



# ボリュームのプロビジョニングと管理

## Trident

NetApp  
January 14, 2026

# 目次

ボリュームのプロビジョニングと管理	1
ボリュームをプロビジョニングする	1
概要	1
PVCの作成	1
ボリュームを展開します	5
iSCSI ボリュームを展開します	5
FC ボリュームを拡張します	9
NFS ボリュームを拡張します	13
ボリュームをインポート	16
概要と考慮事項	16
ボリュームをインポートします	17
例	18
ボリュームの名前とラベルをカスタマイズする	24
開始する前に	24
制限事項	24
カスタマイズ可能なボリューム名の主な動作	24
名前テンプレートとラベルを使用したバックエンド構成の例	25
名前テンプレートの例	26
考慮すべきポイント	27
ネームスペース間でNFSボリュームを共有します	27
特徴	27
クイックスタート	28
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を設定します	29
共有ボリュームを削除	30
下位ボリュームのクエリに使用 <code>tridentctl get</code>	30
制限事項	31
詳細情報	31
ネームスペース全体でボリュームをクローニング	31
前提条件	31
クイックスタート	31
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を設定します	32
制限事項	34
SnapMirrorによるボリュームのレプリケート	34
レプリケーションの前提条件	34
ミラーPVCの作成	34
ボリュームレプリケーションの状態	38
計画外フェールオーバー時にセカンダリPVCを昇格する	38
計画的フェイルオーバー中にセカンダリPVCを昇格	38

フェイルオーバー後にミラー関係をリストアする	39
その他の処理	39
ONTAPがオンラインのときにミラー関係を更新	40
ONTAPがオフラインの場合にミラー関係を更新	40
CSI トポロジを使用します	40
概要	40
手順 1：トポロジ対応バックエンドを作成する	42
手順 2：トポロジを認識するストレージクラスを定義する	44
ステップ 3：PVC を作成して使用する	45
バックエンドを更新して含める <code>supportedTopologies</code>	48
詳細情報	48
スナップショットを操作します	48
概要	48
ボリューム Snapshot を作成します	49
ボリュームSnapshotからPVCを作成	50
ボリュームSnapshotのインポート	51
Snapshotを使用したボリュームデータのリカバリ	53
Snapshotからのインプレースボリュームのリストア	53
Snapshotが関連付けられているPVを削除する	55
ボリュームSnapshotコントローラの導入	55
関連リンク	56

# ボリュームのプロビジョニングと管理

## ボリュームをプロビジョニングする

設定したKubernetes StorageClassを使用してPVへのアクセスを要求するPersistentVolumeClaim (PVC) を作成します。その後、PVをポッドにマウントできます。

### 概要

```
https://kubernetes.io/docs/concepts/storage/persistent-volumes["PersistentVolumeClaim_"] (PVC) は、クラスタ上のPersistentVolumeへのアクセス要求です。
```

PVCは、特定のサイズまたはアクセスモードのストレージを要求するように設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVCを作成したら、ボリュームをポッドにマウントできます。

### PVCの作成

#### 手順

##### 1. PVCを作成

```
kubectl create -f pvc.yaml
```

##### 2. PVCステータスを確認します。

```
kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
pvc-storage	Bound	pv-name	1Gi	RWO		5m

##### 1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```



進捗状況はを使用して監視でき `kubectl get pod --watch` ます。

2. ボリュームがにマウントされていることを確認します /my/mount/path。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

マニフェストの例

## PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

### RWOアクセスを備えたPVC

この例は、という名前のStorageClassに関連付けられたRWOアクセスを持つ基本的なPVCを示しています basic-csi。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

### NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています protection-gold。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san-nvme
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 300Mi
  storageClassName: protection-gold
```

## PODマニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

### 基本構成

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
        claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
  volumeMounts:
    - mountPath: "/my/mount/path"
      name: storage
```

### NVMe/TCPの基本構成

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については PersistentVolumeClaim、を参照してください["Kubernetes オブジェクトと Trident](#)

オブジェクト"。

## ボリュームを展開します

Tridentを使用すると、Kubernetesユーザは作成後にボリュームを拡張できます。ここでは、iSCSI、NFS、およびFCのボリュームを拡張するために必要な設定について説明します。

### iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、iSCSI Persistent Volume ( PV ) を拡張できます。



iSCSIボリュームの拡張は、`ontap-san-economy`、`solidfire-san`ドライバでサポートされ`ontap-san`であり、Kubernetes 1.16以降が必要です。

手順 1：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して、フィールドをに`true`設定し`allowVolumeExpansion`ます。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存のStorageClassの場合は、パラメータを含めるように編集します`allowVolumeExpansion`。

手順 2：作成した **StorageClass** を使用して **PVC** を作成します

PVC定義を編集し、を更新して、`spec.resources.requests.storage`新しく希望するサイズ（元のサイズよりも大きくなればなりません）を反映させます。

```
cat pvc-ontapsan.yaml
```

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san

```

Tridentは永続的ボリューム (PV) を作成し、この永続的ボリューム要求 (PVC) に関連付けます。

```

kubectl get pvc
NAME      STATUS      VOLUME                                     CAPACITY
ACCESS MODES      STORAGECLASS      AGE
san-pvc    Bound      pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO          ontap-san      8s

kubectl get pv
NAME          CAPACITY      ACCESS MODES
RECLAIM POLICY      STATUS      CLAIM      STORAGECLASS      REASON      AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi      RWO
Delete      Bound      default/san-pvc      ontap-san      10s

```

### 手順 3 : PVC を接続するポッドを定義します

サイズを変更するポッドにPVを接続します。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。

- PVがポッドに接続されている場合、Tridentはストレージバックエンド上のボリュームを拡張し、デバイスを再スキャンして、ファイルシステムのサイズを変更します。
- 接続されていないPVのサイズを変更しようとすると、Tridentはストレージバックエンド上のボリュームを拡張します。PVC がポッドにバインドされると、Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、Kubernetes は PVC サイズを更新します。

この例では、を使用するポッドが作成されて `san-pvc` います。

```
kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
ubuntu-pod    1/1     Running   0          65s

kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
                pv.kubernetes.io/bound-by-controller: yes
                volume.beta.kubernetes.io/storage-provisioner:
                csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:   ubuntu-pod
```

#### ステップ4：PVを拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集してを2Giに更新します  
spec.resources.requests.storage。

```
kubectl edit pvc san-pvc
```

```

# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
  # ...

```

## 手順5：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正常に機能したことを検証できます。

```

kubectl get pvc san-pvc
NAME      STATUS      VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS   AGE
san-pvc    Bound      pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO          ontap-san    11m

kubectl get pv
NAME                                     CAPACITY      ACCESS MODES
RECLAIM POLICY  STATUS      CLAIM      STORAGECLASS      REASON      AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi          RWO
Delete          Bound      default/san-pvc  ontap-san          12m

tridentctl get volumes -n trident
+-----+-----+-----+
+-----+-----+-----+
|           NAME          |  SIZE   | STORAGE CLASS | 
PROTOCOL |           BACKEND UUID      | STATE   | MANAGED | 
+-----+-----+-----+
+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san      | 
block     | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online  | true      | 
+-----+-----+-----+
+-----+-----+-----+

```

## FC ボリュームを拡張します

CSIプロビジョニングツールを使用して、FC永続ボリューム (PV) を拡張できます。



FCボリュームの拡張はドライバでサポートされ `ontap-san` しており、Kubernetes 1.16以降が必要です。

手順 1：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して、フィールドをに `true` 設定し `allowVolumeExpansion` ます。

```
cat storageclass-ontapsan.yaml
```

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True

```

既存のStorageClassの場合は、パラメータを含めるように編集します `allowVolumeExpansion`。

## 手順 2：作成した StorageClass を使用して PVC を作成します

PVC定義を編集し、を更新して、`spec.resources.requests.storage`新しく希望するサイズ（元のサイズよりも大きくなればなりません）を反映させます。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続的ボリューム (PV) を作成し、この永続的ボリューム要求 (PVC) に関連付けます。

```
kubectl get pvc
NAME      STATUS      VOLUME                                     CAPACITY
ACCESS MODES      STORAGECLASS      AGE
san-pvc    Bound      pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO          ontap-san          8s

kubectl get pv
NAME                                     CAPACITY      ACCESS MODES
RECLAIM POLICY      STATUS      CLAIM      STORAGECLASS      REASON      AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi          RWO
Delete          Bound      default/san-pvc      ontap-san          10s
```

## 手順 3：PVC を接続するポッドを定義します

サイズを変更するポッドにPVを接続します。FC PVのサイズを変更する場合は、次の2つのシナリオが考えられます。

- PVがポッドに接続されている場合、Tridentはストレージバックエンド上のボリュームを拡張し、デバイスを再スキャンして、ファイルシステムのサイズを変更します。
- 接続されていないPVのサイズを変更しようとすると、Tridentはストレージバックエンド上のボリュームを拡張します。PVCがポッドにバインドされると、Tridentはデバイスを再スキャンし、ファイルシステム

ムのサイズを変更します。展開操作が正常に完了すると、Kubernetes は PVC サイズを更新します。

この例では、を使用するポッドが作成されて `san-pvc` います。

```
kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
ubuntu-pod    1/1     Running   0          65s

kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
                pv.kubernetes.io/bound-by-controller: yes
                volume.beta.kubernetes.io/storage-provisioner:
                csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:   ubuntu-pod
```

#### ステップ4：PVを拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集してを2Giに更新します  
spec.resources.requests.storage。

```
kubectl edit pvc san-pvc
```

```

# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
# ...

```

## 手順5：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正常に機能したことを検証できます。

```

kubectl get pvc san-pvc
NAME      STATUS      VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS   AGE
san-pvc   Bound      pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO          ontap-san   11m

kubectl get pv
NAME                                     CAPACITY      ACCESS MODES
RECLAIM POLICY  STATUS      CLAIM      STORAGECLASS      REASON      AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi          RWO
Delete          Bound      default/san-pvc  ontap-san           12m

tridentctl get volumes -n trident
+-----+-----+-----+
+-----+-----+-----+
|           NAME          |  SIZE   | STORAGE CLASS |
PROTOCOL |           BACKEND UUID      | STATE   | MANAGED   |
+-----+-----+-----+
+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san    |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online  | true      |
+-----+-----+-----+
+-----+-----+-----+

```

## NFS ボリュームを拡張します

Tridentでは、`ontap-nas-economy` `ontap-nas-flexgroup`、`gcp-cvs` `azure-netapp-files`、およびバックエンドでプロビジョニングされるNFS PVのボリューム拡張がサポートされます`ontap-nas`。

手順1：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PVのサイズを変更するには、管理者はまず、フィールドを`true`設定してボリュームの拡張を許可するようにストレージクラスを設定する必要があります。`allowVolumeExpansion`

```
cat storageclass-ontapnas.yaml
```

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true

```

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編集するだけでボリュームを拡張できます `kubectl edit storageclass`。

## 手順2：作成した **StorageClass** を使用して **PVC** を作成します

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

TridentはこのPVC用に20MiBのNFS PVを作成する必要があります。

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY      ACCESS MODES  STORAGECLASS      AGE
ontapnas20mb  Bound     pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  20Mi
RWO           ontapnas   9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME
RECLAIM POLICY  STATUS    CLAIM
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  20Mi      RWO
Delete        Bound     default/ontapnas20mb  ontapnas
2m42s
```

## ステップ3：PVを拡張する

新しく作成した20MiB PVのサイズを1GiBに変更するには、PVCを編集して1GiBに設定し`spec.resources.requests.storage`ます。

```
kubectl edit pvc ontapnas20mb
```

```

# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
    - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...

```

#### 手順4：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、サイズ変更が正しく機能したかどうかを検証できます。

```

kubectl get pvc ontapnas20mb
NAME           STATUS  VOLUME
CAPACITY      ACCESS MODES  STORAGECLASS      AGE
ontapnas20mb  Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi
RWO           ontapnas  4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME
RECLAIM POLICY  STATUS  CLAIM
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi      RWO
Delete        Bound    default/ontapnas20mb  ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----+-----+-----+
+-----+-----+-----+-----+
|           NAME           |  SIZE  | STORAGE CLASS  |
PROTOCOL |           BACKEND UUID           | STATE  | MANAGED  |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

## ボリュームをインポート

を使用して、既存のストレージボリュームをKubernetes PVとしてインポートできます  
tridentctl import。

### 概要と考慮事項

Tridentにボリュームをインポートする目的は次のとおりです。

- ・アプリケーションをコンテナ化し、既存のデータセットを再利用する
- ・一時的なアプリケーションにはデータセットのクローンを使用
- ・障害が発生したKubernetesクラスタを再構築します
- ・ディザスタリカバリ時にアプリケーションデータを移行

### 考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- ・Tridentでインポートできるのは、RW（読み取り/書き込み）タイプのONTAPボリュームのみです。DP（

データ保護) タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをTridentにインポートする前に、ミラー関係を解除する必要があります。

- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。



Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- PVCで指定する必要がありますが、`StorageClass` Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、PVはClaimRefを使用してPVCに作成されます。
  - 再利用ポリシーは、PVでは最初に`retain`と設定されています。KubernetesがPVCとPVを正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合、`delete` PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、TridentはPVCを管理し、バックエンドでFlexVol volumeとLUNの名前を変更します。フラグを渡して管理対象外のボリュームをインポートできます`--no-manage`。を使用する場合`--no-manage`、Tridentはオブジェクトのライフサイクル中、PVCまたはPVに対して追加の操作を実行しません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などの他の処理も無視されます。



このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

- PVCとPVにアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、およびPVCとPVが管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

## ボリュームをインポートします

を使用してボリュームをインポートできます`tridentctl import`。

### 手順

- PVCの作成に使用するPersistent Volume Claim (PVC; 永続的ボリューム要求) ファイル(など)を作成します`pvc.yaml`。PVCファイルには、`namespace`、`accessModes`、および`storageClassName`が含まれている必要があります`name`。必要に応じて、PVC定義でを指定できます`unixPermissions`。

最小仕様の例を次に示します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class
```



PV名やボリュームサイズなどの追加のパラメータは指定しないでください。これにより原因、インポートコマンドが失敗する可能性があります。

2. コマンドを使用して `tridentctl import`、ボリュームを含むTridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Element Volume、Cloud Volumes Serviceパスなど）を指定します。`-f`PVCファイルへのパスを指定するには、引数が必要です。

```
tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
```

## 例

サポートされているドライバについて、次のボリュームインポートの例を確認してください。

### ONTAP NASおよびONTAP NAS FlexGroup

Tridentは、ドライバと`ontap-nas-flexgroup`ドライバを使用したボリュームインポートをサポートしています`ontap-nas`ます

- 
- `ontap-nas-economy`ドライバはqtreeをインポートおよび管理できません。
  - `ontap-nas`ドライバと`ontap-nas-flexgroup`ドライバでは、ボリューム名の重複は許可されません。

ドライバを使用して作成される各ボリューム`ontap-nas`は、ONTAPクラスタ上のFlexVol volumeになります。ドライバを使用したFlexVolボリュームのインポート`ontap-nas`も同様に機能します。ONTAPクラスタにすでに存在するFlexVolボリュームは、PVCとしてインポートできます`ontap-nas`。同様に、FlexGroupボリュームはPVCとしてインポートできます`ontap-nas-flexgroup`。

### ONTAP NASの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

## 管理対象ボリューム

次の例は、という名前のバックエンドにある`ontap\_nas`という名前のボリュームをインポートし`managed\_volume`ます。

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+-----+-----+-----+
+-----+-----+-----+
|           NAME           |  SIZE   | STORAGE CLASS |
PROTOCOL |           BACKEND UUID           | STATE  | MANAGED |
+-----+-----+-----+
+-----+-----+-----+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard   |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online  | true      |
+-----+-----+-----+
+-----+-----+-----+
```

## 管理対象外のボリューム

引数を使用した場合`--no-manage`、Tridentはボリュームの名前を変更しません。

次に、バックエンドで`ontap\_nas`をインポートする例を示し`unmanaged\_volume`ます。

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+-----+-----+-----+
+-----+-----+-----+
|           NAME           |  SIZE   | STORAGE CLASS |
PROTOCOL |           BACKEND UUID           | STATE  | MANAGED |
+-----+-----+-----+
+-----+-----+-----+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard   |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online  | false     |
+-----+-----+-----+
+-----+-----+-----+
```

## ONTAP SAN

Tridentは、ドライバと`ontap-san-economy`ドライバを使用したボリュームインポートをサポートしている`ontap-san`ます

Tridentでは、単一のLUNを含むONTAP SAN FlexVolボリュームをインポートできます。これは、ドライバと一致している`ontap-san`ます。ドライバは、PVCごとにFlexVol volumeを作成し、FlexVol volume内にLUNを作成します。TridentはFlexVol volumeをインポートし、PVC定義に関連付けます。

## ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

### 管理対象ボリューム

管理対象ボリュームの場合、TridentはFlexVol volumeの名前を形式に、FlexVol volume内のLUNの名前をに`lun0`変更`pvc-<uuid>`します。

次に、バックエンドにあるFlexVol volume `ontap\_san\_default`をインポートする例を示し`ontap-san-managed`ます。

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-basic-import.yaml -n trident -d
```

PROTOCOL	NAME	BACKEND	UUID	SIZE	STORAGE CLASS	STATE	MANAGED
block	pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a	cd394786-ddd5-4470-adc3-10c5ce4ca757	20 MiB	basic	online	true	

### 管理対象外のボリューム

次に、バックエンドで`ontap\_san`をインポートする例を示し`unmanaged\_example\_volume`ます。

PROTOCOL	NAME	BACKEND	UUID	SIZE	STORAGE CLASS	STATE	MANAGED
block	pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228	e3275890-7d80-4af6-90cc-c7a0759f555a	1.0 GiB	san-blog	online	false	

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、というエラーが表示されます。`LUN already mapped to initiator(s) in this group`ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

Vserver	Igroup	Protocol	OS	Type	Initiators
svm0	k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3	iscsi	linux	iqn.1994-05.com.redhat:4c2e1cf35e0	
svm0	unmanaged-example-igroup	mixed	linux	iqn.1994-05.com.redhat:4c2e1cf35e0	

## 要素

Tridentは、NetApp Elementソフトウェアとドライバを使用したNetApp HCIボリュームインポートをサポートしています solidfire-san。



Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合、Tridentはエラーを返します。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

## 要素の例

次の例は、バックエンドにボリュームを `element\_default` インポートし `element-managed` ます。

```
tridentctl import volume element_default element-managed -f pvc-basic-import.yaml -n trident -d

+-----+-----+-----+
+-----+-----+-----+-----+
|           NAME           |  SIZE  | STORAGE CLASS | 
PROTOCOL |           BACKEND UUID           | STATE | MANAGED | 
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element | 
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    | 
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

## Google Cloud Platform

Tridentはドライバを使用したボリュームインポートをサポートしてい `gcp-cvs` ます。



NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスの後に続く部分です :/。たとえば、エクスポートパスがの場合、`10.0.0.1:/adroit-jolly-swift` ボリュームパスはになり `adroit-jolly-swift` ます。

## Google Cloud Platformの例

次の例は、ボリュームパスがの `adroit-jolly-swift` バックエンドにボリュームを `gpcvs\_YEppr` インポートし `gcp-cvs` ます。

```
tridentctl import volume gpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-file> -n trident

+-----+-----+-----+
+-----+-----+-----+
|           NAME           |  SIZE  | STORAGE CLASS | 
PROTOCOL |           BACKEND UUID           | STATE | MANAGED | 
+-----+-----+-----+
+-----+-----+-----+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true      |
+-----+-----+-----+
+-----+-----+-----+
```

## Azure NetApp Files

Tridentはドライバを使用したボリュームインポートをサポートしてい `azure-netapp-files` ます。



Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスの後に続く部分です :/。たとえば、マウントパスがの場合、`10.0.0.2:/importvol1` ボリュームパスはになり `importvol1` ます。

## Azure NetApp Filesの例

次の例は、ボリュームパスを持つ `importvol1` バックエンドのボリューム `azurenappfiles\_40517` をインポートし `azure-netapp-files` ます。

```
tridentctl import volume azurenappfiles_40517 importvol1 -f <path-to-pvc-file> -n trident

+-----+-----+-----+
+-----+-----+-----+
|           NAME           |  SIZE  | STORAGE CLASS | 
PROTOCOL |           BACKEND UUID           | STATE | MANAGED | 
+-----+-----+-----+
+-----+-----+-----+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage | file
| 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true      |
+-----+-----+-----+
+-----+-----+-----+
```

## Google Cloud NetAppボリューム

Tridentはドライバを使用したボリュームインポートをサポートしてい`google-cloud-netapp-volumes`ます。

### Google Cloud NetApp Volumeの例

次の例は、ボリュームと一緒に`testvoleasiaeast1`バックエンドにボリュームを`backend-tbc-gcnv1`インポートし`google-cloud-netapp-volumes`ます。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-to-pvc> -n trident

+-----+-----+
+-----+-----+
+-----+-----+
|           NAME           |  SIZE   | STORAGE CLASS
| PROTOCOL |           BACKEND UUID           | STATE  | MANAGED |
+-----+-----+
+-----+-----+
+-----+-----+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 | 10 GiB | gcnv-nfs-sc-
identity | file    | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+-----+-----+
+-----+-----+
+-----+-----+
```

次の例は、同じリージョンに2つのボリュームがある場合にボリュームをインポートし`google-cloud-netapp-volumes`ます。

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+-----+-----+
+-----+-----+
+-----+-----+
|           NAME           |  SIZE   | STORAGE CLASS
| PROTOCOL |           BACKEND UUID           | STATE  | MANAGED |
+-----+-----+
+-----+-----+
+-----+-----+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 | 10 GiB | gcnv-nfs-sc-
identity | file      | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+-----+-----+
+-----+-----+
+-----+-----+
```

## ボリュームの名前とラベルをカスタマイズする

Tridentでは、作成したボリュームにわかりやすい名前とラベルを割り当てることができます。これにより、ボリュームを特定し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできます。また、バックエンドレベルでテンプレートを定義してカスタムボリューム名とカスタムラベルを作成することもできます。作成、インポート、またはクローンを作成するボリュームは、テンプレートに準拠します。

### 開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

1. ボリュームの作成、インポート、クローニングの各処理。
2. ontap-nas-economy ドライバの場合、qtreeボリュームの名前だけがテンプレート名に準拠します。
3. ontap-san-economy ドライバの場合、名前テンプレートに準拠するのはLUN名のみです。

### 制限事項

1. カスタマイズ可能なボリューム名は、ONTAPオンプレミスドライバとのみ互換性があります。
2. カスタマイズ可能なボリューム名は、既存のボリュームには適用されません。

### カスタマイズ可能なボリューム名の主な動作

1. 名前テンプレートの無効な構文が原因でエラーが発生した場合、バックエンドの作成は失敗します。ただし、テンプレートアプリケーションが失敗した場合は、既存の命名規則に従ってボリュームに名前が付け

られます。

2. バックエンド構成の名前テンプレートを使用してボリュームの名前が指定されている場合、ストレージプレフィックスは適用されません。任意のプレフィックス値をテンプレートに直接追加できます。

## 名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベルで定義できます。

### ルートレベルの例

```
{  
  "version": 1,  
  "storageDriverName": "ontap-nas",  
  "backendName": "ontap-nfs-backend",  
  "managementLIF": "<ip address>",  
  "svm": "svm0",  
  "username": "<admin>",  
  "password": "<password>",  
  "defaults": {  
    "nameTemplate":  
      "{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.RequestName}}"  
  },  
  "labels": {  
    "cluster": "ClusterA",  
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"  
  }  
}
```

## プールレベルの例

```
{  
  "version": 1,  
  "storageDriverName": "ontap-nas",  
  "backendName": "ontap-nfs-backend",  
  "managementLIF": "<ip address>",  
  "svm": "svm0",  
  "username": "<admin>",  
  "password": "<password>",  
  "useREST": true,  
  "storage": [  
    {  
      "labels": {  
        "labelname": "label1",  
        "name": "{{ .volume.Name }}"  
      },  
      "defaults": {  
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster }}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"  
      }  
    },  
    {  
      "labels": {  
        "cluster": "label2",  
        "name": "{{ .volume.Name }}"  
      },  
      "defaults": {  
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster }}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"  
      }  
    }  
  ]  
}
```

## 名前テンプレートの例

\*例1\* :

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{ .config.BackendName }}"
```

\*例2\* :

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{ slice .volume.RequestName 1 5 }}""
```

## 考慮すべきポイント

1. ボリュームインポートの場合、既存のボリュームに特定の形式のラベルがある場合にのみラベルが更新されます。例： {"provisioning": {"Cluster": "ClusterA", "PVC": "pvcname"} }。
2. 管理対象ボリュームのインポートの場合、ボリューム名はバックエンド定義のルートレベルで定義された名前テンプレートの後に続きます。
3. Tridentでは、storageプレフィックスを指定したスライス演算子の使用はサポートされていません。
4. テンプレートによってボリューム名が一意にならない場合、Tridentではいくつかのランダムな文字が追加されて一意のボリューム名が作成されます。
5. NASエコノミーボリュームのカスタム名の長さが64文字を超える場合、Tridentは既存の命名規則に従ってボリュームに名前を付けます。他のすべてのONTAPドライバでは、ボリューム名が名前の上限を超えると、ボリュームの作成プロセスが失敗します。

## ネームスペース間でNFSボリュームを共有します

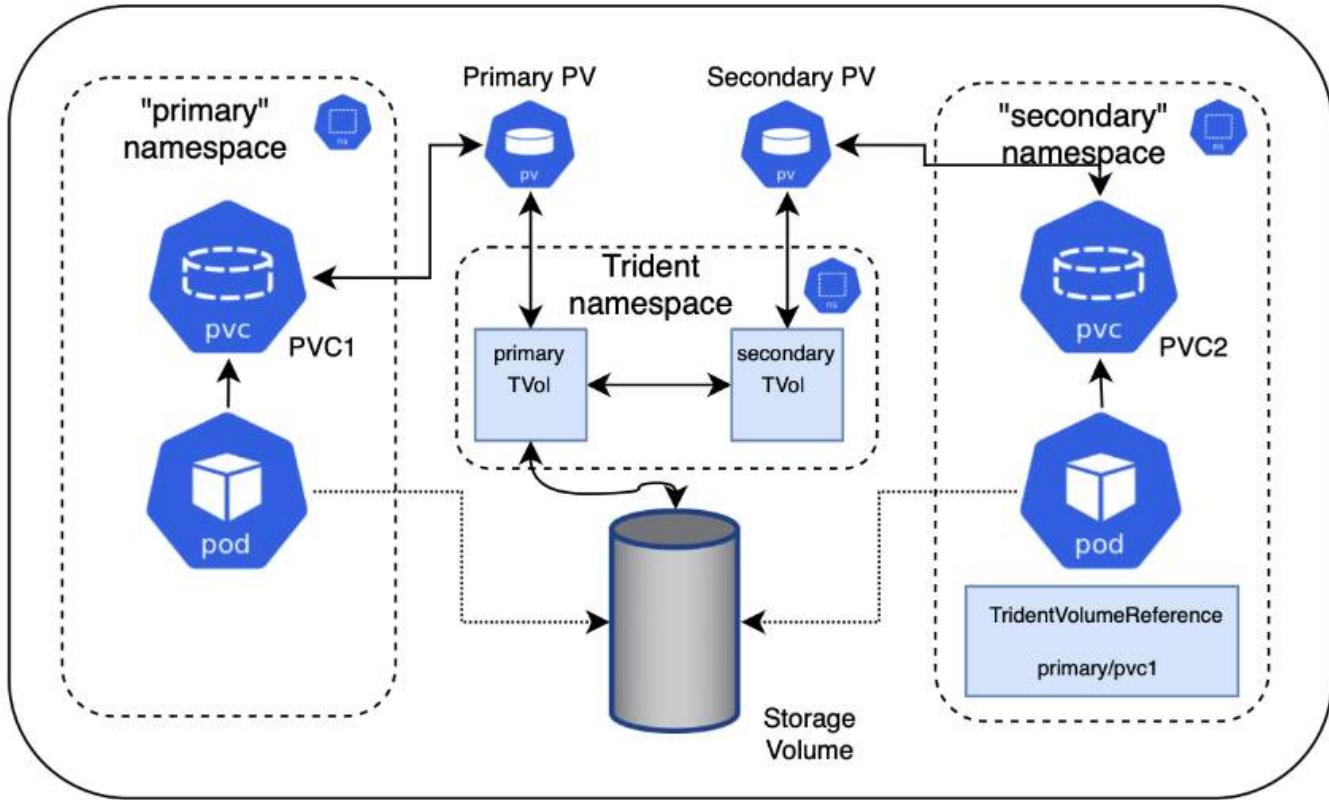
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

### 特徴

TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany (RWX) NFSボリュームを安全に共有できます。このKubernetesネイティブ解決策には、次のようなメリットがあります。

- ・セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- ・すべてのTrident NFSボリュームドライバで動作
- ・tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。



## クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1

ボリュームを共有するように送信元PVCを設定する

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2

宛先名前空間にCRを作成する権限を付与する

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3

デスティネーションネームスペースにTridentVolumeReferenceを作成

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4

宛先ネームスペースに下位PVCを作成します。

宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

## 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を設定します

セキュリティを確保するために、ネームスペース間共有では、ソースネームスペースの所有者、クラスタ管理者、および宛先ネームスペースの所有者によるコラボレーションとアクションが必要です。ユーザロールは各手順で指定します。

### 手順

1. ソースネームスペースの所有者: pvc(pvc1を作成します) (`namespace2。注釈を使用して、デスティネーションネームスペースとの共有権限を付与します。 shareToNamespace

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Tridentは、PVとそのバックエンドNFSストレージボリュームを作成します。

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。たとえば、`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`です。
- を使用して、すべてのネームスペースと共有できます \*。例えば、`trident.netapp.io/shareToNamespace: \*`
- PVCはいつでも更新してアノテーションを含めることができます`shareToNamespace`。



2. \*クラスタ管理者: \*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. \*デスティネーションネームスペースの所有者: \*ソースネームスペースを参照するTridentVolumeReference CRをデスティネーションネームスペースに作成します pvc1。

```

apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1

```

4. 宛先ネームスペース所有者：(pvc2`宛先ネームスペースにPVCを作成(`namespace2)。注釈を使用して送信元PVCを指定します。shareFromPVC

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi

```



宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

## 結果

TridentはデスティネーションPVCのアノテーションを読み取り shareFromPVC、ソースPVストレージリソースを共有する独自のストレージリソースのない下位ボリュームとしてデスティネーションPVを作成します。宛先PVCとPVは、通常どおりバインドされているように見えます。

## 共有ボリュームを削除

複数のネームスペースで共有されているボリュームは削除できます。Tridentは、ソースネームスペース上のボリュームへのアクセスを削除し、そのボリュームを共有する他のネームスペースへのアクセスを維持します。このボリュームを参照しているネームスペースをすべて削除すると、Tridentによってボリュームが削除されます。

## 下位ボリュームのクエリに使用 tridentctl get

ユーティリティを使用する[tridentctl`と、コマンドを実行して従属ボリュームを取得できます`get。詳

細については、リンク`../Trident-reference/tridentctl.html`コマンドとオプション]を参照して[`tridentctl`ください。

```
Usage:  
tridentctl get [option]
```

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`：クエリをボリュームの下位に限定します。

## 制限事項

- Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれないようにすることはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- または `shareFromNamespace`、注釈を削除したり、CRを削除したりして `TridentVolumeReference`、て、送信元PVCへのアクセスを取り消すことはできません。`shareToNamespace`。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

## 詳細情報

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします"ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

## ネームスペース全体でボリュームをクローニング

Tridentを使用すると、同じKubernetesクラスタ内の別のネームスペースから既存のボリュームまたはボリュームSnapshotを使用して新しいボリュームを作成できます。

### 前提条件

ボリュームをクローニングする前に、ソースとデスティネーションのバックエンドのタイプとストレージクラスが同じであることを確認してください。

### クイックスタート

ボリュームクローニングはわずか数ステップでセットアップできます。

1

ボリュームのクローンを作成するためのソースPVCの設定

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2

宛先名前空間にCRを作成する権限を付与する

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3

デスティネーションネームスペースにTridentVolumeReferenceを作成

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4

デスティネーションネームスペースにクローンPVCを作成します。

宛先ネームスペースの所有者は、PVCを作成して、送信元ネームスペースからPVCを複製します。

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を設定します

セキュリティを確保するために、ネームスペース間でボリュームをクローニングするには、ソースネームスペースの所有者、クラスタ管理者、およびデスティネーションネームスペースの所有者が協力して対処する必要があります。ユーザロールは各手順で指定します。

手順

1. ソースネームスペース所有者：(pvc1`、ソースネームスペースにPVCを作成(`namespace1)。注釈(`namespace2`を使用して、デスティネーションネームスペースと共有する権限を付与します。  
`cloneToNamespace

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Tridentは、PVとそのバックエンドストレージボリュームを作成します。



- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。たとえば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`です。
- を使用して、すべてのネームスペースと共有できます \*。例えば、`trident.netapp.io/cloneToNamespace: \*`
- PVCはいつでも更新してアノテーションを含めることができます`cloneToNamespace`。

2. \*クラスタ管理者：\*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRをデスティネーションネームスペースに作成する権限を付与し(`namespace2`ます)。
3. \*デスティネーションネームスペースの所有者：\*ソースネームスペースを参照するTridentVolumeReference CRをデスティネーションネームスペースに作成します pvc1。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者：(pvc2`宛先ネームスペースに`cloneFromNamespace`PVCを作成(`namespace2)。または`cloneFromSnapshot`アノテーションを使用して、送信元PVCを指定します`cloneFromPVC`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

## 制限事項

- ONTAP NASエコノミードライバを使用してプロビジョニングされたPVCでは、読み取り専用クローンはサポートされません。

## SnapMirrorによるボリュームのレプリケート

Tridentでは、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、ピア関係にあるクラスタのソースボリュームとデスティネーションボリュームの間のミラー関係をサポートしています。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

- \* Trident \* : Tridentバージョン22.10以降が、バックエンドとしてONTAPを利用するソースとデスティネーションの両方のKubernetesクラスタに存在している必要があります。
- 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。

- \* TridentとSVM \* : ピア関係にあるリモートSVMをデスティネーションクラスタのTridentで使用できる必要があります。

### 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
```

Trident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）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"
```

Tridentは、設定した関係ポリシー名（ONTAPの場合はデフォルト）を使用してSnapMirror関係を作成して初期化します。

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
```

TridentはTridentMirrorRelationship CRDをチェックし、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、Tridentは新しいFlexVol volumeを、MirrorRelationshipで定義されているリモートSVMとピア関係にあるSVMに配置します。

## 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。宛先TMRには、目的の状態をTridentに通知する状態があります。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

- 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。
- 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
```

- 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。
- 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

Tridentでは、プライマリボリュームとセカンダリボリュームで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ックスol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

Tridentは、レプリケートされたPVCがないかどうかを確認し、レプリケーション関係を解放してからボリュームの削除を試行します。

### TMRの削除

ミラー関係の片側のTMRを削除すると、Trident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されたTMRがすでに\_promoted\_stateにある場合、既存のミラー関係は存在せず、TMRは削除され、TridentはローカルPVCを\_ReadWrite\_にプロモートします。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、TridentがONTAPクラスタに直接接続されていなくても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。

## CSI トポロジを使用します

Tridentでは、を使用して、Kubernetesクラスタ内のノードを選択的に作成して接続できます "CSI トポロジ機能"。

### 概要

CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、Kubernetes管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリュームのプロビジョニングを容易にするために、TridentではCSIトポロジを使用しています。



CSI トポロジ機能の詳細については、こちらを参照して "[ここをクリック](#)" ください。

Kubernetes には、2 つの固有のボリュームバインドモードがあります。

- `VolumeBindingMode` をに設定する `Immediate` と、Tridentはトポロジを認識せずにボリュームを作成します。ボリュームバインディングと動的プロビジョニングは、PVC が作成されるときに処理されます。これはデフォルト `VolumeBindingMode` であり、トポロジの制約を適用しないクラスタに適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。

- `VolumeBindingMode`を`WaitForFirstConsumer`に設定する`WaitForFirstConsumer`と、PVCの永続ボリュームの作成とバインディングは、PVCを使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。



`WaitForFirstConsumer` バインディングモードではトポロジラベルは必要ありません。これは CSI トポロジ機能とは無関係に使用できます。

## 必要なもの

CSI トポロジを使用するには、次のものが必要です。

- を実行するKubernetesクラスタ"サポートされるKubernetesバージョン"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedead99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedead99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードには、トポロジ対応と`topology.kubernetes.io/zone`を示すラベルを付ける必要があります(`topology.kubernetes.io/region`も)。これらのラベル\*は、Tridentをトポロジ対応にするためにTridentをインストールする前に、クラスタ内のノード\*に設定しておく必要があります。

```
kubectl get nodes -o=jsonpath='{range .items[*]}{{.metadata.name},\n{.metadata.labels}}{"\n"}{end}' | grep --color "topology.kubernetes.io"\n[node1,\n{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube\nrnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/\nos":"linux","node-\nrole.kubernetes.io/master":"","topology.kubernetes.io/region":"us-\nwest1","topology.kubernetes.io/zone":"us-west1-a"}]\n[node2,\n{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/\nos":"linux","node-\nrole.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-\nwest1","topology.kubernetes.io/zone":"us-west1-b"}]\n[node3,\n{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/\nos":"linux","node-\nrole.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-\nwest1","topology.kubernetes.io/zone":"us-west1-c"}]
```

## 手順1：トポロジ対応バックエンドを作成する

Tridentストレージバックエンドは、アベイラビリティゾーンに基づいて選択的にボリュームをプロビジョニングするように設計できます。各バックエンドは、サポートされているゾーンとリージョンのリストを表すオプションのブロックを運ぶことができます `supportedTopologies`。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン/ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

## YAML

```
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

## JSON

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```



`supportedTopologies`は、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、StorageClass で指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むストレージクラスの場合、Tridentはバックエンドにボリュームを作成します。

ストレージプールごとにも定義できます `supportedTopologies`。次の例を参照してください。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

この例では `region`、ラベルと `zone`、ラベルはストレージプールの場所を表しています。  
`topology.kubernetes.io/region` `topology.kubernetes.io/zone` ストレージプールの消費元を指定します。

## 手順 2：トポロジを認識するストレージクラスを定義する

クラスタ内のノードに提供されるトポロジラベルに基づいて、トポロジ情報を含めるように `StorageClasses` を定義できます。これにより、作成された PVC 要求の候補となるストレージプール、および Trident によってプロビジョニングされたボリュームを使用できるノードのサブセットが決まります。

次の例を参照してください。

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata: null
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions: null
  - key: topology.kubernetes.io/zone
    values:
      - us-east1-a
      - us-east1-b
  - key: topology.kubernetes.io/region
    values:
      - us-east1
parameters:
  fsType: ext4

```

前述のStorageClass定義では、volumeBindingMode`がに設定されて`WaitForFirstConsumer`います。このStorageClassで要求されたPVCは、ポッドで参照されるまで処理されません。およびに、`allowedTopologies`を使用するゾーンとリージョンを示します。StorageClassは`netapp-san-us-east1`、上記で定義したバックエンドにPVCを作成し`san-backend-us-east1`ます。

### ステップ3：PVCを作成して使用する

StorageClassを作成してバックエンドにマッピングすると、PVCを作成できるようになりました。

次の例を参照して`spec`ください。

```

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1

```

このマニフェストを使用してPVCを作成すると、次のような結果になります。

```

kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS      VOLUME      CAPACITY      ACCESS MODES      STORAGECLASS
AGE
pvc-san   Pending      2s          netapp-san-us-east1
2s

kubectl describe pvc
Name:            pvc-san
Namespace:       default
StorageClass:    netapp-san-us-east1
Status:          Pending
Volume:
Labels:          <none>
Annotations:     <none>
Finalizers:      [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:      Filesystem
Mounted By:      <none>
Events:
  Type      Reason      Age      From      Message
  ----      ----      ----      ----      -----
  Normal    WaitForFirstConsumer  6s      persistentvolume-controller  waiting
for first consumer to be created before binding

```

Trident でボリュームを作成して PVC にバインドするには、ポッド内の PVC を使用します。次の例を参照してください。

```

apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false

```

このpodSpecは、リージョンに存在するノードでポッドをスケジュールし、ゾーンまたは`us-east1-b`ゾーンに存在する任意のノードから選択する`us-east1-a`のようにKubernetesに指示し`us-east1`ます。

次の出力を参照してください。

```

kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS    VOLUME
ACCESS MODES   STORAGECLASS   AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO           netapp-san-us-east1   48s   Filesystem

```

## バックエンドを更新して含める `supportedTopologies`

既存のバックエンドを更新して、使用の `tridentctl backend update` リストを含めることができます ``supportedTopologies``。これは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

## 詳細情報

- "コンテナのリソースを管理"
- "ノードセレクタ"
- "アフィニティと非アフィニティ"
- "塗料および耐性"

## スナップショットを操作します

永続ボリューム (PV) のKubernetesボリュームSnapshotを使用すると、ボリュームのポイントインタイムコピーを作成できます。Tridentを使用して作成したボリュームのSnapshotの作成、Tridentの外部で作成したSnapshotのインポート、既存のSnapshotからの新しいボリュームの作成、Snapshotからのボリュームデータのリカバリを実行できます。

## 概要

ボリュームスナップショットは以下でサポートされています `ontap-nas`、`ontap-nas-flexgroup`、`ontap-san`、`ontap-san-economy`、`solidfire-san`、`gcp-cvs`、`azure-netapp-files`、そして ``google-cloud-netapp-volumes`` ドライバー。

### 開始する前に

スナップショットを操作するには、外部スナップショットコントローラとカスタムリソース定義 (CRD) が必要です。Kubernetesオーケストレーションツール (例: Kubeadm、GKE、OpenShift) の役割を担っています。

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、を参照してください [ボリュームSnapshotコントローラの導入](#)。



GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEでは、内蔵の非表示のスナップショットコントローラを使用します。

## ボリューム Snapshot を作成します

手順

1. を作成し `VolumeSnapshotClass` ます。詳細については、を参照してください"ボリュームSnapshotクラス"。

- は `driver` Trident CSI ドライバを示しています。
- `deletionPolicy` には、または `Retain` を指定できます `Delete` に設定する `Retain` と、オブジェクトが削除されても、ストレージクラスタの基盤となる物理Snapshotが保持され `VolumeSnapshot` ます。

例

```
cat snap-sc.yaml
```

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

2. 既存のPVCのスナップショットを作成します。

例

- 次に、既存のPVCのスナップショットを作成する例を示します。

```
cat snap.yaml
```

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

- この例では、というPVCのボリュームSnapshotオブジェクトを作成し pvc1、Snapshotの名前をに設定して `pvc1-snap` います。VolumeSnapshotはPVCに似ており、実際のSnapshotを表すオブジェクト

に関連付けられて `VolumeSnapshotContent` います。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME          AGE
pvc1-snap    50s
```

- VolumeSnapshotのオブジェクト `pvc1-snap` を説明することで特定できます  
`VolumeSnapshotContent` は Snapshot Content Name、このSnapshotを提供するVolumeSnapshotContentオブジェクトを識別します。パラメータは、`Ready To Use` スナップショットを使用して新しいPVCを作成できることを示します。

```
kubectl describe volumesnapshots pvc1-snap
Name:           pvc1-snap
Namespace:      default
...
Spec:
  Snapshot Class Name:  pvc1-snap
  Snapshot Content Name: snapcontent-e8d8a0ca-9826-11e9-9807-
  525400f3f660
  Source:
    API Group:
    Kind:      PersistentVolumeClaim
    Name:      pvc1
  Status:
    Creation Time:  2019-06-26T15:27:29Z
    Ready To Use:  true
    Restore Size:  3Gi
  ...
...
```

## ボリュームSnapshotからPVCを作成

を使用して、という名前のVolumeSnapshotをデータのソースとして使用してPVCを作成 `<pvc-name>` できます `dataSource`。作成されたPVCは、ポッドに接続して、他のPVCと同様に使用できます。



PVCはソースボリュームと同じバックエンドに作成されます。を参照してください ["KB : Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"。](#)

次に、をデータソースとして使用してPVCを作成する例を示し `pvc1-snap` ます。

```
cat pvc-from-snap.yaml
```

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvcl-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io

```

## ボリュームSnapshotのインポート

Tridentでは、クラスタ管理者がを"Kubernetesの事前プロビジョニングされたSnapshotプロセス"使用して、オブジェクトを作成したり、Tridentの外部で作成されたSnapshotをインポートしたりできます VolumeSnapshotContent。

開始する前に

TridentでSnapshotの親ボリュームが作成またはインポートされている必要があります。

手順

1. \*クラスタ管理者 : \*バックエンドSnapshotを参照するオブジェクトを作成します VolumeSnapshotContent。これにより、TridentでSnapshotワークフローが開始されます。
  - にバックエンドスナップショットの名前を `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">` 指定します `annotations`。
  - で指定します `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` `snapshotHandle`。この情報は、呼び出しで外部スナップショットによってTridentに提供される唯一の情報です `ListSnapshots`。



CRの名前の制約により、は`<volumeSnapshotContentName>`バックエンドスナップショット名と常に一致しません。

例

次の例では、バックエンドスナップショットを参照するオブジェクトを `snap-01` 作成し `VolumeSnapshotContent` ます。

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01" # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default

```

2. \*クラスタ管理者：\*オブジェクトを参照するCR `VolumeSnapshotContent`を作成します`VolumeSnapshot`。これにより、指定された名前空間でを使用するためのアクセスが要求され`VolumeSnapshot`ます。

例

次の例では、という名前 `import-snap-content` を参照する `VolumeSnapshotContent`、という名前のCRを `import-snap` 作成します `VolumeSnapshot`。

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
  provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content

```

3. \*内部処理（アクション不要）：\*外部スナップショットは、新しく作成されたを認識して`VolumeSnapshotContent`呼び出しを実行します `ListSnapshots`。Tridentによってが作成され`TridentSnapshot`ます。
- 外部スナップショットは、をに `readyToUse` 設定し、`VolumeSnapshot`をに `true` 設定し`VolumeSnapshotContent`ます。
  - Tridentが戻ります `readyToUse=true`。
4. \*任意のユーザー：\*を作成し `PersistentVolumeClaim` て、新しいを参照します`VolumeSnapshot`。`spec.dataSource`（または `spec.dataSourceRef`）の名前は名前です

VolumeSnapshot。

## 例

次に、という名前の `import-snap` を参照するPVCを作成する例を示し `VolumeSnapshot` ます。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
  apiGroup: snapshot.storage.k8s.io
```

## Snapshotを使用したボリュームデータのリカバリ

デフォルトでは、ドライバと `ontap-nas-economy` ドライバを使用してプロビジョニングされたボリュームの互換性を最大限に高めるため、snapshotディレクトリは非表示になってい `ontap-nas` ます。ディレクトリがスナップショットからデータを直接リカバリできるようにし `snapshot` ます。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::>*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```



Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

## Snapshotからのインプレースボリュームのリストア

Tridentでは、(TASR) CRを使用してSnapshotからボリュームをインプレースで迅速にリストアできます TridentActionSnapshotRestore。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。

Tridentは、`ontap-san-economy` `ontap-nas`、`ontap-nas-flexgroup` `azure-netapp-files`、`gcp-cvs` のSnapshotリストアをサポートしています。`ontap-san`、`google-cloud-netapp-volumes`、および `solidfire-san` ドライバ。

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。この例では、PVCおよびボリュームスナップショット用のCRを作成し `pvc1`、`pvc1-snapshot` ます。



TASR CRは、PVCおよびVSが存在する名前空間に存在する必要があります。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  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 `pvc1` ます。

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

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

結果

Tridentはスナップショットからデータをリストアします。Snapshotリストアのステータスを確認できます。

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    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Tridentで処理が自動的に再試行されることはありません。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。

## Snapshotが関連付けられているPVを削除する

Snapshotが関連付けられている永続ボリュームを削除すると、対応するTridentボリュームが「削除中」に更新されます。ボリュームSnapshotを削除してTridentボリュームを削除します。

## ボリュームSnapshotコントローラの導入

KubernetesディストリビューションにスナップショットコントローラとCRDが含まれていない場合は、次のように導入できます。

### 手順

1. ボリュームのSnapshot作成

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

## 2. スナップショットコントローラを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```



必要に応じて、名前空間を開い `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` で更新し `namespace` ます。

## 関連リンク

- ・"ボリューム Snapshot"
- ・"ボリュームSnapshotクラス"

## 著作権に関する情報

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を隨時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5225.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用権を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用権については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

## 商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。