確認します。

```
./tridentctl -n trident version
```

応答：

```
+----------------+----------------+-------------+ | SERVER
VERSION | CLIENT VERSION | ACP VERSION | +----------------
+----------------+-------------+ | 24.02.0 | 24.02.0 | 24.02.0. |
+----------------+----------------+-------------+
```

Helm

a. Astra Trident 23.07.1以前がインストールされている場合は "をアンインストールします" 、オペレータとその他のコンポーネント。

b. Kubernetesクラスタが1.24以前を実行している場合は、pspを削除します。

```
kubectl delete psp tridentoperatorpod
```

c. Astra Trident Helmリポジトリを追加します。

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

d. Helmチャートを更新します。

```
helm repo update netapp-trident
```

応答：

```
Hang tight while we grab the latest from your chart
repositories...
...Successfully got an update from the "netapp-trident" chart
repository
Update Complete. □Happy Helming!□
```

e. 画像を一覧表示します。

```
./tridentctl images -n trident
```

応答：

```
| v1.28.0            | netapp/trident:24.02.0|
|                    | docker.io/netapp/trident-
autosupport:24.02|
|                    | registry.k8s.io/sig-storage/csi-
provisioner:v4.0.0|
|                    | registry.k8s.io/sig-storage/csi-
attacher:v4.5.0|
|                    | registry.k8s.io/sig-storage/csi-
resizer:v1.9.3|
|                    | registry.k8s.io/sig-storage/csi-
snapshotter:v6.3.3|
|                    | registry.k8s.io/sig-storage/csi-node-
driver-registrar:v2.10.0 |
|                    | netapp/trident-operator:24.02.0 (optional)
```

f. trident-operator 24.02.0が使用可能であることを確認します。

```
helm search repo netapp-trident/trident-operator --versions
```

応答：

```
NAME                                CHART VERSION   APP VERSION
DESCRIPTION
netapp-trident/trident-operator 100.2402.0      24.02.0         A
```

g. これらの設定を含む次のいずれかのオプションを使用して `helm install` 実行します。

- 導入場所の名前
- Astra Tridentバージョン
- Astra Control Provisionerの名前の画像
- プロビジョニングツールを有効にするフラグ
- （任意）ローカルレジストリパス。ローカルレジストリを使用している場合、は "Tridentの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。
- Tridentネームスペース

オプション（ **Options** ）

◦ レジストリなしのイメージ

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=cr.astra.netapp.io/astra/trident-
acp:24.02.0 --set enableACP=true --set operatorImage=netapp/trident-
operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

◦ 1つまたは複数のレジストリ内の画像

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=<your-registry>:<acp image> --set
enableACP=true --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=netapp/trident-operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

を使用できます `helm list` 名前、ネームスペース、グラフ、ステータス、アプリケーションバージョンなどのインストールの詳細を確認するには、次の手順を実行します。とリビジョン番号。

Helmを使用したTridentの導入で問題が発生した場合は、次のコマンドを実行してAstra Tridentを完全にアンインストールします。

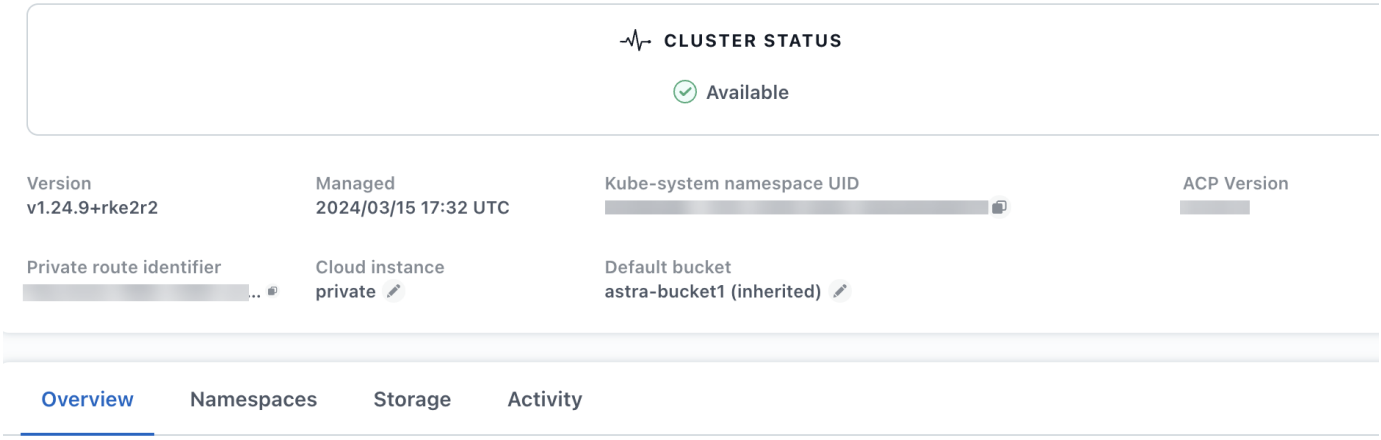
```
./tridentctl uninstall -n trident
```

- Astra Control Provisionerを再度有効にする前に、アンインストールの一環として実行しないでください。 "Astra TridentのCRDを完全に削除"

結果

Astra Control Provisionerの機能が有効になっており、実行しているバージョンで使用可能なすべての機能を使用できます。

Astra Control Provisionerをインストールすると、Astra Control UIでプロビジョニングツールをホストしているクラスタに、フィールドではなく、現在インストールされているバージョン番号が表示され `ACP version` `Trident version` ます。



を参照してください。

- "Astra Tridentのアップグレードに関するドキュメント"