ボリュームのプロビジョニングと管理
Trident

NetApp
March 05, 2026

目次

ボリュームのプロビジョニングと管理

ボリュームをプロビジョニングする

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

概要

あ "永続ボリュームクレーム"(PVC) は、クラスター上の 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。関連付けられた StorageClass を使用すると、クラスター管理者は PersistentVolume のサイズやアクセス モードだけでなく、パフォーマンスやサービス レベルなどの制御も行えます。

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

```
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
```

> 進捗状況は以下で確認できます。 `kubectl get pod --watch` 。

2. ボリュームがマウントされていることを確認する `/my/mount/path`。

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

3. これでポッドを削除できます。 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、StorageClassに関連付けられています。 `basic-csi` 。

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

NVMe/TCP 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を示しており、これはStorageClassに関連付けられています。 `protection-gold` 。

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

ポッド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構成を示しています。

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
```

参照"KubernetesとTridentオブジェクト"ストレージクラスがどのように相互作用するかの詳細については、`PersistentVolumeClaim` 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ためのパラメー

タ。

ボリュームを拡張する

Trident は、Kubernetes ユーザーにボリュームを作成後に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`ドライバーと 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**を構成する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドへ `true`。

```
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、 `allowVolumeExpansion`パラメータ。

ステップ**2:** 作成した**StorageClass**で**PVC**を作成する

PVC定義を編集して更新します `spec.resources.requests.storage`新しく希望するサイズを反映するため、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
```

ステップ**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 バックエンド上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と、 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` 。

```
 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` 2Giまで。

```
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かどうかを確認できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

FCボリュームを拡張する

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`ドライバーであり、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**を構成する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドへ `true`。

```
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、 `allowVolumeExpansion`パラメータ。

ステップ**2:** 作成した**StorageClass**で**PVC**を作成する

PVC定義を編集して更新します `spec.resources.requests.storage`新しく希望するサイズを反映するため、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
```

ステップ**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 バックエンド上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と、 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` 。

```
 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` 2Giまで。

```
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かどうかを確認できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

NFSボリュームを拡張する

Tridentは、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ます。 `ontap-nas` 、 `ontap-nas-economy` 、 `ontap-nas-flexgroup` 、 `gcp-cvs` 、 そして `azure-netapp-files`バックエンド。

ステップ**1:** ボリューム拡張をサポートするように**StorageClass**を構成する

NFS PVのサイズを変更するには、管理者はまず、ボリューム拡張を可能にするためにストレージクラスを設定する必要があります。 `allowVolumeExpansion`フィールドへ `true`:

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

```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、既存のストレージクラスを編集するだけで済みます。 `kubectl edit storageclass`ボリュームの拡張を可能にします。

ステップ**2:** 作成した**StorageClass**で**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を編集して設定します。
`spec.resources.requests.storage` 1 GiBまで:

```
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行われたかどうかを確認できます。

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

輸入量

`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

ボリュームをTridentにインポートすると、次のような効果が得られます。

- 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クラスターを再構築する
- 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 はRW (読み取り/書き込み) タイプのONTAPボリュームのみをインポートできます。 DP (データ保護) タイプのボリュームは、 SnapMirror の宛先ボリュームです。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。これによりデータが破損する可能性があります。

- けれど `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利用可能なプールから選択するために使用されます。ボリュームは既に存在するため、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 ドライバーによってインポートされた後、PVC への ClaimRef を使用して PV が作成されます。
  - 回収ポリシーは初期設定では `retain`PVでは。 Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーはストレージクラスの再利用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回収ポリシーが `delete`PV が削除されると、ストレージ ボリュームも削除されます。
- デフォルトでは、 Trident はPVC を管理し、バックエンドのFlexVol volumeと LUN の名前を変更します。あなたは合格することができます `--no-manage`管理されていないボリュームをインポートするためのフラグ。使用する場合 `--no-manage`Trident は、オブジェクトのライフサイクル中に PVC または PV に対して追加の操作を実行しません。 PV が削除されてもストレージ ボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などの他の操作も無視されます。

このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、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. 使用 `tridentctl import`ボリュームを含むTridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前 (例: ONTAP FlexVol、Element Volume、 Cloud Volumes Serviceパス) を指定するコマンド。その `-f`PVC ファイルへのパスを指定するには引数が必要です。

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

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については、次のボリューム インポート例を確認してください。

ONTAP NAS およびONTAP NAS FlexGroup

Tridentはボリュームインポートをサポートしており、 `ontap-nas`そして `ontap-nas-flexgroup`ドライバー。

- Tridentは、 `ontap-nas-economy` ドライバ。
- その `ontap-nas`そして `ontap-nas-flexgroup`ドライバーは重複したボリューム名を許可しません。

各ボリュームは、 `ontap-nas`ドライバーは、 `ONTAP`クラスタ上の`FlexVol volume`です。 `FlexVol`ボリュームをインポートするには `ontap-nas`ドライバーは同じように動作します。 `ONTAP`クラスタにすでに存在する`FlexVol`ボリュームは、 `ontap-nas` PVC。同様に、 FlexGroupボリュームは次のようにインポートできます。 `ontap-nas-flexgroup` PVC。

tridentctl を使用した ONTAP NAS の例

以下に、管理対象ボリュームと管理対象外ボリュームのインポートの例を示します。

管理ボリューム

次の例では、 `managed_volume`バックエンドで `ontap_nas`:

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

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

管理されていないボリューム

使用する際は `--no-manage`引数が指定されていない場合、 Trident はボリュームの名前を変更しません。

次の例では、 `unmanaged_volume`上の `ontap_nas`バックエンド:

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

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

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: <umanaged-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>
```

ONTAP SAN

Tridentはボリュームインポートをサポートしており、 `ontap-san` (iSCSI、NVMe/TCP、FC)および `ontap-san-economy` ドライバー。

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` ドライバは、各 PVC に対してFlexVol volumeを作成し、 FlexVol volume内に LUN を作成します。TridentはFlexVol volumeをインポートし、それを PVC 定義に関連付けます。Tridentは輸入できる `ontap-san-economy` 複数の LUN を含むボリューム。

ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリューム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 volumeの名前を `pvc-<uuid>`FlexVol volume内のフォーマットとLUNを `lun0`。

次の例では、 `ontap-san-managed` FlexVol volumeは、 `ontap_san_default`バックエンド:

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

+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             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

次の例では、 `unmanaged_example_volume`上の `ontap_san`バックエンド:

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             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
```

要素

TridentはNetApp Elementソフトウェアと NetApp HCIボリュームのインポートをサポートしています。 `solidfire-san`ドライバ。

> Element ドライバーは重複したボリューム名をサポートします。ただし、ボリューム名が重複している場合、 Trident はエラーを返します。回避策として、ボリューム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。

次の例では、 `element-managed`バックエンドのボリューム `element_default。

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

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

Google Cloud Platform

Tridentはボリュームインポートをサポートしており、 `gcp-cvs`ドライバ。

> Google Cloud Platform でNetApp Cloud Volumes Serviceによってバックアップされた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、ボリュームのエクスポートパスの `:/`。たとえば、エクスポートパスが `10.0.0.1:/adroit-jolly-swift`ボリュームパスは `adroit-jolly-swift`。

Google Cloud Platform の例

次の例では、 `gcp-cvs`バックエンドのボリューム `gcpcvs_YEppr`ボリュームパスの `adroit-jolly-swift。

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

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

Azure NetApp Files

Tridentはボリュームインポートをサポートしており、 `azure-netapp-files`ドライバ。

> 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、ボリュームのエクスポートパスの `:/`。たとえば、マウントパスが `10.0.0.2:/importvol1`ボリュームパスは `importvol1。

次の例では、 `azure-netapp-files`バックエンドのボリューム `azurenetappfiles_40517`ボリュームパス `importvol1。

```
tridentctl import volume azurenetappfiles_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はボリュームインポートをサポートしており、 `google-cloud-netapp-volumes`ドライバ。

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、 `google-cloud-netapp-volumes`同じリージョンに 2 つのボリュームが存在する場合のボリューム:

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

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

ボリューム名とラベルをカスタマイズする

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:

1. ボリュームの作成、インポート、およびクローン操作。
2. ontap-nas-economy ドライバーの場合、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。
3. ontap-san-economy ドライバーの場合、LUN 名のみが名前テンプレートに準拠します。

制限事項

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

カスタマイズ可能なボリューム名の主な動作

1. 名前テンプレートの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が失敗した場合、ボリュームは既存の命名規則に従って名前が付けられます。
2. バックエンド構成の名前テンプレートを使用してボリューム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.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

プールレベルの例

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

名前テンプレートの例

例**1**:

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

例**2**:

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

考慮すべき点

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。例えば：`{"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 プレフィックス付き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、 Trident はランダムな文字をいくつか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の長さが 64 文字を超える場合、 Trident は既存の命名規則に従ってボリュームに名前を付けます。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は失敗します。

名前空間間で**NFS**ボリュームを共有する

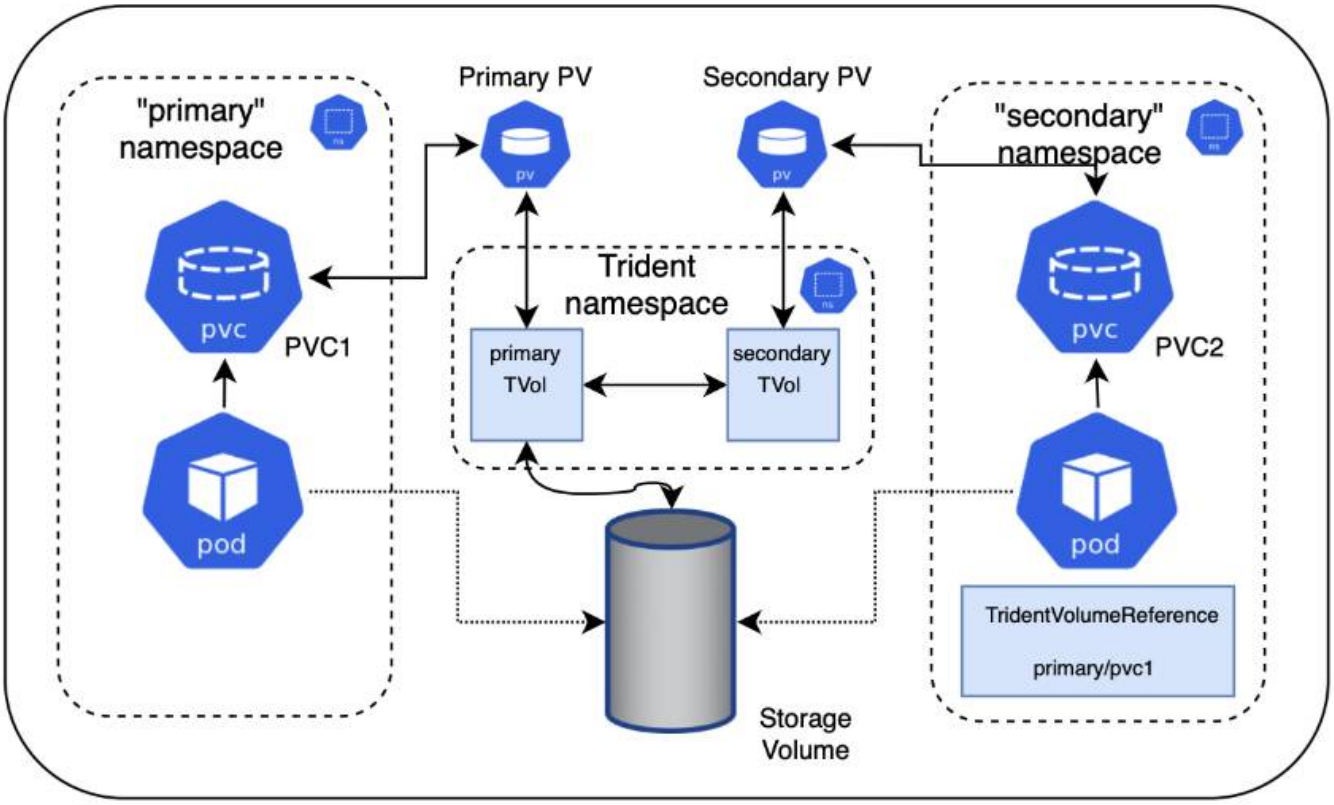
Trident を使用すると、プライマリ名前空間にボリュームを作成し、それを 1 つ以上のセカンダリ名前空間で共有できます。

機能

TridentVolumeReference CR を使用すると、1 つ以上の Kubernetes 名前空間間で ReadWriteMany (RWX) NFS ボリュームを安全に共有できます。この Kubernetes ネイティブ 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 ドライバーで動作します
- 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**を構成する

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 宛先名前空間に **CR** を作成する権限を付与する

クラスター管理者は、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 を参照するための TridentVolumeReference CR を作成します。

4 宛先名前空間に従属**PVC**を作成する

宛先名前空間の所有者は、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。

ソースと宛先の名前空間を構成する

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. ソース名前空間の所有者: PVCを作成する(`pvc1`）をソース名前空間に作成し、宛先名前空間と共有する権限を付与する(`namespace2`）を使用して `shareToNamespace`注釈。

```
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。例えば、`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4` 。
- すべての名前空間に共有するには、 `*` 。例えば、`trident.netapp.io/shareToNamespace: *`
- PVCを更新して、 `shareToNamespace`いつでも注釈を付けることができます。

2. クラスター管理者: 宛先名前空間の所有者に宛先名前空間に TridentVolumeReference CR を作成するための権限を付与するための適切な RBAC が設定されていることを確認します。
3. 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
```

4. 宛先名前空間の所有者: PVCを作成する(`pvc2`）を宛先名前空間に(`namespace2`）を使用して `shareFromPVC`送信元 PVC を指定するための注釈。

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

宛先 PVC のサイズは、送信元 PVC のサイズ以下である必要があります。

結果

Tridentは `shareFromPVC`宛先 PVC にアノテーションを追加し、宛先 PV を、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、独自のストレージ リソースを持たない従属ボリュームとして作成します。宛先 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、ソース名前空間上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。

使用 `tridentctl get`従属ボリュームを照会する

使用して[`tridentctl`ユーティリティを実行すると、 `get`従属ボリュームを取得するコマンド。詳細については、次のリンクを参照してください:../trident-reference/tridentctl.html[`tridentctl`コマンドとオプション]。

```
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、宛先名前空間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へのアクセスは、 `shareToNamespace`または `shareFromNamespace`注釈や削除 `TridentVolumeReference`CR。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、以下を参照してください。

- 訪問"名前空間間でボリュームを共有する: クロスネームスペースボリュームアクセスの実現"。
- デモを見る"ネットアップTV"。

名前空間を越えてボリュームを複製する

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`そして `ontap-nas`ストレージ ドライバー。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. ボリュームを複製するためのソース**PVC**を構成する

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. 宛先名前空間に **CR** を作成する権限を付与する

クラスター管理者は、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 を参照するための 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 を複製するための PVC を作成します。

ソースと宛先の名前空間を構成する

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: PVCを作成する(`pvc1`）ソース名前空間(`namespace1`）は、宛先名前空間と共有する権限を付与します(`namespace2`）を使用して `cloneToNamespace`注釈。

```
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。例えば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4` 。
- すべての名前空間に共有するには、 `*` 。例えば、`trident.netapp.io/cloneToNamespace: *`
- PVCを更新して、 `cloneToNamespace`いつでも注釈を付けることができます。

2. クラスタ管理者: 宛先名前空間の所有者に、宛先名前空間に TridentVolumeReference CR を作成する権限を付与するための適切な RBAC が設定されていることを確認します。(`namespace2` ）。
3. 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
```

4. 宛先名前空間の所有者: PVCを作成する(`pvc2`）を宛先名前空間に(`namespace2`）を使用して `cloneFromPVC`または `cloneFromSnapshot`、 そして `cloneFromNamespace`ソース PVC を指定するための注釈。

```
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を使用してボリュームを複製する

Trident は、災害復旧のためにデータを複製するために、1 つのクラスター上のソース ボリュームとピア クラスター上の宛先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

- 鏡の関係を破る
- 災害発生時にセカンダリボリュームを昇格する（フェイルオーバー）
- 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使用するソース Kubernetes クラスターと宛先 Kubernetes クラスターの両方に、 Tridentバージョン 22.10 以降が存在している必要があります。
- ライセンス: データ保護バンドルを使用するONTAP SnapMirror非同期ライセンスは、ソースと宛先の両方のONTAPクラスタで有効にする必要があります。参照 "ONTAPにおけるSnapMirrorライセンスの概要"詳細についてはこちらをご覧ください。

  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。参照"ONTAP Oneに含まれるライセンス"詳細についてはこちらをご覧ください。

  SnapMirror非同期保護のみがサポートされます。

ピアリング

- クラスタと **SVM**: ONTAPストレージ バックエンドをピアリングする必要があります。参照 "クラスタとSVMのピアリングの概要"詳細についてはこちらをご覧ください。

  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ます。

- * Tridentと SVM*: ピア接続されたリモート SVM は、宛先クラスター上の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ー

NetApp Trident は、次のドライバーでサポートされるストレージ クラスを使用して、 NetApp SnapMirrorテクノロジによるボリューム 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 (最低でも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、以下を参照してください。"ASA r2 ストレージシステムについて学ぶ" 。

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 クラスターで次の手順を実行します。
   a. StorageClassオブジェクトを作成し、 `trident.netapp.io/replication: true`パラメータ。

      例

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

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

      例

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

   c. ローカル情報を使用して 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 のステータス 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 クラスターで次の手順を実行します。

a. trident.netapp.io/replication: true パラメータを使用して StorageClass を作成します。

例

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

b. 宛先とソースの情報を含む MirrorRelationship CR を作成します。

例

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

Trident は、設定された関係ポリシー名 (またはONTAPのデフォルト) を使用してSnapMirror関係を作成し、初期化します。

c. 以前に作成した StorageClass を使用して、セカンダリ (SnapMirror の宛先) として機能する PVC を作成します。

例

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

Trident はTridentMirrorRelationship CRD をチェックし、関係が存在しない場合はボリュームの作成に失敗します。関係が存在する場合、 Trident は、新しいFlexVol volumeが、MirrorRelationship で定義されたリモート SVM とピアリングされている SVM 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 Mirror Relationship (TMR) は、PVC 間のレプリケーション関係の一方の端を表す CRD です。宛先 TMR には状態があり、これによってTrident に目的の状態が伝えられます。宛先 TMR の状態は次のとおりです。

- 確立済み: ローカル PVC はミラー関係の宛先ボリュームであり、これは新しい関係です。

- 昇格: ローカル PVC は読み取り/書き込み可能でマウント可能であり、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 はミラー関係の宛先ボリュームであり、以前もそのミラー関係にありました。
  - 宛先ボリュームがソース ボリュームと関係があった場合、宛先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 クラスターで次の手順を実行します。

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

計画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 クラスターで、内部詳細を取得するための SnapshotInfo CR を作成します。

   例

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

3. セカンダリ Kubernetes クラスターで、*TridentMirrorRelationship* CR の *spec.state* フィールドを *promoted* に更新し、*spec.promotedSnapshotHandle* をスナップショットの internalName に更新します。
4. 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) が昇格され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

ミラー関係を復元する前に、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クラスタで、TridentMirrorRelationshipの_spec.mirror_フィールドを次のように更新します。 `reestablished` 。

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**に複製する

プライマリ PVC とセカンダリ PVC がすで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 (宛先) クラスターから PersistentVolumeClaim および 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. 確立する新しいセカンダリ (宛先) PVC のプライマリ (ソース) クラスターに新しい 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 は通常どおりサイズを変更できますが、データの量が現在のサイズを超えると、 ONTAP は宛先 flevxol 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 ボリュームに対して次のいずれかの操作を実行します。

- セカンダリ PVC の MirrorRelationship を削除します。これにより、レプリケーション関係が切断されます。
- 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

ミラー関係の一方の TMR を削除すると、 Trident が削除を完了する前に、残りの TMR が *promoted* 状態に移行します。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、ボリュームのレプリケーション状態が確立された新しい 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。使用することができます `state: promoted`または `state: reestablished`関係を更新するためのフィールド。宛先ボリュームを通常の ReadWrite ボリュームに昇格する場合、*promotedSnapshotHandle* を使用して、現在のボリュームを復元する特定のスナップショットを指定できます。

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されていなくても、 SnapMirror の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`TridentActionMirrorUpdate CRD の状態を反映します。 *Succeeded*、*In Progress*、*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、 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。 "CSIトポロジ機能" 。

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ーゾーンに基づいて、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ーゾーン、または複数のリージョンに配置できます。マルチゾーン アーキテクチャでの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トポロジー機能の詳細 "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。

- と `VolumeBindingMode`に設定 `Immediate`Trident はトポロジを考慮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。これはデフォルトです `VolumeBindingMode`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。
- と `VolumeBindingMode`に設定 `WaitForFirstConsumer`、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。

その `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、次のものが必要です。

- Kubernetesクラスタは"サポートされているKubernetesバージョン"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
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:"1e11e4a2108024935ecfcb2912226cedeafd99df",
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`そして `topology.kubernetes.io/zone`）。 Tridentがトポロジを認識するためには、 Tridentがインストールされる前に、これらのラベルが クラスター内のノード上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","kube
rnetes.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","kube
rnetes.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","kube
rnetes.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。各バックエンドはオプションで `supportedTopologies`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ブロック。このようなバックエンドを利用する 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。

ヤムル

```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

```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`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 で提供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。

定義できます `supportedTopologies`ストレージ プールごとにも同様です。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`そして `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`そして `topology.kubernetes.io/zone`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: トポロジーを考慮したストレージクラスを定義する

クラスター内のノードに提供されるトポロジ ラベルに基づいて、トポロジ情報を格納するように StorageClasses を定義できます。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`設定されている `WaitForFirstConsumer`。この StorageClass で要求された PVC は、ポッドで参照されるまで処理されません。そして、 `allowedTopologies`使用するゾーンとリージョンを提供します。その `netapp-san-us-east1`StorageClassはPVCを `san-backend-us-east1`上記で定義されたバックエンド。

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ので、PVC を作成できるようになりました。

例を見る `spec`下に：

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

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

このpodSpecはKubernetesに、 `us-east1`地域を選択し、その地域にある任意のノードから選択します。 `us-east1-a`または `us-east1-b`ゾーン。

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含める `supportedTopologies`

既存のバックエンドを更新して、 `supportedTopologies`使用して `tridentctl backend update`。これは、すでにプロビジョニングされているボリュームには影響せず、後続の 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"
- "ノードセレクタ"
- "親和性と反親和性"
- "汚名と寛容"

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 スナップショットにより、ボリュームのポイントインタイム 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の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 データを回復したりできます。

概要

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

開始する前に

スナップショットを操作するには、外部スナップショット コントローラーとカスタム リソース定義 (CRD) が必要です。これは、Kubernetes オーケストレーター (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とcrdが含まれていない場合は、[ボリュームスナップショットコントローラを展開する] 。

GKE 環境でオンデマンド ボリューム スナップショットを作成する場合は、スナップショット コントローラを作成しないでください。 GKE は組み込みの隠しスナップショット コントローラを使用します。

ボリュームスナップショットを作成する

手順

1. 作成する `VolumeSnapshotClass`詳細については、"ボリュームスナップショットクラス" 。
   - その `driver`Trident CSI ドライバーを指します。
   - `deletionPolicy`できる `Delete`または `Retain`。に設定すると `Retain`ストレージクラスター上の基礎となる物理スナップショットは、 `VolumeSnapshot`オブジェクトは削除されます。

     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、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ます。 `pvc1`スナップショットの名前は `pvc1-snap`。 VolumeSnapshotはPVCに類似しており、 `VolumeSnapshotContent`実際のスナップショットを表すオブジェクト。

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

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

- あなたは、 `VolumeSnapshotContent`のオブジェクト `pvc1-snap`VolumeSnapshot を記述します。その `Snapshot Content Name`このスナップショットを提供する VolumeSnapshotContent オブジェクトを識別します。その `Ready To Use`パラメーターは、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
...
```

ボリュームスナップショットからPVCを作成する

使用できます `dataSource`VolumeSnapshotを使用してPVCを作成します `<pvc-name>`データのソースとして。 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に作成されます。参照"KB: Trident PVC スナップショットから PVC を作成すると、代替バックエンドでは作成できません"。

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

```
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
```

ボリュームスナップショットをインポートする

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"クラスタ管理者が `VolumeSnapshotContent`オブジェクトを作成し、 Tridentの外部で作成されたスナップショットをインポートします。

開始する前に

Trident は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いる必要があります。

手順

1. クラスタ管理者: `VolumeSnapshotContent`バックエンドスナップショットを参照するオブジェクト。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。
   - バックエンドスナップショットの名前を指定します `annotations`として `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 特定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`で `snapshotHandle`これは、外部スナップショット装置からTridentに提供される唯一の情報です。 `ListSnapshots`電話。

     その `<volumeSnapshotContentName>`CR 命名制約により、バックエンド 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、 `VolumeSnapshotContent`バックエンドスナップショットを参照するオブジェクト `snap-01`。

```
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` CRは、 `VolumeSnapshotContent`物体。これは、 `VolumeSnapshot`特定の名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap`を参照する `VolumeSnapshotContent`名前 `import-snap-content`。

```
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`そして、 `ListSnapshots`電話。Tridentは `TridentSnapshot`。
   - 外部スナップショットは、 `VolumeSnapshotContent`に `readyToUse`そして `VolumeSnapshot`に `true`。
   - Tridentが復活 `readyToUse=true`。
4. 任意のユーザー: 作成 `PersistentVolumeClaim`新しいものを参照する `VolumeSnapshot`、ここで `spec.dataSource`（または `spec.dataSourceRef`）の名前は `VolumeSnapshot`名前。

   例

次の例では、次のものを参照するPVCを作成します。 `VolumeSnapshot`名前 `import-snap`。

```
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

スナップショットディレクトリは、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、デフォルトでは非表示になっています。 `ontap-nas`そして `ontap-nas-economy`ドライバー。有効にする `.snapshot`スナップショットからデータを直接回復するためのディレクトリ。

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
```

スナップショット コピーを復元すると、既存のボリューム構成が上書きされます。スナップショット 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。

スナップショットからのインプレースボリューム復元

Tridentは、スナップショットからボリュームを迅速に復元する機能を提供します。 `TridentActionSnapshotRestore` （TASR）CR。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、スナップショットの復元をサポートしています。 `ontap-san` 、 `ontap-san-economy` 、 `ontap-nas` 、 `ontap-nas-flexgroup` 、 `azure-netapp-files` 、 `gcp-cvs` 、 `google-cloud-netapp-volumes` 、 そして `solidfire-san`ドライバー。

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のCRを作成します `pvc1`ボリュームスナップショット `pvc1-snapshot`。

   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。この例ではスナップショットから復元します `pvc1`。

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

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

結果

Trident はスナップショットからデータを復元します。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: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- ほとんどの場合、失敗した場合にTrident は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ボリュームが「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。

ボリュームスナップショットコントローラを展開する

Kubernetes ディストリビューションにスナップショット コントローラーと 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.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. スナップショット コントローラーを作成します。

```
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
```

必要に応じて開く `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`更新 `namespace`あなたの名前空間に。

関連リンク

- "ボリュームスナップショット"
- "ボリュームスナップショットクラス"

ボリュームグループのスナップショットを操作する

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 グループのスナップショットは、同じ時点で取得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。VolumeGroupSnapshot に必要な最小バージョンは Kubernetes 1.32 です。

ボリュームグループのスナップショットを作成する

ボリュームグループスナップショットは、 `ontap-san`ドライバーは iSCSI プロトコル専用であり、ファイバー チャネル (FCP) や NVMe/TCP ではまだサポートされていません。始める前に

- 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 オーケストレーター (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とCRDが含まれていない場合は、[ボリュームスナップショットコントローラを展開する] 。

  > 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 コントローラを使用します。

- スナップショットコントローラYAMLで、 `CSIVolumeGroupSnapshot`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するには、機能ゲートを 'true' に設定します。
- 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。
- VolumeGroupSnapshot を作成できるようにするには、すべての PVC/ボリュームが同じ SVM 上にあることを確認します。

手順

- VolumeGroupSnapshot を作成する前に、VolumeGroupSnapshotClass を作成します。詳細については、"ボリュームグループスナップショットクラス" 。

```
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、PVCを作成します。 `pvc1-group-snap`データソースとラベルとして `consistentGroupSnapshot: groupA`。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
```

- 同じラベルのVolumeGroupSnapshotを作成する(`consistentGroupSnapshot: groupA`) を PVC で指定します。

  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

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復元できます。ボリューム グループ スナップショットを単位として復元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
```

スナップショット コピーを復元すると、既存のボリューム構成が上書きされます。スナップショット 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。

スナップショットからのインプレースボリューム復元

Tridentは、スナップショットからボリュームを迅速に復元する機能を提供します。`TridentActionSnapshotRestore` （TASR）CR。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、 "スナップショットからのインプレースボリューム復元"。

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:

- グループ内の個々のスナップショットではなく、VolumeGroupSnapshots 全体を削除できます。
- PersistentVolume のスナップショットが存在している間に PersistentVolume が削除された場合、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、 Trident はそのボリュームを「削除中」状態に移行します。
- 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を作成し、その後グループを削除する場合は、クローン時に分割操作が開始され、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。

ボリュームスナップショットコントローラを展開する

Kubernetes ディストリビューションにスナップショット コントローラーと 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_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.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. スナップショット コントローラーを作成します。

```
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
```

必要に応じて開く `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`更新 `namespace`あなたの名前空間に。

関連リンク

- "ボリュームグループスナップショットクラス"
- "ボリュームスナップショット"

著作権に関する情報

商標に関する情報