



ボリュームのプロビジョニングと管理

Trident

NetApp
October 29, 2026

目次

ボリュームのプロビジョニングと管理	1
Tridentボリュームをプロビジョニングする	1
概要	1
PVC を作成する	1
ボリュームの移動	4
要件	4
仕組み	4
ボリュームの移動	4
Tridentボリュームの拡張	7
iSCSI ボリュームを拡張する	7
FCボリュームを拡張する	11
NFSボリュームを拡張する	14
RWX NVMe サブシステムの制限について	17
64ノード制限について理解する	17
NVMeサブシステムモデルを理解する	18
エラーの兆候を特定する	18
サブシステム制限エラーを解決する	18
スーパーサブシステムモデルを適用するために Trident をアップグレードする	18
コントローラーのスケラビリティ	19
主要な概念と定義	19
デフォルトの動作（シリアルモード）	19
コントローラーのスケラビリティサポート	19
enableConcurrency の仕組み	20
有効にする前に	20
コントローラーのスケラビリティを有効にする	21
並行動作	23
注意点と制限事項	23
推奨	24
自動ボリューム拡張	24
要件	24
制限事項	25
ボリュームの自動拡張ポリシーによるプロビジョニング	25
自動成長ポリシーを作成する	25
自動成長ポリシーを作成する	26
ポリシーの状態	27
ポリシーをStorageClassに関連付ける	27
ポリシーの優先順位	28
設定例	29
自動成長ポリシーの管理	31

自動拡張ポリシーを表示	31
自動成長ポリシーを更新する	32
自動成長ポリシーを削除する	32
自動拡張ポリシーの使用状況を監視する	33
サポートされるプロトコル	34
サポートされているバックエンド	34
既知の制限事項	34
よくある質問	34
Tridentボリュームのインポート	40
概要と考慮事項	40
ボリュームをインポートする	41
例	43
ONTAP SAN-economyの例	49
Tridentのボリューム名とラベルのカスタマイズ	52
開始する前に	52
制限事項	52
カスタマイズ可能なボリューム名の主な動作	52
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	52
名前 template の例	54
考慮すべき点	54
名前空間をまたいでTrident NFSボリュームを共有する	54
機能	55
クイック スタート	55
ソースネームスペースと宛先ネームスペースを設定する	56
共有ボリュームを削除する	57
`tridentctl get`を使用して従属ボリュームを照会する	58
制限事項	58
詳細情報	58
名前空間をまたいだTridentボリュームのクローン作成	58
前提条件	58
クイック スタート	59
ソースネームスペースと宛先ネームスペースを設定する	59
制限事項	61
SnapMirrorを使用したボリュームのレプリケート	61
レプリケーションの前提条件	61
ミラー PVC を作成する	62
ボリュームレプリケーションの状態	65
計画外のフェイルオーバー中にセカンダリ PVC を昇格する	65
計画されたフェイルオーバー中にセカンダリ PVC を昇格する	65
フェイルオーバー後にミラー関係を復元する	66
追加操作	66

ONTAPがオンラインのときにミラー関係を更新する	67
ONTAPがオフラインのときにミラー関係を更新する	67
CSI トポロジを使用する	68
概要	68
ステップ1：トポロジー対応のバックエンドを作成する	69
ステップ2：トポロジーを認識するStorageClassesを定義する	71
ステップ3：PVCを作成して使用する	72
バックエンドを更新して`supportedTopologies`を含める	74
詳細情報の参照	74
Tridentスナップショットの操作	74
概要	75
ボリューム Snapshot を作成する	75
ボリュームSnapshotからPVCを作成する	77
ボリューム Snapshot をインポートする	77
スナップショットを使用してボリュームデータを回復する	79
Snapshotからのインプレースボリュームリストア	80
関連するスナップショットを含む PV を削除する	81
ボリュームSnapshotコントローラを導入する	82
関連リンク	82
Tridentボリュームグループのスナップショットの操作	83
ボリュームグループのSnapshotを作成する	83
グループSnapshotを使用してボリュームデータをリカバリする	85
Snapshotからのインプレースボリュームリストア	85
関連するグループスナップショットを含む PV を削除する	85
ボリュームSnapshotコントローラを導入する	85
関連リンク	86

ボリュームのプロビジョニングと管理

Tridentボリュームをプロビジョニングする

設定済みの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

A "[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構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

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 を作成する

手順

1. PVC を作成します。

```
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 `kubectl get pod --watch` で確認できます。

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

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

3. これでPodを削除できます。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
```

サンプルマニフェスト

▼ PersistentVolumeClaim サンプルマニフェスト

これらの例は、基本的な PVC 構成オプションを示しています。

RWO アクセス付き PVC

この例では、RWOアクセスを持つ基本的なPVCが、`basic-csi`という名前のStorageClassに関連付けられています。

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

NVMe/TCP を使用した PVC

この例では、RWOアクセスを持つNVMe/TCPの基本的なPVCが、StorageClassという名前`protection-gold`に関連付けられています。

```
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

これらの例は、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。

基本設定

```
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
```

基本的なNVMe/TCP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
```

"[KubernetesとTridentオブジェクト](#)"を参照して、ストレージクラスが`PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

ボリュームの移動

Trident 26.06以降、ストレージ管理者は、iSCSIボリュームを使用しているワークロードの稼働を維持したまま、同じONTAPクラスター内の別のONTAPノードおよびアグリゲートへ中断なく移動できます。移動は`TridentVolumeMove`カスタムリソースまたは`tridentctl`同等の方法で宣言的にリクエストでき、TridentがONTAPとホストを調整することでI/Oが中断されることはありません。

要件

ボリューム移動は、iSCSIボリューム専用の`ontap-san`ドライバにのみ適用されます。NVMeおよびNASプロトコルはサポートされていません。

ボリュームを移動する前に、以下の要件を満たしていることを確認してください。

- バックエンドは`ontap-san`ドライバーとONTAP REST APIを使用します。
- ONTAPクラスターには少なくとも2つのノードがあります。
- 以下のいずれかの認証情報セットを持っています。
 - 専用カスタム ONTAP ロールを使用する場合は、ONTAP 9.19.1 以降。
 - 以前の ONTAP リリースにおけるクラスター管理者クレデンシャル。

仕組み

Tridentは、同じONTAPクラスター内の2つの場所間でボリューム（LUN）を移動します。データが移動する前に、Tridentは既存のパスに加えてターゲットへの新しいiSCSIパスを追加するため、ホストのマルチパスデバイスがアクセスを失うことはありません。ONTAPが移動を完了した後、Tridentは古いソースパスを削除します。

Tridentは、`TridentVolumeMove`リソースで確認できる順序付けられた一連の状態を通じて移動を進めるため、移動の進捗状況を追跡できます。

ボリュームの移動

ボリュームと場所を特定する

1. 移動するPVの名前を見つめます。PV名は、移動リソースの名前でもあります。
2. ONTAP クラスターで、対象ボリュームの現在の（ソース）ノードとアグリゲートを確認し、移動先（ターゲット）ノードとアグリゲートを選択します。

移動を開始する

どちらの方法でも構いません。どちらも同じ`TridentVolumeMove`リソースを作成します。リソースの名前はPV名です。

tridentctlの使用

```
tridentctl create volume-move \  
  --volume      pvc-8a814d62-bd58-4253-b0d1-82f2885db671 \  
  --source-node ontap-node-01 \  
  --source-pool  aggr1_node01 \  
  --target-node  ontap-node-02 \  
  --target-pool  aggr1_node02 \  
  -n trident
```

`tvm`は`volume-move`のエイリアスです。`--delete-after-success 10m`を追加すると、移動が成功した後にリソースが自動的に削除されます。

TridentVolumeMove リソースの使用

```
apiVersion: trident.netapp.io/v1  
kind: TridentVolumeMove  
metadata:  
  name: pvc-8a814d62-bd58-4253-b0d1-82f2885db671    # the PV to move  
  namespace: trident  
spec:  
  sourceNode: ontap-node-01  
  sourcePool: aggr1_node01  
  targetNode: ontap-node-02  
  targetPool: aggr1_node02  
  # deleteAfterSuccess: 10m    # optional
```

リソースを適用する：

```
kubectl apply -f tridentvolumemove.yaml
```

仕様フィールド

sourceNode / targetNode

ONTAP ノード名 (UUIDではない)。

sourcePool / targetPool

ONTAP アグリゲート名。

deleteAfterSuccess

オプションの期間 (例: 30s または 10m)。`0`に設定するとリソースを即座に削除し、省略するとリソースを保持します。

移動を監視する

進捗状況を追跡するには、`tvm`略称または`tridentvolumemove`を使用します：

```
kubectl get tridentvolumemove -n trident
```

現在の状態とノードごとの進捗状況を確認するには、リソースを記述してください。

```
kubectl describe tridentvolumemove <pv-name> -n trident
```

ボリュームは、`Moving`状態の間に同時接続を持つことができます。これは、ONTAP が大容量ボリュームの移動を完了するまでに時間がかかる場合があるためです。シームレスな移動を確保するために、Trident は Trident コントローラーと Trident ノード間で調整を行います。各状態は、以下の手順を順番に実行し、`Succeeded`または`Failed`で終了します。

状態	何が起こっているか
Pending	Trident は移動要求を受け入れて検証し、ストレージの変更を開始する前に必要な情報を収集します。
ControllerStaging	Trident はターゲットの場所を準備します。ターゲットノードのストレージインターフェイスを LUN にマッピングし、Kubernetes を更新することで、将来のノード再起動時に正しい場所に再接続できるようにします。
NodeStaging	ボリュームがマウントされている各ホストで、Trident はターゲットノードからの新しいストレージパスを既存のパスと並行して追加することで、Pod の I/O が両方のパスセットにわたって継続されるようにします。
Moving	ONTAP はボリュームデータをコピーして、ターゲットのアグリゲートにカットオーバーします。これは最も長いフェーズであり、ONTAP が完全に処理する一方で、Trident は完了するまでポーリングを続けます。この状態では、新規の公開が許可されます。
ControllerUnstaging	データ移行が完了しました。Trident は Kubernetes を更新して、ソースパスを目的の状態から削除し、その後、ソースノードのストレージインターフェイスを LUN からマッピング解除します。
NodeUnstaging	各ホスト上で、Trident は古いソースパスを削除します。ホストにはターゲットパスのみが残され、マルチパスデバイスは中断することなく I/O 処理を継続します。
Succeeded	ボリュームは完全に移動され、ターゲットでオンライン状態になりました。Trident は `TridentVolumeMove` リソースを保持するか、`deleteAfterSuccess` を設定すると自動的に削除します。
Failed	回復不能なエラーにより、移動処理が停止したか、開始できませんでした。Trident はボリュームの状態をリセットして通常の操作を再開できるようにし、エラーメッセージとともにリソースを保持します。メッセージを確認し、根本原因を修正し、リソースを削除してから、もう一度試してください。

`status.attachments` リストは、接続されている各ノードの進捗状況を
`state` フィールドに報告します：

添付ファイルの状態	説明
Pending	ノードは接続済みとして識別されました。作業はまだ開始されていません。
Bridged	既存のパスに加えて、新しいターゲットパスが追加されました。
Migrated	トラフィックは新しいパスを経由して処理されるようになりました。
Cleaned	ノードから古いソースパスが削除されました。
Detached	ノードは移動中に切り離され、Trident がアタッチメントが次のステップに進む準備ができているかどうかを評価する間、それ以上の検討から除外されました。
Failed	このノードでターミナルエラーが発生しました。メッセージをご覧ください。

移動リソースをクリーンアップします

`deleteAfterSuccess` を設定すると、Trident はリソースを自動的に削除します。それ以外の場合は、手動で削除できます。同じボリュームで別の移動をリクエストする前に、そのボリュームに `TridentVolumeMove` リソースが `Succeeded` または `Failed` の状態で存在しないことを確認してください。



Trident は、失敗した移動を自動的に削除しません。Trident は `Failed` リソースを監査レコードとして保持します。根本原因を修正した後、再試行する前にリソースを手動で削除してください。

Trident ボリュームの拡張

Trident は、Kubernetes ユーザーにボリューム作成後にボリュームを拡張する機能を提供します。iSCSI、NFS、SMB、NVMe/TCP、および FC ボリュームを拡張するために必要な構成に関する情報を見つけます。

iSCSI ボリュームを拡張する

CSI プロビジョナーを使用して iSCSI 永続ボリューム (PV) を拡張できます。



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

ステップ1：ボリューム拡張をサポートするように **StorageClass** を設定する

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

```
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、編集して`allowVolumeExpansion`パラメータを含めます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、`spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

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

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

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（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
RW0          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する

サイズを変更するには、PVをポッドに接続します。iSCSI PV のサイズを変更する場合、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のボリュームを拡張し、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。
- アタッチされていないPVのサイズを変更しようとする、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、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。拡張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、`san-pvc`を使用するポッドが作成されます。

```
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、`spec.resources.requests.storage`を2Giに更新します。

```
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
# ...
```

ステップ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、拡張が正しく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
pvc-8a814d62-bd58-4253-b0d1-82f2885db671	2.0 GiB	ontap-san
block	online	true

FCボリュームを拡張する

CSI プロビジョナーを使用して 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、`ontap-san` ドライバでサポートされており、Kubernetes 1.16 以降が必要です。

ステップ1：ボリューム拡張をサポートするように**StorageClass**を設定する

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

```
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、編集して`allowVolumeExpansion`パラメータを含めます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、`spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

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

```

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

```

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（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する

サイズを変更するには、PVをポッドに接続します。FC PVのサイズを変更する場合、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のボリュームを拡張し、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。
- アタッチされていないPVのサイズを変更しようとする、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、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。拡張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、`san-pvc`を使用するポッドが作成されます。

```

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、`spec.resources.requests.storage`を2Giに更新します。

```
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
# ...
```

ステップ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、拡張が正しく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 |
+-----+-----+-----+-----+
|          BACKEND UUID  | STATE | MANAGED  |
+-----+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san |
block      | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

NFSボリュームを拡張する

Tridentは、ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、および`azure-netapp-files`バックエンドでプロビジョニングされたNFS PVのボリューム拡張をサポートしています。

ステップ1：ボリューム拡張をサポートするようにStorageClassを設定する

NFS PVのサイズを変更するには、管理者はまず、`allowVolumeExpansion`フィールドを`true`に設定して、ボリューム拡張を可能に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：

```
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
```

このオプションを使用せずにすでにストレージクラスを作成している場合は、`kubectl edit storageclass`を使用して既存のストレージクラスを編集し、ボリュームの拡張を可能にすることができます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

```
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 に対して 20 MiB 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                CAPACITY  ACCESS MODES
RECLAIM POLICY      STATUS    CLAIM                STORAGECLASS  REASON
```

```
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20 MiBのPVを1 GiBにサイズ変更するには、PVCを編集して`spec.resources.requests.storage`を1 GiBに設定します：

```
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
# ...
```

ステップ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ことを検証で

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          CAPACITY  ACCESS MODES
RECLAIM POLICY STATUS    CLAIM          STORAGECLASS  REASON
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      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

RWX NVMe サブシステムの制限について

ReadWriteMany (RWX) ボリュームのうち、NVMeプロトコルを使用するものは、ボリュームあたり64ノードという拡張性の制限があります。以下では、制限事項、関連するNVMeサブシステムアーキテクチャの説明、および必要な解決手順の概要を示します。

64ノード制限について理解する

ReadWriteMany (RWX) ボリュームをNVMeプロトコルで使用する予定がある場合、Kubernetesクラスターでは、単一のRWX NVMeボリュームを64ノード以上でマウントすることはできません。

同じRWX NVMe PersistentVolumeClaim をマウントするワークロードを64ノードを超えてスケジュールしないでください。

この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。

NVMeサブシステムモデルを理解する

ボリュームごとのサブシステムモデル (**Trident 26.02**より前のリリース)

Trident 26.02より前のリリースでは、RWX NVMeボリュームはボリュームごとのサブシステムモデルを使用してプロビジョニングされます。各RWX NVMeボリュームは、ONTAP上の専用のNVMeサブシステムにマッピングされます。

このモデルはシンプルですが、スケーラビリティの上限が低くなっています。大規模な Kubernetes クラスターでは、各 RWX ボリュームが専用のサブシステムを消費するため、サブシステムコントローラの制限にすぐに達してしまいます。

スーパーサブシステムモデル (**Trident 26.02**で導入)

Trident 26.02以降、RWX NVMeボリュームは共有スーパーサブシステムモデルを使用します。複数のRWX NVMeボリュームが同じNVMeサブシステムを共有します。

各スーパーサブシステムは、最大1024個のネームスペース (ボリューム) をサポートします。このモデルはRWXワークロードのスケーラビリティを大幅に向上させ、ONTAPサブシステムの制限に達する可能性を低減します。

各RWX NVMeボリュームは最大64ノードをサポートします。

エラーの兆候を特定する

RWX NVMe ボリュームを大規模に作成または接続すると、次のようなエラーが発生する可能性があります：

```
Maximum number of controllers reached. No more controllers can be created.
```

このエラーは、ONTAP NVMe サブシステムコントローラの制限に達したことを示しています。

サブシステム制限エラーを解決する

ボリュームごとのサブシステム制限を超えてスーパーサブシステムモデルを活用するには、Trident 26.02以降にアップグレードしてください。

スーパーサブシステムモデルを適用するために **Trident** をアップグレードする

RWX NVMe ボリュームにスーパーサブシステムモデルを適用するには：

1. Trident をバージョン 26.02 以降にアップグレードしてください。
2. RWX NVMeボリュームを使用するすべてのポッドのレプリカ数をゼロに縮小します。
3. ワークロードがRWX NVMeボリュームをアクティブに使用していないことを確認してください。
4. ポッドをスケールアップして元に戻します。

この再起動シーケンスにより、RWX NVMe ボリュームがスーパーサブシステムモデルを使用して接続されることが保証されます。

- この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。
- 64ノードの制限は、RWX NVMe ボリュームごとに適用されます。
- その他のアクセスモードおよびプロトコルには影響はありません。

コントローラーのスケラビリティ

Tridentは、複数のストレージドライバ間での同時実行性を向上させることで、コントローラーのスケラビリティを改善します。コントローラーのスケラビリティを有効にすると、Tridentコントローラーはストレージ操作を直列処理するのではなく並列処理するため、同時実行操作が多いKubernetes環境でのスループットが向上します。

デプロイする前に、Trident 26.06において、どの Trident ドライバーが一般提供時にコントローラーのスケラビリティをサポートし、どのドライバーがテクニカルプレビューとして利用可能かを確認してください。これにより、十分な情報に基づいたデプロイの意思決定を行い、リスクを管理することができます。コントローラーのスケラビリティは、デフォルトでは無効になっています。

主要な概念と定義

コントローラーのスケラビリティ

コントローラーのスケラビリティとは、Trident コントローラーが、複数のストレージ操作を単一のロックの背後で直列化するのではなく、並列に処理できる能力のことです。これらの操作には、ボリュームの作成、削除、サイズ変更、スナップショットの作成と削除、ボリュームの公開と非公開、およびバックエンド管理が含まれます。

コントローラーのスケラビリティを有効にすると、異なるボリュームやバックエンドでの操作が同時に実行されます。これにより、PersistentVolumeClaimおよびVolumeSnapshotの同時操作数が多い環境でのスループットが向上し、エンドツーエンドの操作時間が短縮されます。

デフォルトの動作（シリアルモード）

デフォルトでは、Tridentコントローラーは一度に1つの操作を処理します。作成、削除、サイズ変更、またはスナップショットの各リクエストは、Tridentが次の処理を開始する前に完了します。シリアルモードはすべてのインストールでサポートされているデフォルトであり、設定は不要です。

ほとんどのワークロードでは、シリアルモードで十分です。コントローラーのスケラビリティを有効にするのは、処理量が増加した結果、シリアル処理でバックログが発生した場合のみにしてください。

コントローラーのスケラビリティサポート

Tridentは、さまざまなストレージドライバに対して、異なる成熟度レベルでコントローラーの拡張性をサポートします。

一般提供

以下のドライバーは、Trident 26.06 の一般提供開始時点でコントローラーのスケラビリティをサポートしています：

- `ontap-san`

- ontap-nas
- ontap-nas-economy
- ontap-san-economy
- google-cloud-netapp-volumes
- azure-netapp-files
- solidfire-san



``google-cloud-netapp-volumes``と``google-cloud-netapp-volumes-san``のドライバーは異なります。Tridentは``google-cloud-netapp-volumes``のみをサポートします。バックエンドの設定や例では``google-cloud-netapp-volumes-san``を使用しないでください。

技術プレビュー

以下のドライバは、Trident 26.06においてテクニカルプレビュー機能としてコントローラのスケーラビリティをサポートしています：

- asa-r2（SANおよびNVMe）

このドライバーには以下の制限があります：

- コントローラの並行処理機能は、評価およびテスト目的でのみ利用可能です。
- 動作はリリースごとに変化する可能性があります。
- NetApp は実稼働環境での使用を推奨しません。

enableConcurrency の仕組み

``enableConcurrency``を``true``に設定すると、Tridentはコントローラが管理するすべてのバックエンドに並行処理を適用します。この設定は、すべてのバックエンドに同時に適用されます。個々のバックエンドや個々のドライバーに対して有効にすることはできません。

設定済みのバックエンドはすべて、一般提供版またはテクニカルプレビュー版のリストに掲載されているドライバを使用する必要があります。バックエンドがサポートされていないドライバを使用している場合、Tridentは起動せず、Tridentはサポートされていないドライバを使用するバックエンドを追加しません。

起動を復元するには、サポートされていないドライバを使用しているバックエンドを削除または再構成するか、コントローラの拡張性を無効にしてください。

有効にする前に



コントローラーのスケーラビリティを有効にする前に、設定されているすべてのバックエンドが、一般提供版またはテクニカルプレビュー版のリストに含まれるドライバーを使用している

ことを確認してください。バックエンドがサポートされていないドライバーを使用している場合、この機能を有効にした後、Trident は起動しません。

コントローラーのスケラビリティを有効にするかどうかは、以下の表を参考にしてください。

お使いの環境が	操作
サポートされているドライバーのみを使用し、コントローラー操作のバックログが発生します	コントローラーのスケラビリティを有効にする。
サポートされていないドライバーを使用する	デフォルトのシリアルモードを維持します。コントローラーのスケラビリティを有効にしないでください。
処理量が少なく、バックログなしで対応できます。	デフォルトのシリアルモードを維持します。

コントローラーのスケラビリティを有効にする

この `enableConcurrency` 設定オプションは、コントローラーのスケラビリティを制御します。このオプションは、Trident のインストール時、または既存のデプロイメントを更新する際に、明示的に有効にする必要があります。

Tridentオペレーターの配置

Trident operatorでコントローラの拡張性を有効にするには、`TridentOrchestrator` カスタムリソース（CR）で `enableConcurrency` を `true` に設定します。

新規インストール

`TridentOrchestrator` CR を作成または編集し、`enableConcurrency` を `true` に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

CR を適用します：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

既存のインストール

既存の TridentOrchestrator CR にパッチを適用して、コントローラーのスケラビリティを有効にし

す：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

Trident が設定を適用したことを確認します：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm デプロイメント

Helm でコントローラーのスケラビリティを有効にするには、`enableConcurrency` の値を `true` に設定します。

新規インストール

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

既存のインストール

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

または、カスタムの `values.yaml` ファイルで `enableConcurrency` を `true` に設定します：

```
# values.yaml
enableConcurrency: true
```

次に、値ファイルを使用してインストールまたはアップグレードします。

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl デプロイメント

コントローラーのスケラビリティを `tridentctl` で有効にするには、インストール中に `--enable-concurrency` フラグを渡してください。

新規インストール

```
tridentctl install -n trident --enable-concurrency
```

既存のインストール

既存の `tridentctl` デプロイメントでコントローラーのスケラビリティを有効にするには、Trident をアンインストールし、次のフラグを付けて再インストールしてください：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

コントローラーのスケラビリティが有効になっていることを確認します。

コントローラーのスケラビリティを有効にした後、Trident コントローラーが同時実行を有効にして動作していることを確認してください。コントローラポッドのログを確認してください：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

出力には、並行処理が有効になっていることを確認するログエントリが含まれています。

並行動作

コントローラーのスケラビリティが有効になっている場合、Trident コントローラーは以下の動作を適用します：

- Tridentは、単一のグローバルロックをリソースごとのきめ細かいロックに置き換えます。
- Tridentは、同じリソースを変更する操作を直列化することで、データの一貫性を維持します。
- リソースから読み取りのみを行う操作は、そのリソースに対する他の読み取り操作と並行して実行されます。
- Tridentは、バックエンドストレージシステムの過負荷を防ぐため、管理LIFあたりの同時ONTAP APIリクエスト数を20に制限します。
- 複数のバックエンドが同じ管理LIFを共有している場合、この20リクエストの制限も共有されます。

注意点と制限事項

Trident 26.06 のコントローラーのスケラビリティには、以下の考慮事項が適用されます：

- コントローラーのスケラビリティは、一般提供版およびテクニカルプレビュー版のリストに掲載されているドライバーのみをサポートします。詳細については、[有効にする前に](#)を参照してください。
- Trident コントローラーは内部的に並行処理を管理します。このリリースでは、ユーザーが設定可能な同時実行制限は提供されていません。
- 全体の処理能力は、使用するストレージドライバ、バックエンドの応答性、およびKubernetes APIサーバーのパフォーマンスに依存します。

- 同時接続数が多いと、バックエンドのストレージシステムへの負荷が増加する可能性があります。
- コントローラの拡張性に関する動作は、すべてのドライバで同一ではありません。
- テクニカルプレビュー版のドライバーは、高負荷時にパフォーマンスが不安定になる場合があります、リリースごとに動作が変わる可能性があります。
- 並列実行のため、同時実行処理のデバッグはより複雑になる場合があります。メトリクスとログには、インターリーブされた操作出力が表示される場合があります。

推奨

コントローラのスケラビリティを有効にする前に、以下の手順を完了してください。

1. 設定されているすべてのバックエンドが、一般提供版またはテクニカルプレビュー版のリストに掲載されているドライバを使用していることを確認してください。
2. 本番環境に適用する前に、非本番環境のクラスターで変更内容をテストしてください。
3. 変更を適用した後、コントローラのスケラビリティが有効になっていることを確認してください。

コントローラのスケラビリティを運用する際には、以下の一般的な推奨事項を適用してください。

- 高い拡張性が求められる本番環境では、汎用ドライバを使用してください。
- テクニカルプレビュー版ドライバは、本番環境以外でのみ評価してください。
- 大規模運用を行う際は、バックエンドとコントローラのパフォーマンスを監視してください。
- 自動化スクリプトでは、操作の順序を前提としないでください。

自動ボリューム拡張

自動ボリューム拡張により、Trident によってプロビジョニングされた永続ボリュームは、使用容量が定義されたしきい値に達すると自動的に拡張されます。この機能により、運用上の負担が軽減され、容量不足によるアプリケーションの中断を防ぐことができます。自動ボリューム拡張は、Autogrow ポリシーを使用して実装されます。Autogrow ポリシーでは以下を定義します：

- 拡張をトリガーする利用率の閾値
- ボリュームが増加する量
- ボリュームが到達できる最大サイズ



定義された使用率しきい値を超えると、ボリュームのサイズは自動的に増加します。ボリュームが自動的に縮小されることはありません。

要件

ボリュームの自動拡張を構成する前に、次の要件が満たされていることを確認してください。

- Trident 26.02以降

- カスタムリソースを作成するためのロールベースのアクセス制御権限 `TridentAutogrowPolicy`
- `StorageClasses`で構成 `allowVolumeExpansion: true`
- サポート対象の ONTAP プロトコル：
 - Network File System (NFS)
 - Internet Small Computer Systems Interface (iSCSI)
 - 不揮発性メモリエクスプレス (NVMe)
 - Fiber Channel Protocol (FCP)

制限事項

- ONTAP 9.16.1 より前の ONTAP Non-Volatile Memory Express raw ブロックボリュームは、自動拡張をサポートしていません。
- ストレージエリアネットワークボリュームの場合、`growthAmount` が50メビバイト以下であれば、Tridentは、結果のサイズが`maxSize`を超えない限り、サイズ変更前に値を自動的に51メビバイトに増やします。
- 既存環境においては、ボリューム公開の移行動作により、一部の既存ボリュームで自動拡張が機能しない場合があります。
- ボリュームが`maxSize`に達すると、それ以上の拡張は発生しません。
- 自動ボリューム拡張でサポートされているプロトコル：
 - Network File System (NFS)
 - Internet Small Computer Systems Interface (iSCSI)
 - 不揮発性メモリエクスプレス (NVMe)
 - Fiber Channel Protocol (FCP)

ボリュームの自動拡張ポリシーによるプロビジョニング

自動拡張ポリシーは、次の2つのレベルで構成できます：

- ストレージクラスレベル：すべてのボリュームのデフォルトを設定します（アノテーションを使用）
- PVC レベル：ストレージクラスのデフォルトをオーバーライドします（アノテーションを使用）

自動成長ポリシーを作成する

自動拡張ポリシーを使用すると、ボリュームが定義された容量しきい値に達したときにボリュームを自動的に拡張できます。

以下のものを用意してください：

- Trident 26.02以降がインストールされている
- `TridentAutogrowPolicy` リソースを作成するためのロールベースのアクセス制御権限
- ワークロードの増加要件の理解

ボリュームの自動拡張ポリシーは、定義された容量しきい値に達したときにボリュームが自動的に拡張される方法を定義します。

自動拡張ポリシーはワークフローのどの時点でも作成できます：

- StorageClassesとボリュームが作成される前
- StorageClassesが存在した後
- ボリュームのプロビジョニング後

この柔軟性により、既存のリソースを再作成せずに自動拡張を導入できます。

自動拡張ポリシーの仕様

自動拡張ポリシーは、次のように定義される Kubernetes カスタムリソースです：

フィールド	概要	フォーマット	必須	例	デフォルト
名前	一意のポリシー識別子	文字列	はい	production-db-ポリシー	なし
usedThreshold	拡張をトリガーする容量の割合	パーセンテージ文字列	はい	「80%」	なし
growthAmount	しきい値に達したときに増加する量	パーセンテージまたはサイズ	いいえ	「10%」または「5Gi」	「10%」
maxSize	最大ボリュームサイズの制限	Kubernetesの数量	いいえ	「500Gi」	無制限

自動成長ポリシーを作成する

手順

1. 自動拡張ポリシーを定義する YAML ファイルを作成します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. クラスターにポリシーを適用します：

```
kubectl apply -f autogrow-policy.yaml
```

3. ポリシーが作成されたことを確認します：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

期待される出力

NAME	USED THRESHOLD	GROWTH AMOUNT	STATE
standard-autogrow	80%	10%	Success

ポリシーの状態

ポリシーを作成したら、Tridentが仕様を検証し、次のいずれかの状態を割り当てます：

状態	概要	必要なアクション
成功	ポリシーは検証され、使用できる状態です。	なし。
失敗	検証エラーが検出されました。	仕様を確認して修正してください。
削除中	削除中です。	完了するまでお待ちください。

ポリシーをStorageClassに関連付ける

`trident.netapp.io/autogrowPolicy` アノテーションを使用して、自動拡張ポリシーをStorageClassに関連付けることができます。そのStorageClassからプロビジョニングされたすべてのボリュームは、ポリシーを継承します。



StorageClassには `allowVolumeExpansion: true` が必要です。

手順

1. Autogrow Policyアノテーションを使用してStorageClassを作成または変更します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
```

```
fsType: "ext4"
allowVolumeExpansion: true
```

2. StorageClassを適用します。

```
kubectl apply -f storageclass.yaml
```

3. アノテーションを確認します：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

期待される出力

```
production-db-policy
```

ポリシーの優先順位

自動拡張ポリシー注釈がStorageClassとPVCの両方に設定されている場合、Tridentは次の優先順位ルールを適用します：

1. *PVC注釈が優先されます。*PVCが `trident.netapp.io/autogrowPolicy` を設定した場合、その値が常に使用されます。
2. **StorageClass** アノテーションは、**PVC** にアノテーションがない場合にのみ適用されます。
3. *どちらにもアノテーションがない場合、*ボリュームの自動拡張ポリシーは適用されません。

StorageClass アノテーション	PVC アノテーション	効果的な行動
trident.netapp.io/autogrowPolicy: standard-agp	未設定	用途 standard-agp
trident.netapp.io/autogrowPolicy: standard-agp	trident.netapp.io/autogrowPolicy: logs-policy	使用 logs-policy (PVCはStorageClassをオーバーライドします)。
trident.netapp.io/autogrowPolicy: standard-agp	trident.netapp.io/autogrowPolicy: "none"	自動拡張ポリシーなし (PVC は自動拡張を無効にします)。
未設定	trident.netapp.io/autogrowPolicy: dev-policy	用途 dev-policy
未設定	未設定	自動拡張ポリシーなし。

設定例

次の例は、さまざまなユースケースにおける一般的な Autogrow Policy 構成を示しています。

本番データベースに対する保守的なポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

固定増分によるログストレージ

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

デフォルト付きの最小限のポリシー

`growthAmount`と `maxSize`を省略すると、Tridentはデフォルトの(`10%` (成長、無制限のサイズ)を使用します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

カスタムmaxSizeおよびデフォルトgrowthAmountを使用したポリシー

```
apiVersion: trident.netapp.io/v1
```

```
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

無制限の**maxSize**による積極的な成長

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

小数パーセンテージを含むポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```



小数点以下のパーセンテージがサポートされています。小数点以下3桁以上を指定すると、Tridentは値を小数点以下3桁に丸めます。

NAS StorageClass（自動拡張付き）

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
```

```
allowVolumeExpansion: true
```

SAN StorageClass（自動拡張付き）

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

自動成長ポリシーの管理

自動成長ポリシーを作成した後、それらを表示、更新、削除することができます。特定のポリシーを使用しているボリュームを監視することもできます。

自動拡張ポリシーを表示

すべてのポリシーを一覧表示する

`kubectl`を使用して、クラスター内のすべての自動拡張ポリシーを一覧表示します：

```
kubectl get tridentautogrowpolicy
```

あるいは、tridentctl：

```
tridentctl get autogrowpolicy
```

ポリシーの詳細を表示

ポリシーの完全な仕様とステータスを表示するには：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

YAML 形式でポリシーとそれに関連付けられたボリュームを表示するには：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

自動成長ポリシーを更新する

既存のポリシーを変更して、しきい値、増加量、または最大サイズを変更できます。変更は、ポリシーを使用するすべてのボリュームに対して直ちに有効になります。



変更内容は、当該ポリシーを使用するすべてのボリュームに影響します。可能であれば、まず非本番環境で変更をテストしてください。

手順

1. ポリシーを編集します：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

2. 必要に応じて `spec` フィールドを変更します：

```
spec:
  usedThreshold: "75%"      # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

3. 保存して終了します。変更は直ちに有効になります。

更新に関する考慮事項

- 即時適用： このポリシーを使用するすべてのボリュームは、次回の成長評価時に新しいパラメータを採用します。
- ボリュームの再起動は不要： 変更は次回の拡張操作に適用されます。
- **Test first:** 可能な場合は、非本番環境で変更を検証します。
- *変更を伝える：*共有ポリシーを変更するときにチームに通知します。

自動成長ポリシーを削除する

自動拡張ポリシーは、ボリュームがアクティブに使用されている間、誤って削除されないようにファイナライザー保護を使用します。

手順

1. ポリシーを削除します：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. ボリュームがまだポリシーを使用している場合、削除は `Deleting` 状態になります。影響を受けるボリュームを確認してください。

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 影響を受ける各ボリュームからポリシーを削除します。次のいずれかのオプションを選択します。

- オプションA：明示的に自動拡張を無効にする アノテーションを `none` に設定します：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

- オプションB：注釈を完全に削除する：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

削除動作

シナリオ	動作
このポリシーを使用するボリュームはありません	ポリシーは直ちに削除されます。
ボリュームはポリシーを使用します	ポリシーが `Deleting` 状態になります。ファイナライザーは、すべてのボリュームが削除されるまで完了をブロックします。
すべてのボリュームがポリシーから削除されます	ファイナライザーが削除され、ポリシーが削除されます。

自動拡張ポリシーの使用状況を監視する

ポリシーを使用してボリュームをチェックする

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

ボリュームが使用するポリシーを確認する

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

ポリシーイベントを監視する

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

サポートされるプロトコル

Autogrow は次のストレージプロトコルをサポートしています：

- NFS
- iSCSI
- FCP
- NVMe



SANボリュームの場合、構成された `growthAmount` が50 MiB以下である場合、Tridentは、結果のサイズが `maxSize` を超えない限り、サイズ変更操作の増加量を自動的に51MBに増加します。

サポートされているバックエンド

AutogrowはONTAPバックエンドをサポートしています。AutogrowはNASボリュームとSANボリュームの両方に対応するGoogle Cloud NetApp Volumes (GCNV) もサポートしています。

既知の制限事項

- **ONTAP NVMe raw** ブロックボリューム： ONTAP 9.16.1より前のバージョンで作成されたボリュームは、自動拡張をサポートしていません。
- 既存のボリューム（ブラウンフィールド展開）： 有効な自動拡張ポリシーが適用されていても、既存のボリュームでは自動拡張が機能しない場合があります。この制限は、ボリュームパブリケーションの移行が進行中であることに起因します。移行が完了したことを確認するには、Trident コントローラーログで "Migration completed" メッセージを確認してください。

よくある質問

Tridentはいつ閾値を評価しますか？

Tridentはボリュームの使用状況を継続的に監視します。使用容量が `usedThreshold` を超えると、Tridentは内部サイズ変更要求を作成し、設定された `growthAmount` でボリュームを拡張します。

たとえば、このポリシーは容量の80%で拡張をトリガーし、ボリュームを毎回10%ずつ最大500 GiBまで増加させます：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
```

```
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

ボリュームがすでにプロビジョニングされた後にポリシーを適用できますか？

はい。自動拡張ポリシーはいつでも作成でき、`trident.netapp.io/autogrowPolicy` アノテーションを追加または更新することで既存のPVCに適用できます。PVCまたはStorageClassを再作成する必要はありません。

既存の PVC にポリシーを適用するには：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

既存のStorageClassにポリシーを適用するには：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

StorageClassと**PVC**の両方に自動拡張ポリシーを設定するとどうなりますか？

PVC 注釈は常に優先されます。PVC に `trident.netapp.io/autogrowPolicy` 注釈がある場合、Trident はStorageClassの指定に関係なくその値を使用します。詳細については、["ポリシーの優先順位"](#)を参照してください。

例えば、このStorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

そして、この PVC は StorageClass ポリシーを上書きします：

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```

metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
    storageClassName: ontap-gold

```

Tridentは`logs-policy`を`database-pvc`に使用し、`standard-agp`は使用しません。

特定のボリュームの自動拡張を無効にするにはどうすればよいですか？

PVC アノテーションを`"none"`に設定します。この設定は、そのボリュームに対する StorageClass レベルのポリシーを上書きします：

```

kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

```

自動拡張が無効になっていることを確認できます：

```

kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'

```

期待される出力

```

none

```

ボリュームが**maxSize**に達するとどうなりますか？

Tridentはボリュームの拡張を停止します。使用量が上限を超えて増加し続けても、そのボリュームに対するサイズ変更要求はそれ以上発生しません `usedThreshold`。

例えば、このポリシーでは、Trident はボリュームが 100 GiB に達した後、それ以上拡張しません：

```

apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:

```

```
usedThreshold: "90%"
growthAmount: "10Gi"
maxSize: "100Gi"
```

無制限に成長させるには、`maxSize`を省略するか、`0`に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

ボリュームを再起動せずにポリシーを変更できますか？

はい。ポリシーを更新すると、そのポリシーを使用しているすべてのボリュームは、次回の成長評価時に新しいパラメータを適用します。ボリュームの再起動は不要です。

既存のポリシーを更新するには：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

必要に応じてフィールドを変更します：

```
spec:
  usedThreshold: "75%"      # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

保存して終了します。更新されたポリシーを確認します：

```
kubectl get tridentautogrowpolicy production-db-policy
```

期待される出力

NAME	USED THRESHOLD	GROWTH AMOUNT	STATE
production-db-policy	75%	20%	Success

ポリシーが失敗状態になっているのはなぜですか？

A `Failed`状態は、ポリシー仕様に検証エラーが含まれていることを示します。エラーの詳細を表示するに

は、次のコマンドを実行します：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

一般的な原因には、無効な `usedThreshold`（1～99%である必要があります）、`growthAmount` が `maxSize` を超えている場合、または無効な Kubernetes 数量フォーマットなどがあります。仕様を修正して再適用してください：

```
kubectl apply -f autogrow-policy.yaml
```

ポリシーを削除できないのはなぜですか？

ポリシーはファイナライザー保護を使用します。ボリュームがまだポリシーを使用している場合、削除は `Deleting` 状態になり、ポリシーからすべてのボリュームを削除するまで待機します。

影響を受けるボリュームを特定します：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

次に、各 PVC から注釈を削除します：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

すべてのボリュームを削除すると、Trident はファイナライザーを解放し、ポリシーを削除します。

autogrowはどのバックエンドをサポートしていますか？

AutogrowはONTAPバックエンドをサポートしています。また、NASボリュームとSANボリュームの両方について、Google Cloud NetApp Volumes（GCNV）もサポートしています。

Autogrowは、NFS、iSCSI、FCP、およびNVMeプロトコルをサポートしています。NVMe rawブロックボリュームには、ONTAP 9.16.1以降が必要です。

既存のボリュームをブラウンフィールド環境で使用している場合、自動拡張が有効になる前にボリューム公開の移行を完了する必要がある場合があります。移行状況を確認するには、Trident コントローラーログを確認してください：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration completed"
```

次のStorageClass例では、NAS および SAN バックエンドに設定された自動拡張を示しています：

NASバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SANバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN ボリュームの最小増加量はどれくらいですか？

SAN ボリュームの場合、有効な最小増加量は 51 MB です。`growthAmount`を 50 MiB 以下に設定すると、Trident はサイズ変更操作のために自動的に増加量を 51 MB に増加します。

例えば、このポリシーは`growthAmount`の`"40Mi"`を設定しますが、Tridentはこれを使用するすべてのSAN ボリュームに51MBの拡張を適用します（

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
```

```
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

この自動調整を回避するには、`growthAmount` を 50 MiB を超える値に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

Tridentボリュームのインポート

`tridentctl import`を使用するか、Tridentインポート
アノテーションを使用して永続ボリューム要求（PVC）を作成することで、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。

概要と考慮事項

次の目的でボリュームをTridentにインポートすることができます：

- 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データセットのクローンを使用する
- 障害が発生した Kubernetes クラスタを再構築する
- 災害復旧時にアプリケーションデータを移行する

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（読み書き）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。
- アクティブな接続なしでボリュームをインポートすることをお勧めします。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。



これは、Kubernetes が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、ブロックボリュームの場合に特に重要です。その結果、データ破損が発

生する可能性があります。

- ただし `StorageClass` PVCで指定する必要がありますが、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。したがって、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する場合でも、インポートは失敗しません。
- 既存のボリュームサイズが決定され、PVCに設定されます。ボリュームがストレージドライバによってインポートされると、ClaimRefを使用してPVCへの参照を持つPVが作成されます。
 - 回収ポリシーは初期設定では PV で `retain` に設定されています。Kubernetes が PVC と PV を正常にバインドすると、回収ポリシーはストレージクラスの回収ポリシーと一致するように更新されます。
 - ストレージクラスの再利用ポリシーが `delete` の場合、PVが削除されると、Tridentはストレージボリュームを削除します。
- デフォルトでは、Trident は PVC を管理し、バックエンドの FlexVol ボリュームおよび LUN または qtree の名前を変更します。`--no-manage` フラグを渡すと管理対象外のボリュームをインポートでき、`--no-rename` フラグを渡すとボリューム名を保持できます。
 - `--no-manage` - `--no-manage` フラグを使用する場合、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は実行されません。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操作も無視されます。
 - `--no-rename` - `--no-rename` フラグを使用する場合、Trident はボリュームをインポートする際に既存のボリューム名を保持し、ボリュームのライフサイクルを管理します。このオプションは、`ontap-nas`、`ontap-san` (ASA r2 システムを含む)、`ontap-san-economy`、および `ontap-nas-economy` (qtree インポート) ドライバーでのみサポートされています。



これらのオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、それ以外の場合は Kubernetes の外部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役立ちます。

- PVC と PV に注釈が追加されます。注釈には、ボリュームがインポートされたことと、PVC と PV が管理されているかどうかを示すという 2 つの目的があります。この注釈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する

`tridentctl import` を使用するか、Trident インポートアノテーションを使用して PVC を作成することで、ボリュームをインポートできます。



PVC アノテーションを使用する場合は、`tridentctl` をダウンロードしたり使用してボリュームをインポートしたりする必要はありません。

tridentctl の使用

手順

1. PVCを作成するために使用するPVCファイル（例：`pvc.yaml`）を作成します。PVCファイルには `name`、`namespace`、`accessModes`、および `storageClassName` を含める必要があります。必要

に応じて、PVC定義で `unixPermissions` を指定することもできます。

以下は最小仕様の例です：

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



必須パラメータのみを入力してください。PV名やボリュームサイズなどの追加パラメータは、インポートコマンドの失敗の原因となる可能性があります。

2. `tridentctl import` コマンドを使用して、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（例：ONTAP FlexVol、Element Volume）を指定します。`-f` 引数は、PVCファイルへのパスを指定するために必要です。

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

PVCアノテーションの使用

手順

1. 必要なTridentインポートアノテーションを含むPVC YAMLファイル（例：pvc.yaml）を作成します。PVCファイルには以下を含める必要があります：

- `name` および `namespace` メタデータ内
- `accessModes`、`resources.requests.storage`、および `storageClassName` 仕様
- 注釈：
 - `trident.netapp.io/importOriginalName`：バックエンドのボリューム名
 - `trident.netapp.io/importBackendUUID`：ボリュームが存在するバックエンドUUID
 - `trident.netapp.io/notManaged`（オプション）：管理されていないボリュームの場合は `true` に設定します。デフォルトは `false` です。

以下は、管理対象ボリュームをインポートするための仕様例です：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
```

```
namespace: <namespace>
annotations:
  trident.netapp.io/importOriginalName: "<volume-name>"
  trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
    storageClassName: <storage-class-name>
```

2. PVC YAML ファイルを Kubernetes クラスターに適用します：

```
kubectl apply -f <pvc-file>.yaml
```

Trident はボリュームを自動的にインポートし、PVC にバインドします。

例

サポートされているドライバーの次のボリュームインポート例を確認してください。

ONTAP NAS と ONTAP NAS FlexGroup

Trident は、`ontap-nas` および `ontap-nas-flexgroup` ドライバー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-economy` ドライバーは qtree レベルでストレージを管理するため、単独での FlexVol volume のインポートはサポートしていません。`ontap-nas-economy` ドライバーを使用して qtree をインポートするには、[qtrees をインポート](#) (Tech Preview) を参照してください。Trident が qtree をインポートする際、その qtree を含む親 FlexVol もバケットとしてインポートします。
- `ontap-nas` および `ontap-nas-flexgroup` ドライバーは、重複するボリューム名を許可しません。

`ontap-nas` ドライバーで作成された各ボリュームは、ONTAP クラスター上の FlexVol ボリュームです。`ontap-nas` ドライバーを使用した FlexVol ボリュームのインポートも同様に機能します。ONTAP クラスターにすでに存在する FlexVol ボリュームは、`ontap-nas` PVC としてインポートできます。同様に、FlexGroup ボリュームは `ontap-nas-flexgroup` PVC としてインポートできます。

tridentctl を使用した ONTAP NAS の例

次の例は、`tridentctl` を使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、`ontap\_nas`というバックエンド上の`managed\_volume`というボリュームをインポートします：

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

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

管理されていないボリューム

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

次の例は、`unmanaged\_volume`を`ontap\_nas`バックエンドでインポートします：

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

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

PVC アノテーションを使用した ONTAP NAS の例

次の例は、PVC アノテーションを使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、`81abcb27-ea63-49bb-b606-0a5315ac5f21` から `ontap\_volume1` という名前の `ontap-nas` ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

管理されていないボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、バックエンド `34abcb27-ea63-49bb-b606-0a5315ac5f34` からという名前 `ontap-volume2` の 1Gi `ontap-nas` ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <unmanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
storageClassName: <storage-class-name>
```

qtree をインポート



qtree インポートは Trident 26.06 のテクニカルプレビュー機能です。テクニカルプレビュー機能は、本番環境での使用はサポートされていません。

Trident 26.06以降、`ontap-nas-economy`ドライバを使用して、既存のONTAP qtreeをTridentにインポートできます。FlexVol volumeとqtree名を`flexvol/qtree`形式で指定し、それを含むバックエンドのUUIDとともにqtreeを識別します。[\[概要と考慮事項\]](#)の一般的なインポートに関する考慮事項は、qtreeのインポートにも適用されます。

tridentctlを使用してqtreeをインポートする

qtreeを`flexvol/qtree`形式でボリューム名として指定します。次の例では、バックエンド`ontap\_nas\_economy`上のFlexVol volume `flexvol1`からqtree `qtree1`をインポートします。

```
tridentctl import volume ontap_nas_economy flexvol1/qtree1 -f <path-to-pvc-file>
```

qtree の名前を変更せずにインポートするには、`--no-rename`フラグを追加してください。Trident がライフサイクルを管理せずに qtree をインポートするには、`--no-manage`フラグを追加してください。

PVCアノテーションを使用してqtreeをインポートする

以下のアノテーションを付けてPVCを作成します：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "flexvol1/qtree1"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: <size>
      storageClassName: <storage-class-name>
```

既存のFlexVol volumeとqtree名を保持するには、`trident.netapp.io/importNoRename: "true"`アノテーションを追加します。qtreeをアンマネージドとしてインポートするには、`trident.netapp.io/notManaged: "true"`アノテーションを追加します。

命名と管理の動作

管理インポート（デフォルト）

Trident は、qtree の名前を PVC 名に合わせて変更し、FlexVol ボリュームの名前を `ontap-nas-economy` 命名規則に合わせて変更します。ジャンクションパスは変更されません。

--no-rename

Trident は既存の FlexVol volume と qtree 名を保持します。名前はすでに Trident の命名規則に従っている必要があります。従っていない場合、Trident は qtree の削除またはマウント操作を管理できません。

--no-manage

Trident は qtree を参照しますが、そのライフサイクルは管理しません。名前は Trident の命名規則に従う必要があります、Trident は qtree の自動エクスポートポリシーを使用または管理しません。

エクスポートポリシーの動作

バックエンドが自動エクスポートポリシーを使用している場合、インポートされた qtree は qtree ごとのエクスポートポリシーモデルを使用し、各 qtree が独自のエクスポートポリシーを持ちます。Trident はインポート時に qtree に空のエクスポートポリシーを割り当てます。公開時に、自動エクスポートが有効になっており、既存の FlexVol エクスポートポリシーが `default` または空の場合、Trident は FlexVol ボリュームをバックエンドごとのエクスポートポリシーに更新します。



FlexVol ボリュームのエクスポートポリシーを更新すると、同じ FlexVol ボリューム内で現在マウントされている他の qtree に影響を与える可能性があります。qtree をインポートするのは、FlexVol ボリューム内の qtree がマウントされていない場合のみにしてください。



2 つの Kubernetes クラスターを同じアクティブな SVM に向けないでください。また、別のクラスターが引き続き管理している qtree をインポートしないでください。両方のクラスターが同じボリュームを管理している場合、競合する操作によってデータが破損または消失する可能性があります。

ONTAP SAN

Trident は、ontap-san (iSCSI、NVMe/TCP、FC) および `ontap-san-economy` ドライバー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。

Trident 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 `ontap-san` ドライバと一致しており、各 PVC に対して FlexVol ボリュームを作成し、FlexVol ボリューム内に LUN を作成します。Trident は FlexVol ボリュームをインポートし、PVC 定義に関連付けます。Trident は、複数の LUN を含む `ontap-san-economy` ボリュームをインポートできます。

次の例は、管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています：

管理ボリューム

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

次の例では、`ontap-san-managed` バックエンド上に存在する FlexVol volume を `ontap\_san\_default` インポートします：

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

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

管理されていないボリューム

次の例は、`unmanaged\_example\_volume`を`ontap\_san`バックエンドでインポートします：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
```

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

次の例に示すように、Kubernetes ノード IQN と IQN を共有する igroup にマップされた LUN がある場合は、次の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、イニシエーターを削除するか、LUN のマップ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

ONTAP SAN-economyの例

以下の例は、`ontap-san-economy`バックエンドの管理ボリュームと非管理ボリュームをインポートする方法を示しています。

管理ボリューム

管理ボリュームをインポートすると、TridentがFlexVolの所有権を取得し、名前を変更します。同じFlexVolから複数のLUNをインポートする場合は、この名前変更を考慮する必要があります。

次の例では、`lun1`をFlexVol `toimport`から`vol-managed-saneco`という名前の管理ボリュームとしてインポートします：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

インポート後`lun1`、TridentはFlexVolの名前を変更します（例：`trident\_lun\_pool\_xyz`）。同じFlexVolから追加のLUNをインポートするには、新しいFlexVol名を使用します：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```



`ontap-san-economy`バックエンドは一度に1つのLUNをインポートします。スクリプトを使用すれば、複数のインポートを自動化できます。

管理されていないボリューム

管理されていないボリュームをインポートすると、TridentはFlexVolの所有権を取得しません。ただし、FlexVolとLUNはTridentの命名規則に従う必要があります。

FlexVol命名形式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX`は、バックエンド設定の`storagePrefix`の値です。デフォルトは`trident`です。
- `RANDOMSTRING`は任意の文字列です。

LUN の命名要件

LUN には`lun0`という名前を付ける必要があります。

例

もしあなたの`storagePrefix`が`xyz`の場合、LUNへの完全パスは次のとおりです：

```
trident_lun_pool_xyz_randomstring/lun0
```

要素

Tridentは、`solidfire-san`ドライバを使用して、NetApp Element ソフトウェアおよびNetApp HCI ボリュームのインポートをサポートします。



Element ドライバは重複したボリューム名をサポートします。ただし、Trident は重複したボリューム名がある場合はエラーを返します。回避策として、ボリュームのクローンを作成し、一意のボリューム名を指定して、クローンされたボリュームをインポートします。

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

```
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      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

Azure NetApp Files

Tridentは`azure-netapp-files`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。



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識別します。ボリュームパスは、`:/`の後のボリュームのエクスポートパスの部分です。たとえば、マウントパスが`10.0.0.2:/importvol1`の場合、ボリュームパスは`importvol1`です。

次の例は、バックエンド `azurenetafiles\_40517` 上の `azure-netapp-files` ボリュームを、ボリュームパス `importvol1` でインポートします。

```
tridentctl import volume azurenetafiles_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 Volumes

Tridentは`google-cloud-netapp-volumes`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。

次の例では、ボリューム`testvoleasiaeast1`を持つバックエンド`backend-tbc-gcnv1`上のボリュームをインポートします。

```
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
+-----+-----+-----+-----+			
+-----+-----+-----+-----+			

次の例では、同じリージョンに2つのボリュームが存在する場合に`google-cloud-netapp-volumes`ボリュームをインポートします：

```
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

Tridentのボリューム名とラベルのカスタマイズ

Tridentを使用すると、作成したボリュームに意味のある名前とラベルを割り当てることができます。これにより、ボリュームを識別し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできるようになります。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すべて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：

- ボリュームの作成、インポート、およびクローン処理。
- `ontap-nas-economy`ドライバの場合、Qtreeボリュームの名前のみが名前テンプレートに準拠します。
- `ontap-san-economy`ドライバの場合、LUN名のみが名前テンプレートに準拠します。

制限事項

- カスタムボリューム名は ONTAP オンプレミスドライバーのみと互換性があります。
- カスタムラベルは、ontap-san、ontap-nas、および `ontap-nas-flexgroup` ドライバでのみサポートされます。
- カスタム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

- 名前テンプレートの構文が無効であるために障害が発生した場合、バックエンドの作成は失敗します。ただし、テンプレートの適用に失敗した場合、ボリュームは既存の命名規則に従って命名されます。
- 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に名前を付ける場合、ストレージプレフィックスは適用されません。必要な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、あるいはその両方で定義できます。

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 **template** の例

例1：

```

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

```

例2：

```

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

```

考慮すべき点

1. ボリュームのインポートの場合、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に従います。
3. Trident は、ストレージプレフィックスでのスライス演算子の使用をサポートしていません。
4. テンプレートによって一意のボリューム名が生成されない場合は、Tridentがいくつかのランダムな文字を追加して、一意のボリューム名を作成します。
5. NASエコノミーボリュームのカスタム名が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名を付けます。その他のすべてのONTAPドライバーでは、ボリューム名が名前制限を超えると、ボリューム作成プロセスが失敗します。

名前空間をまたいで**Trident NFS**ボリュームを共有する

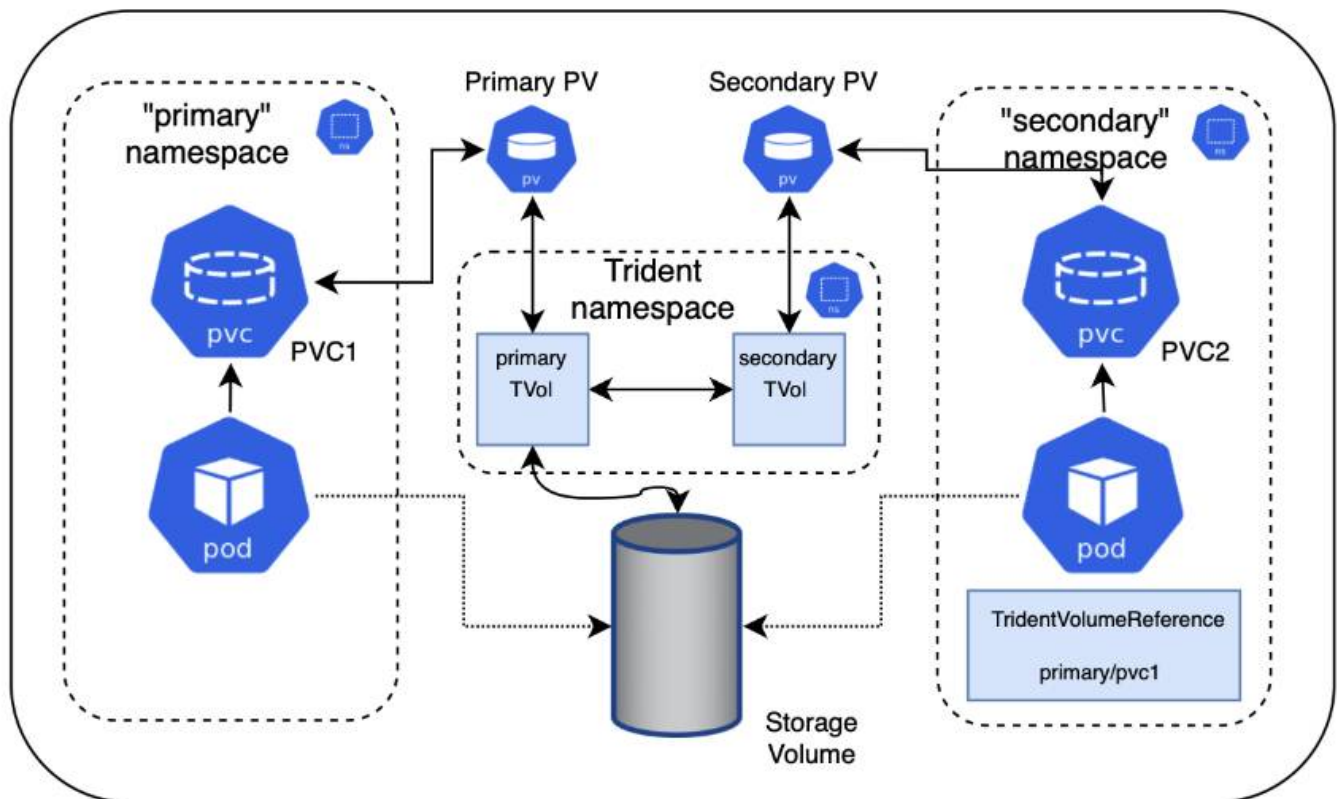
Tridentを使用すると、プライマリ名前空間にボリュームを作成し、それを1つ以上のセカンダリ名前空間で共有できます。

機能

TridentVolumeReference CR を使用すると、ReadWriteMany (RWX) NFS ボリュームを 1 つ以上の Kubernetes 名前空間にわたって安全に共有できます。この Kubernetes ネイティブソリューションには、次の利点があります：

- セキュリティを確保するための複数レベルのアクセス制御
- すべての Trident NFS ボリュームドライバーに対応
- tridentctl やその他の非ネイティブ Kubernetes 機能に依存しない

この図は、2 つの Kubernetes 名前空間間での NFS ボリュームの共有を示しています。



クイック スタート

わずか数ステップで NFS ボリューム共有を設定できます。

1

ボリュームを共有するようにソース **PVC** を設定する

ソース名前空間の所有者は、ソース PVC 内のデータにアクセスする権限を付与します。

2

宛先ネームスペースに **CR** を作成する権限を付与します

クラスタ管理者は、宛先名前空間の所有者に TridentVolumeReference CR を作成する権限を付与します。

3

宛先名前空間に**TridentVolumeReference**を作成する

宛先名前空間の所有者は、ソースPVCを参照するTridentVolumeReference CRを作成します。

4

宛先名前スペースに従属**PVC**を作成する

宛先名前スペースの所有者は、ソース PVC のデータソースを使用するために従属 PVC を作成します。

ソース名前スペースと宛先名前スペースを設定する

セキュリティを確保するには、名前空間間の共有には、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による連携とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. *ソース名前スペースの所有者*: ソース名前スペースでPVC(`pvc1`を作成し、(`namespace2`宛てに共有する権限を付与するために、`shareToNamespace`アノテーションを使用します。

```
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。



- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。たとえば、`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、`\*`を使用します。例：
``trident.netapp.io/shareToNamespace: *`
- PVCを更新して、`shareToNamespace`アノテーションをいつでも含めることができます。

2. *クラスター管理者*: 適切なRBACが設定されていることを確認し、宛先名前スペースの所有者が宛先名前スペースにTridentVolumeReference CRを作成する権限を付与します。

- 宛先名前空間の所有者：ソース名前空間を参照するTridentVolumeReference CRを宛先名前空間に作成します pvc1。

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

- 宛先ネームスペースの所有者：宛先ネームスペース(namespace2) でPVC(pvc2) を作成し、ソースPVCを指定するために`shareFromPVC`アノテーションを使用します。

```
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
```



デスティネーション PVC のサイズは、ソース 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`shareFromPVC`アノテーションを読み取り、デスティネーションPVを、独自のストレージリソースを持たず、ソースPVを指してソースPVストレージリソースを共有する従属ボリュームとして作成します。デスティネーションPVCとPVは正常に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する

複数の名前空間間で共有されているボリュームを削除できます。Tridentは、ソース名前空間のボリュームへのアクセスを削除し、ボリュームを共有する他の名前空間へのアクセスを維持します。ボリュームを参照するすべての名前空間が削除されると、Tridentはボリュームを削除します。

`tridentctl get`を使用して従属ボリュームを照会する

[`tridentctl`ユーティリティを使用して、`get`コマンドを実行すると、従属ボリュームを取得できます。詳細については、link: [../trident-reference/trident-cl.html](#)[`tridentctl`コマンドとオプション]を参照してください。

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

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを従属ソースボリュームに制限します。
- `--subordinateOf string`：ボリュームの下位にクエリを制限します。

制限事項

- Tridentは、宛先ネームスペースが共有ボリュームに書き込むのを防ぐことはできません。共有ボリュームのデータが上書きされないようにするには、ファイルロックまたはその他のプロセスを使用する必要があります。
- ソースPVCへのアクセスを取り消すには、`shareToNamespace`または`shareFromNamespace`アノテーションを削除するか、`TridentVolumeReference`CRを削除しても取り消すことはできません。アクセスを取り消すには、従属PVCを削除する必要があります。
- 従属ボリュームではスナップショット、クローン、ミラーリングは実行できません。

詳細情報

クロスネームスペースボリュームアクセスの詳細については、次を参照してください：

- ["NetAppTV"](#)でデモをご覧ください。

名前空間をまたいだTridentボリュームのクローン作成

Tridentを使用すると、同じKubernetesクラスター内の別の名前空間の既存のボリュームまたはボリュームスナップショットを使用して、新しいボリュームを作成できます。

前提条件

ボリュームを複製する前に、ソースと宛先のバックエンドが同じタイプであり、同じストレージクラスを持っていることを確認してください。



名前空間をまたがるクローン作成は、`ontap-san`および`ontap-nas`ストレージドライバでのみサポートされます。読み取り専用クローンはサポートされていません。

クイック スタート

わずか数ステップでボリュームのクローンを設定できます。

1

ボリュームをクローニングするソース **PVC** を設定します

ソース名前空間の所有者は、ソース PVC 内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します

クラスタ管理者は、宛先名前空間の所有者に TridentVolumeReference CR を作成する権限を付与します。

3

宛先名前空間に **TridentVolumeReference** を作成する

宛先名前空間の所有者は、ソース PVC を参照する TridentVolumeReference CR を作成します。

4

宛先名前空間にクローン **PVC** を作成する

宛先名前空間の所有者は、ソース名前空間から PVC をクローニングするための PVC を作成します。

ソース名前空間と宛先名前空間を設定する

セキュリティを確保するために、名前空間間でボリュームを複製するには、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による共同作業とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. ソース名前空間の所有者：ソース名前空間(namespace1) で PVC(pvc1) を作成し、cloneToNamespace アノテーションを使用して、宛先名前空間(namespace2) と共有する権限を付与します。

```
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。



- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。たとえば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、`*``を使用します。例：
``trident.netapp.io/cloneToNamespace: *`
- PVCを更新して、``cloneToNamespace``アノテーションをいつでも含めることができます。

2. クラスタ管理者：適切なRBACが設定されていることを確認し、宛先名前空間の所有者が宛先名前空間にTridentVolumeReference CRを作成する権限を付与します(namespace2)。
3. 宛先名前空間の所有者：ソースPVCを参照するTridentVolumeReference CRを宛先名前空間に作成します pvc1。

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

4. 宛先名前空間の所有者：宛先名前空間(namespace2) でPVC(pvc2) を作成し、`cloneFromPVC`または `cloneFromSnapshot、および `cloneFromNamespace`アノテーションを使用してソースPVCを指定します。`

```
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-economy ドライバーを使用してプロビジョニングされた PVC の場合、読み取り専用クローンはサポートされません。

SnapMirrorを使用したボリュームのレプリケート

Tridentは、ディザスタリカバリのためにデータをレプリケートする、1つのクラスタ上のソースボリュームとピアクラスタ上のデスティネーションボリューム間のミラー関係をサポートします。 Trident Mirror Relationship (TMR) と呼ばれる名前空間付きの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：

- ボリューム間のミラー関係を作成する (PVC)
- ボリューム間のミラー関係を削除する
- ミラー関係を解除
- 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
- 計画されたフェイルオーバーまたは移行中に、クラスタ間でのアプリケーションのロスレス移行を実行します

レプリケーションの前提条件

始める前に、次の前提条件が満たされていることを確認してください：

ONTAP クラスタ

- **Trident**：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、Tridentバージョン22.10以降が存在する必要があります。
- **ライセンス**：ONTAP SnapMirror データ保護バンドルを使用する非同期ライセンスは、ソースとデスティネーション クラスタの両方で有効にする必要があります。詳細については、["ONTAP における SnapMirror ライセンスの概要"](#)を参照してください。

ONTAP 9.10.1以降では、すべてのライセンスがNetAppライセンス ファイル (NLF) として提供されます。これは、複数の機能を有効にする単一のファイルです。詳細については、["ONTAP Oneに含まれるライセンス"](#)を参照してください。



SnapMirror非同期保護のみがサポートされています。

ピアリング

- *** クラスタと SVM ***：ONTAP ストレージ バックエンドはピアリングする必要があります。詳細については、["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。

- **Trident**および**SVM**：ピアリングされたりリモートSVMは、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。

サポートされているドライバ

NetApp Tridentは、次のドライバーでサポートされているストレージクラスを使用して、NetApp SnapMirror テクノロジーによるボリュームレプリケーションをサポートします：**ontap-nas : NFS** ontap-san : iSCSI **ontap-san : FC** ontap-san : NVMe/TCP（最小ONTAPバージョン9.15.1が必要）



SnapMirrorを使用したボリュームレプリケーションは、ASA r2システムではサポートされていません。ASA r2システムの詳細については、["ASA r2ストレージシステムの詳細"](#)を参照してください。

ミラー PVC を作成する

次の手順に従って、CRD の例を使用して、プライマリ ボリュームとセカンダリ ボリューム間の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. `trident.netapp.io/replication: true`パラメータを使用してStorageClassオブジェクトを作成します。

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）状態を取得し、MirrorRelationshipのstatusフィールドに入力します。

- 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. StorageClassをtrident.netapp.io/replication: trueパラメータで作成します。

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
  replicationPolicy: MirrorAllSnapshots
  replicationSchedule: 5min
  volumeMappings:
    - localPVCName: csi-nas
      remoteVolumeHandle:
        "datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

TMR spec`には、4つの最上位フィールドがあります：目的の関係状態を設定する `state` ([\[ボリュームレプリケーションの状態\]](#)を参照)、ローカルPVCをリモートボリュームハンドルに関連付ける volumeMappings、および以下で説明するオプションの `replicationPolicy` と `replicationSchedule` フィールドです。

`replicationPolicy` および
`replicationSchedule` フィールドはオプションです。 `replicationPolicy`
にはONTAPの非同期SnapMirrorポリシーの名前を設定し、
`replicationSchedule` にはONTAPスケジュールの名前を設定します。いずれかのフィ
ールドを省略した場合、Tridentはバックエンドで設定されている対応するレプリケーショ
ンポリシーまたはスケジュールを使用します。これらのフィールドは関係を作成する際に設
定します。関係が確立された後にポリシーやスケジュールを変更することはできません。異
なる設定を使用するには、TMRを削除して再作成します。

Tridentは、設定されたレプリケーションポリシーとスケジュールを使用して、SnapMirror関係を作成および初期化します。

1. 以前に作成したStorageClassを使用してPVCを作成し、セカンダリ（SnapMirrorデスティネーション）として機能させます。

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、Tridentは新しいFlexVolボリュームが、MirrorRelationshipで定義されたりリモートSVMとピアリングされたSVM上に配置されるようにします。

ボリュームレプリケーションの状態

Tridentミラー リレーションシップ (TMR) は、PVC間のレプリケーション リレーションシップの一方の端を表すCRDです。デスティネーションTMRには状態があり、Tridentに望ましい状態を伝えます。デスティネーション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でマウント可能ですが、現在ミラー関係は有効ではありません。
- 再確立：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、以前もそのミラー関係にありました。
 - デスティネーション ボリュームがソース ボリュームと関係があった場合、デスティネーション ボリュームの内容が上書きされるため、再確立された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関係していなかった場合、再確立された状態は失敗します。

計画外のフェイルオーバー中にセカンダリ **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`promoted`に更新します。

計画されたフェイルオーバー中にセカンダリ **PVC** を昇格する

計画されたフェイルオーバー（移行）中に、次の手順を実行してセカンダリ PVC を昇格します：

手順

1. プライマリ Kubernetes クラスターで、PVC のスナップショットを作成し、スナップショットが作成されるまで待機します。
2. プライマリKubernetesクラスター上で、内部の詳細を取得するためにTridentTridentSnapshotInfoCRを作成します。

例

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

- a. セカンダリ Kubernetes クラスターで、*TridentMirrorRelationship* CR の *spec.state* フィールドを *promoted* に、*spec.volumeMappings[].promotedSnapshotHandle* をスナップショットの *internalName* に更新します。
3. セカンダリKubernetesクラスターで、*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復元する

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クラスターで、*TridentMirrorRelationship*の *_spec.remoteVolumeHandle_* フィールド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クラスターで、*TridentMirrorRelationship*の *_spec.state_* フィールドを *`reestablished`* に更新します。

追加操作

Trident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操作をサポートします：

プライマリ**PVC**を新しいセカンダリ**PVC**にレプリケートする

プライマリ PVC とセカンダリ PVC が既に存在することを確認します。

手順

1. 確立されたセカンダリ（デスティネーション）クラスターから*PersistentVolumeClaim* と *TridentMirrorRelationship* CRDを削除します。
2. プライマリ（ソース）クラスターから *TridentMirrorRelationship* CRD を削除します。
3. 新しいセカンダリ（デスティネーション）PVC を確立するために、プライマリ（ソース）クラスター上に新しい *TridentMirrorRelationship* CRD を作成します。

ミラーリングされたプライマリまたはセカンダリ **PVC** のサイズを変更する

PVCは通常通りサイズ変更できます。ONTAPは、データの量が現在のサイズを超える場合、デスティネーションのflexvolを自動的に拡張します。

PVCからレプリケーションを削除する

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 上のMirrorRelationshipを削除します。これにより、レプリケーション関係が切断されます。
- または、spec.state フィールドを *promoted* に更新します。

PVC を削除する（以前にミラーリングされたもの）

Tridentはレプリケートされた PVC をチェックし、ボリュームの削除を試みる前にレプリケーション関係を解放します。

TMRを削除する

ミラー関係の片側でTMRを削除すると、残りのTMRは `_promoted_` 状態に移行してから、Tridentが削除を完了します。削除対象として選択されたTMRがすでに `_promoted_` 状態にある場合、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* に昇格します。この削除により、ONTAPのローカルボリュームのSnapMirrorメタデータが解放されます。このボリュームが将来ミラー関係で使用される場合は、新しいミラー関係を作成するときに、`_established_` ボリュームレプリケーション状態を持つ新しい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する

ミラー関係は、確立された後はいつでも更新できます。`state: promoted`または`state: reestablished`フィールドを使用して関係を更新できます。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、現在のボリュームをリストアする特定のSnapshotを指定できます。

ONTAPがオフラインのときにミラー関係を更新する

CRDを使用すると、TridentがONTAPクラスタに直接接続しなくてもSnapMirror更新を実行できます。TridentActionMirrorUpdateの次の例の形式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`は、TridentActionMirrorUpdate CRDの状態を反映しています。 *Succeeded*、*In Progress*、また

は `_Failed_` のいずれかの値を取ることができます。

CSI トポロジを使用する

Trident は、"[CSI Topology機能](#)"を使用して、Kubernetes クラスタ内のノードにボリュームを選択的に作成して接続することができます。

概要

CSI トポロジ機能を使用すると、リージョンと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 プロバイダーは Kubernetes 管理者がゾーン ベースのノードを生成できるようにしています。ノードは、リージョン内の異なるアベイラビリティゾーン、または複数のリージョンに配置できます。マルチゾーンアーキテクチャにおけるワークロード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 は CSI トポロジを使用します。



CSI トポロジ機能の詳細 "[ここをクリックしてください。](#)"

Kubernetes は、2 つの独自のボリューム バインディング モードを提供します：

- `'VolumeBindingMode'` を `'Immediate'` に設定すると、Trident はトポロジを意識せずにボリュームを作成します。ボリュームのバインディングと動的プロビジョニングは、PVC の作成時に処理されます。これはデフォルトの `'VolumeBindingMode'` であり、トポロジ制約を強制しないクラスターに適しています。永続ボリュームは、要求元のポッドのスケジュール要件に依存せずに作成されます。
- `'VolumeBindingMode'` を `'WaitForFirstConsumer'` に設定すると、PVC の永続ボリュームの作成とバインドは、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。このようにして、トポロジ要件によって適用されるスケジュール制約を満たす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 トポロジ機能とは独立して使用できます。

要件

CSI トポロジを利用するには、次のものがが必要です：

- "[サポートされている Kubernetes バージョン](#)"を実行しているKubernetesクラスタ

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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ー認識を導入するラベルが必要です (`topology.kubernetes.io/region`および`topology.kubernetes.io/zone`)。これらのラベルは、Tridentがトポロジを認識できるようにするために、Tridentをインストールする前に*クラスタ内のノードに存在している必要があります*。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},  
{.metadata.labels}]{ "\n"}{end}' | grep --color "topology.kubernetes.io"  
[node1,  
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/os":"linux","node-role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-east1","topology.kubernetes.io/zone":"us-east1-a"}]  
[node2,  
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/os":"linux","node-role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-east1","topology.kubernetes.io/zone":"us-east1-b"}]  
[node3,  
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/os":"linux","node-role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

ステップ1：トポロジー対応のバックエンドを作成する

Tridentストレージ バックエンドは、可用性ゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドには、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 `'supportedTopologies'` ブロックを含めることができます。このようなバックエンドを使用するStorageClassesの場合、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によって要求された場合にのみ作成されます。

バックエンドの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、バックエンドごとの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StorageClassesの場合、Tridentはバックエンドにボリュームを作成します。

`supportedTopologies`をストレージプールごとに定義することもできます。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`および`zone`ラベルはストレージ プールの場所を表します。
`topology.kubernetes.io/region`および`topology.kubernetes.io/zone`は、ストレージ プールをどこから使用できるかを指定します。

ステップ2：トポロジを認識するStorageClasses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、StorageClassesをトポロジ情報を含めるように定義できます。これにより、PVC要求の候補となるストレージプールと、Tridentによってプロビジョニングされたボリュームを利用できるノードのサブセットが決定されます。

次の例を参照してください。

```

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

```

```
fsType: ext4
```

上記のStorageClass定義では、volumeBindingModeが`WaitForFirstConsumer`に設定されています。このStorageClassで要求されたPVCは、ポッド内で参照されるまで処理されません。また、`allowedTopologies`は使用するゾーンとリージョンを提供します。`netapp-san-us-east1` StorageClassは、上記で定義された`san-backend-us-east1`バックエンドにPVCを作成します。

ステップ3：PVCを作成して使用する

StorageClassが作成され、バックエンドにマップされたら、PVCを作成できるようになります。

以下の例を参照してください spec：

```
---
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                                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: voll
  mountPath: /data/demo
securityContext:
  allowPrivilegeEscalation: false

```

このpodSpecは、Kubernetesに対して、`us-east1`リージョンに存在するノードにポッドをスケジュールし、`us-east1-a`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ように指示します。

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                                     CAPACITY
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`supportedTopologies`を含める

既存のバックエンドは、`supportedTopologies`のリストを含めるように`tridentctl backend update`を使用して更新できます。これはすでにプロビジョニングされたボリュームには影響せず、以降のPVCにのみ使用されます。

詳細情報の参照

- ["コンテナのリソースを管理する"](#)
- ["nodeSelector"](#)
- ["親和性と反親和性"](#)
- ["TaintsとTolerations"](#)

Tridentスナップショットの操作

Kubernetes の Persistent Volume （PV）のボリュームスナップショットにより、ポイントインタイムのボリュームコピーが可能になります。Trident を使用して作成されたボリュームのスナップショットを作成したり、Trident 外部で作成されたスナップショットをインポートしたり、既存のスナップショットから新しいボリュームを作成したり、スナップショットからボリュームデータをリカバリしたりできます。

概要

ボリュームスナップショットは `ontap-nas`、`ontap-nas-flexgroup`、`ontap-san`、`ontap-san-economy`、`solidfire-san`、`azure-netapp-files`、および `'google-cloud-netapp-volumes'` ドライバでサポートされています。

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ーとカスタムリソース定義 (CRD) が必要です。これは Kubernetes オークストレーター (例: Kubeadm、GKE、OpenShift) の責任です。

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ーと CRD が含まれていない場合は、[ボリューム Snapshot コントローラーを導入する](#)を参照してください。



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ーを作成しないでください。GKEは組み込みの隠しスナップショットコントローラーを使用します。

ボリューム Snapshot を作成する

手順

1. `'VolumeSnapshotClass'` を作成します。詳細については、["VolumeSnapshotClass"](#)を参照してください。
 - `'driver'` は Trident CSI ドライバを指しています。
 - `'deletionPolicy'` は `'Delete'` または `'Retain'` に設定できます。`'Retain'` に設定すると、`'VolumeSnapshot'` オブジェクトが削除された場合でも、ストレージクラスター上の基礎となる物理 Snapshot は保持されます。

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

- この例では、既存の PVC のスナップショットを作成します。

```
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、`pvc1`という名前のPVCのボリュームスナップショットオブジェクトを作成し、スナップショットの名前は`pvc1-snap`に設定されます。VolumeSnapshotはPVCに類似しており、実際のスナップショットを表す`VolumeSnapshotContent`オブジェクトに関連付けられています。

```

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

kubectl get volumesnapshots
NAME                                AGE
pvc1-snap                          50s

```

- 説明することで、VolumeSnapshotContent オブジェクトを pvc1-snap VolumeSnapshot用に特定できます。`Snapshot Content Name`は、このスナップショットに対応するVolumeSnapshotContent オブジェクトを識別します。`Ready To Use`パラメータは、この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を作成する

`dataSource`を使用して、VolumeSnapshotという名前の `` をデータのソースとして使用するPVCを作成できます。PVCが作成されると、ポッドに接続して他のPVCと同じように使用できるようになります。



PVC はソース ボリュームと同じバックエンドで作成されます。["KB : Trident PVC スナップショットからの PVC の作成は代替バックエンドでは実行できません"](#)を参照してください。

次の例では、`pvc1-snap`をデータソースとして使用してPVCを作成します。

```
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: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

ボリューム Snapshot をインポートする

Tridentは["Kubernetesの事前プロビジョニングされたスナップショットプロセス"](#)をサポートしており、クラスタ管理者が `VolumeSnapshotContent` オブジェクトを作成し、Trident外部で作成されたスナップショットをインポートできるようにします。

開始する前に

Tridentでスナップショットの親ボリュームを作成またはインポートしておく必要があります。

手順

1. *クラスタ管理者*: バックエンドスナップショットを参照する `VolumeSnapshotContent` オブジェクトを作成します。これにより、Tridentでスナップショットワークフローが開始されます。
 - `annotations` の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">` として指定します。

- ``<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>``を`snapshotHandle`で指定します。これは、外部スナップショットターが`ListSnapshots`コールでTridentに提供する唯一の情報です。



`<volumeSnapshotContentName>`は、CR の命名制約により、バックエンドのスナップショット名と常に一致するとは限りません。

例

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

```
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: `VolumeSnapshot`オブジェクトを参照するCRを作成します。
`VolumeSnapshotContent`これにより、指定されたネームスペースで`VolumeSnapshot`を使用するためのアクセスが要求されます。

例

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

```
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（操作は不要）：*外部スナップショットツールは新しく作成された `VolumeSnapshotContent` を認識し、`ListSnapshots` 呼び出しを実行します。Tridentは `TridentSnapshot` を作成します。
 - 外部スナップショットは、`VolumeSnapshotContent` を `readyToUse` に設定し、`VolumeSnapshot` を `true` に設定します。
 - Tridentが返します `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim` を作成して新しい `VolumeSnapshot` を参照します。ここで、`spec.dataSource`（または `spec.dataSourceRef`）名は `VolumeSnapshot` 名です。

例

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

```
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
```

スナップショットを使用してボリュームデータを回復する

`ontap-nas` および `ontap-nas-economy` ドライバー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に、スナップショットディレクトリはデフォルトでは非表示になっています。
`.snapshot` ディレクトリを有効にして、スナップショットから直接データをリカバリします。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

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



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

Snapshotからのインプレースボリュームリストア

Tridentは、TridentActionSnapshotRestore (TASR) CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

Tridentは、ontap-san、ontap-san-economy、ontap-nas、ontap-nas-flexgroup、azure-netapp-files、google-cloud-netapp-volumes、および`solidfire-san`ドライバでのスナップショットのリストアをサポートしています。

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 スナップショットが必要です。

- 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 スナップショットが使用できる状態であ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 `pvc1`とボリュームSnapshot `pvc1-snapshot`のCRを作成します。



TASR CR は、PVC と 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復元するには CR を適用します。この例では snapshot `pvc1`から復元します。

```
kubectl create -f tasr-pvcl-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復元します。スナップショットの復元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: pvcl
    volumeSnapshotName: pvcl-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、Tridentは失敗した場合に操作を自動的に再試行しません。再度操作を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 ユーザーには、アプリケーション名前空間で TASR CR を作成するために、管理者からの権限付与が必要になる場合があります。

関連するスナップショットを含む PV を削除する

関連するスナップショットを持つ永続ボリュームを削除すると、対応するTridentボリュームは「削除中」状態に更新されます。Tridentボリュームを削除するには、ボリュームスナップショットを削除します。

ボリュームSnapshotコントローラを導入する

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、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

```
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. Snapshot Controller 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 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` を開き、`namespace` をネームスペースに更新します。

関連リンク

- ["ボリュームスナップショット"](#)
- ["VolumeSnapshotClass"](#)

Tridentボリュームグループのスナップショットの操作

永続ボリューム（PV）の Kubernetes ボリューム グループ スナップショット NetApp Trident は、複数のボリュームのスナップショット（ボリューム スナップショットのグループ）を作成する機能を提供します。このボリューム グループ スナップショットは、同じポイントインタイムで作成された複数のボリュームからのコピーを表します。



VolumeGroupSnapshotは、ベータ API を備えた Kubernetes のベータ機能です。Kubernetes 1.32は、VolumeGroupSnapshotに必要な最小バージョンです。

ボリュームグループのSnapshotを作成する

ボリュームグループスナップショットは、次のストレージドライバでサポートされます：

- `ontap-san`ドライバ - iSCSI および FC プロトコルのみで、NVMe/TCP プロトコルには使用できません。
- ontap-san-economy - iSCSI プロトコルのみ。
- ontap-nas



ボリュームグループのスナップショットは、NetApp ASA r2 または AFX ストレージシステムではサポートされていません。

開始する前に

- Kubernetes のバージョンが K8s 1.32 以上であることを確認してください。
- 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ーとカスタムリソース定義（CRD）が必要です。これは Kubernetes オークストレーター（例：Kubeadm、GKE、OpenShift）の責任です。

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



GKE 環境でオンデマンドボリュームグループ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ーを作成しないでください。GKEは組み込みの隠しスナップショットコントローラーを使用します。

- スナップショットコントローラーYAMLで、`CSIVolumeGroupSnapshot`機能ゲートを`true`に設定して、ボリュームグループのスナップショットが有効になっていることを確認します。
- ボリュームグループスナップショットを作成する前に、必要なボリュームグループスナップショットクラスを作成します。
- すべてのPVC/ボリュームが同じSVM上にあることを確認して、VolumeGroupSnapshotを作成できるようにします。

手順

- VolumeGroupSnapshot を作成する前に、VolumeGroupSnapshotClass を作成してください。詳細については、["VolumeGroupSnapshotClass"](#)を参照してください。

```

apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete

```

- 既存のストレージクラスを使用して必要なラベルを持つ PVC を作成するか、これらのラベルを既存の PVC に追加します。

次の例では、`pvc1-group-snap`をデータソースとして、`consistentGroupSnapshot: groupA`をラベルとして使用してPVCを作成します。要件に応じてラベルのキーと値を定義します。

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1

```

- PVC で指定されたラベル ((consistentGroupSnapshot: groupA) と同じラベルを持つ VolumeGroupSnapshot を作成します。

この例では、ボリュームグループのスナップショットを作成します：

```

apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:

```

```
consistentGroupSnapshot: groupA
```

グループSnapshotを使用してボリュームデータをリカバリする

ボリューム グループ スナップショットの一部として作成された個々のスナップショットを使用して、個々の永続ボリュームをリストアできます。ボリューム グループ スナップショットをユニットとしてリカバリすることはできません。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

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



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

Snapshotからのインプレースボリュームリストア

Tridentは、TridentActionSnapshotRestore (TASR) CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

詳細については、"[Snapshotからのインプレースボリュームリストア](#)"を参照してください。

関連するグループスナップショットを含む PV を削除する

グループボリュームのスナップショットを削除する場合：

- VolumeGroupSnapshotsは、グループ内の個々のスナップショットではなく、グループ全体として削除できます。
- PersistentVolumesにスナップショットが存在する状態で削除された場合、Tridentはそのボリュームを「削除中」状態に移動します。これは、ボリュームを安全に削除する前にスナップショットを削除する必要があるためです。
- グループ化されたスナップショットを使用してクローンが作成され、その後グループを削除する場合、split-on-clone処理が開始され、分割が完了するまでグループを削除することはできません。

ボリュームSnapshotコントローラを導入する

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、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

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

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.yaml
```

2. Snapshot Controller を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```



必要に応じて `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` を開き、`namespace` をネームスペースに更新します。

関連リンク

- ["VolumeGroupSnapshotClass"](#)
- ["ボリュームスナップショット"](#)

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随時、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 5252.227-19（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权を有し、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权については、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。