ボリュームのプロビジョニングと管理
Trident

NetApp
March 02, 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

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

概要

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

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

PVCを作成したら、ボリュームをポッドにマウントできます。

PVCの作成

手順

1. PVC を作成します。

```
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
```

> 進捗状況は次を使用して監視できます。 `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します。 `/my/mount/path`。

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

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

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

マニフェストの例

PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えたPVC

次の例は、という名前のStorageClassに関連付けられた、RWOアクセスが設定された基本的なPVCを示しています。 `basic-csi`。

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

NVMe / TCP対応PVC

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

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

PODマニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

キホンセツテイ

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

NVMe/TCPの基本構成

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

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

オブジェクト"。

ボリュームを展開します

Trident 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 ボリュームを展開します

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

iSCSI ボリュームの拡張は 'ONTAP-SAN"ONTAP-SAN-エコノミー "olidfire-SAN' ドライバによってサポートされており 'Kubernetes 1.16 以降が必要です

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `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 パラメータを含めるように編集します

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

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、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は永続的ボリューム（PV）を作成し、この永続的ボリューム要求（PVC）に関連付けます。

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

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

手順 **3** ： **PVC** を接続するポッドを定義します

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

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

この例では、ポッドが作成され、「 1-pvc 」が使用されます。

```
 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** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.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
 # ...
```

手順 **5** ：拡張を検証する

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

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

FC ボリュームを拡張します

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

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

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `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 パラメータを含めるように編集します

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

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、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は永続的ボリューム（PV）を作成し、この永続的ボリューム要求（PVC）に関連付けます。

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

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

手順 **3** ： **PVC** を接続するポッドを定義します

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

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

この例では、ポッドが作成され、「 1-pvc 」が使用されます。

```
 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** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.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
 # ...
```

手順 **5** ：拡張を検証する

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

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

NFS ボリュームを拡張します

Trident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` 、 そして `azure-netapp-files`バックエンド。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PV のサイズを変更するには ' まず 'allowVolumeExpansion フィールドを true に設定してボリュームを拡張できるようにストレージ・クラスを構成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
```

このオプションを指定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・クラスを編集するだけで ' ボリュームの拡張が可能になります

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

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

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

Trident はこの PVC 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** を展開します

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
# ...
```

手順 **4** ：拡張を検証する

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

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

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       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

NVMe プロトコルを使用する ReadWriteMany（RWX）ボリュームには、ボリュームあたり 64 ノードのスケーラビリティ制限があります。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ノードの制限について理解する

NVMe プロトコルで ReadWriteMany（RWX）ボリュームを使用する場合、単一の RWX NVMe ボリュームを Kubernetes クラスター内の 64 を超えるノード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を超えるノードにわたっ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制限は低くなります。大規模な Kubernetes クラスタでは、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 個の名前空間（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 個のノードをサポートします。

エラーの症状を特定する

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ことがあります：

```
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の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。

Tridentをアップグレードしてスーパーサブシステムモデルを適用する

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をゼロレプリカにスケールダウンします。
3. RWX NVMe ボリュームをアクティブに使用しているワークロードがないことを確認します。
4. ポッドをスケールアップし直します。

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。
- RWX NVMe ボリュームごとに 64 ノードの制限が適用されます。

- 他のアクセス モードおよび他のプロトコルは影響を受けません。

コントローラのスケーラビリティ

Tridentは、複数のストレージドライバー間の同時実行性の向上により、コントローラーのスケーラビリティを実現します。お客様は、一般提供時にコントローラーのスケーラビリティをサポートするTridentドライバーと、Trident 26.02でテクニカルプレビューとして利用可能なドライバーを特定できます。これにより、スケーラブルなKubernetes環境において、情報に基づいた導入の決定と適切なリスク管理が可能になります。

主要な概念と定義

コントローラのスケーラビリティ

コントローラのスケーラビリティとは、Tridentコントローラが単一のロックでシリアル化するのではなく、複数のストレージ操作を並列に処理する能力を指します。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、バックエンドの管理が含まれます。

コントローラーのスケーラビリティが有効になっている場合、異なるボリュームとバックエンドでの操作が同時に進行します。これにより、スループットが向上し、同時実行されるPersistentVolumeClaim操作とVolumeSnapshot操作の数が多い環境でのエンドツーエンドの操作時間が短縮されます。

コントローラのスケーラビリティサポート

Tridentは、ストレージドライバに応じて異なる成熟度レベルのコントローラスケーラビリティをサポートします。

一般提供

以下のドライバは、Trident 26.02の一般提供時にコントローラのスケーラビリティをサポートします：

- `san`
- `nas`
- `san-nvme`
- `google-cloud-netapp-volumes`

コントローラのスケーラビリティを有効にする

コントローラのスケーラビリティは、 `enableConcurrency`構成オプションによって制御されます。このオプションは、Tridentのインストール時、または既存の導入環境を更新することによって明示的に有効にする必要があります。

Trident オペレータの導入

Tridentオペレーターでコントローラのスケーラビリティを有効にするには、 `enableConcurrency`を `true`に設定し、 `TridentOrchestrator`カスタムリソースで指定します。

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ーラビリティを有効にします：

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

設定が適用され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ー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 `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

`tridentctl`でコントローラのスケーラビリティを有効にするには、インストール時に `--enable-concurrency`フラグを渡します。

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`ベースの導入でコントローラのスケーラビリティを有効にするには、次のフラグを使用してアンインストールして再インストールします：

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

コントローラのスケーラビリティが有効になっていることを確認する

コントローラのスケーラビリティを有効にした後、Tridentコントローラ ポッドのログを確認して、コントローラが同時実行を有効にして実行されていることを確認します：

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

同時実行が有効になっていることを示すログ エントリが表示されます。

テクニカルプレビュー

以下のドライバーは、Trident 26.02で技術プレビューとしてコントローラーのスケーラビリティをサポートしています：

- `nas-eco`
- `san-eco`

これらのドライバーの場合：

- コントローラの同時実行は評価とテストに利用できます
- 今後のリリースでは動作が変更される場合があります
- 本番環境での使用は推奨されません

同時実行動作

コントローラーのスケーラビリティが有効な場合：

- Tridentは単一のグローバルロックを、きめ細かなリソースごとのロックに置き換えます。
- 同じリソースを変更する処理は、データの整合性を維持するためにシリアル化されます。
- リソースからの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同時に実行できます。
- 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に、管理 LIF ごとの同時 ONTAP API リクエストを 20 件に制限します
- 複数のバックエンドが同じ管理LIFを共有する場合、この20リクエストの制限も共有されます

既知の制限事項と考慮事項

コントローラのスケーラビリティには、次の考慮事項が適用されます。

- 同時実行はTridentコントローラによって内部的に管理されます
- このリリースでは、ユーザーが設定できる同時実行制限はありません
- 全体的なスループットは以下に依存します：
  - 使用中のストレージドライバー
  - バックエンドの応答性
  - Kubernetes API サーバーのパフォーマンス
- 同時実行性が高いと、バックエンドのストレージ システムの負荷が増加する可能性があります。

注意点と制限事項

Trident 26.02 には次の制限事項があります：

- コントローラのスケーラビリティ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 ドライバーでは次の症状が現れる場合があります：

- 高負荷時にパフォーマンスが不安定になる
- リリース間の動作の変更

- 並列実行のため、同時操作のデバッグはより複雑になる可能性があります
- 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

- 高いスケーラビリティが求められる本番環境では、一般提供（GA）ドライバを使用する
- 非本番環境でテクニカルプレビュードライバを評価する
- 大規模運用時のバックエンドとコントローラーのパフォーマンスを監視する
- 自動化スクリプトで操作の順序を想定しないようにする

ボリュームをインポート

`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

Tridentにボリュームをインポートする目的は次のとおりです。

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

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Tridentでインポートできるのは、RW（読み取り/書き込み）タイプのONTAPボリュームのみです。DP（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをTridentにインポートする前に、ミラー関係を解除する必要があります。
- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

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

- PVCで指定する必要がありますが、`StorageClass`Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はあ

りません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。

- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、最初にににに設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合 `delete`にすると、PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、 Trident はPVC を管理し、バックエンドの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と `--no-rename`ボリューム名を保持するためのフラグ。
  - `--no-manage`* - を使用する場合 `--no-manage`フラグが設定されている場合、 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。
  - `--no-rename`* - を使用する場合 `--no-rename`フラグを使用すると、 `Trident` はボリュームのインポート時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）、および `ontap-san-economy`ドライバー。

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

ボリュームをインポートします

`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。

   最小仕様の例を次に示します。

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

   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) 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について、次のボリュームインポートの例を確認してください。

ONTAP NASおよびONTAP NAS FlexGroup

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

- Tridentは、 `ontap-nas-economy` ドライバ。
- 。 `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

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

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。

管理対象ボリューム

次の例は、という名前のボリュームをインポートします `managed_volume` という名前のバックエンドで `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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理対象外のボリューム

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

次に、をインポートする例を示します `unmanaged_volume` をクリックします `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。

管理対象ボリューム

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>
```

管理対象外のボリューム

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。

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：

管理対象ボリューム

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

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

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

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

管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `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ノードのIQNとIQNを共有するigroupにLUNをマッピングすると、エラーが表示されます。 `LUN already mapped to initiator(s) in this group`。ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

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

要素（ **Element** ）

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

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

次に、をインポートする例を示します `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    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files の特長

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

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

次に、をインポートする例を示します `azure-netapp-files` バックエンドのボリューム

azurenetappfiles_40517 を指定します 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ボリューム

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

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

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

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

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

ボリュームの名前とラベルをカスタマイズする

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

作業を開始する前に

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、クローニングの各処理。
- の場合 `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。
- の場合 `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

- カスタム 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。
- カスタムラベルは、 `ontap-san`、 `ontap-nas`、 そして `ontap-nas-flexgroup` ドライバー。
- 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無効な構文が原因でエラーが発生した場合、バックエンドの作成は失敗します。ただし、テンプレートアプリケーションが失敗した場合は、既存の命名規則に従ってボリュームに名前が付けられます。
- バックエンド構成の名前テンプレートを使用してボリュームの名前が指定されている場合、ストレージプレフィックスは適用されません。任意のプレフィックス値をテンプレートに直接追加できます。

名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベルで定義できます。

ルートレベルの例

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

考慮すべきポイント

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

ネームスペース間で**NFS**ボリュームを共有します

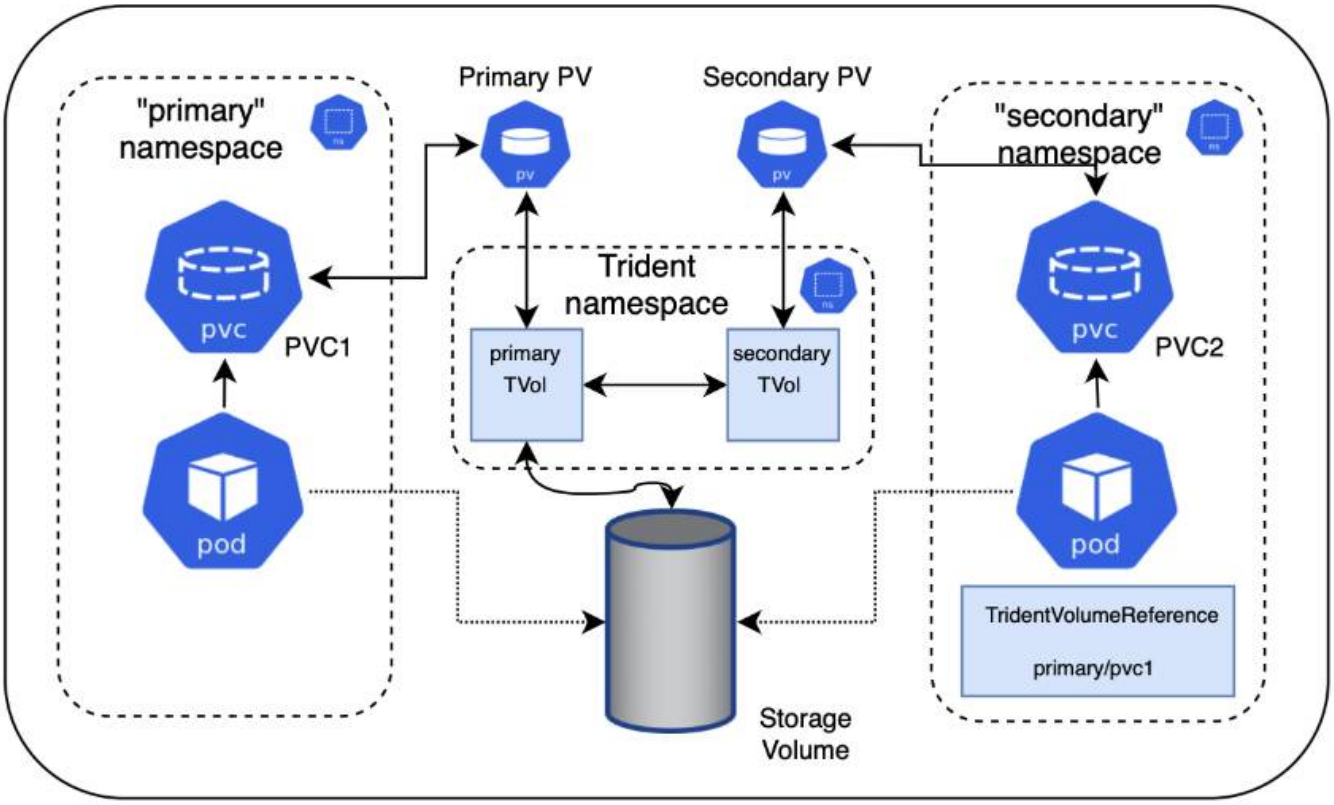
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

の機能

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

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

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



クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1 ボリュームを共有するようにソース**PVC**を設定します

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

2 デスティネーションネームスペースに**CR**を作成する権限を付与します

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

3 デスティネーションネームスペースに**TridentVolumeReference**を作成します

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

4 宛先名前空間に下位**PVC**を作成します

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

ソースネームスペースとデスティネーションネームスペースを設定します

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

手順

1. ソース名前空間の所有者： PVCを作成します (`pvc1`）をソースネームスペースに追加し、デスティネーションネームスペースとの共有権限を付与します (`namespace2`)を使用します `shareToNamespace` アノテーション

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

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

   - カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例： `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
   - を使用して、すべてのネームスペースに共有できます `*`。例： `trident.netapp.io/shareToNamespace: *`
   - PVCを更新してを含めることができます `shareToNamespace` アノテーションはいつでも作成できます。

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. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `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を作成します (pvc2）をデスティネーションネームスペースに展開します (namespace2)を使用します shareFromPVC 送信元PVCを指定する注釈。

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

共有ボリュームを削除

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

使用 tridentctl get 下位のボリュームを照会する

を使用する[tridentctl ユーティリティを使用すると、を実行できます get コマンドを使用して下位のボリ

ュームを取得します。詳細については、リンク:./trident-reference/tridentctl.htmlを参照してください[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：クエリを下位のソースボリュームに制限します。
- --subordinateOf string:クエリをボリュームの下位に制限します。

制限

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

を参照してください。

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

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

ネームスペース全体でボリュームをクローニング

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

前提条件

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

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

ボリュームクローニングはわずか数ステップでセットアップできます。


ボリュームのクローンを作成するためのソース**PVC**の設定

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


デスティネーションネームスペースに**CR**を作成する権限を付与します

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


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

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


デスティネーションネームスペースにクローン**PVC**を作成します。

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

ソースネームスペースとデスティネーションネームスペースを設定します

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

手順

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

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

Tridentは、PVとそのバックエンドストレージボリュームを作成します。

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

2. クラスタ管理者: 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. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `pvc1`。

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

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

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

制限

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

SnapMirrorによるボリュームのレプリケート

Tridentでは、ディザスタリカバリ用にデータをレプリケートするために、ピア関係にあるクラスタのソースボリュームとデスティネーションボリュームの間のミラー関係をサポートしています。 Trident Mirror Relationship (TMR) と呼ばれる名前空間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（PVC）間のミラー関係を作成する
- ボリューム間のミラー関係の削除
- ミラー関係を解除する
- 災害時（フェイルオーバー）にセカンダリボリュームを昇格する
- クラスタからクラスタへのアプリケーションのロスレス移行の実行（計画的なフェイルオーバーまたは移行時）

レプリケーションの前提条件

作業を開始する前に、次の前提条件を満たしていることを確認してください。

ONTAP クラスタ

- * Trident *：Tridentバージョン22.10以降が、バックエンドとしてONTAPを利用するソースとデスティネーションの両方のKubernetesクラスタに存在している必要があります。
- ライセンス：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデステ

ィネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

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

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にピア関係が設定されている必要があります。詳細については、を参照してください "クラスタと SVM のピアリングの概要" 。

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

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

サポートされるドライバ

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。ASAr2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の詳細" 。

ミラーPVCの作成

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

手順

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

例

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

b. 以前に作成したStorageClassを使用してPVCを作成します。

例

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

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

例

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

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

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

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

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

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

      例

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

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

例

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

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

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

例

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

TridentはTridentMirrorRelationship CRDをチェックし、関係が存在しない場合はボリュームの作成に失敗します。関係が存在する場合、Tridentは新しいFlexVol volumeを、MirrorRelationshipで定義されているリモートSVMとピア関係にあるSVMに配置します。

ボリュームレプリケーションの状態

Trident Mirror Relationship（TMR）は、PVC間のレプリケーション関係の一端を表すCRDです。宛先TMRには、目的の状態をTridentに通知する状態があります。宛先TMRの状態は次のとおりです。

- 確立済み：ローカルPVCはミラー関係のデスティネーションボリュームであり、これは新しい関係です。
- 昇格：ローカルPVCはReadWriteでマウント可能であり、ミラー関係は現在有効ではありません。

- * reestablished *：ローカルPVCはミラー関係のデスティネーションボリュームであり、以前はそのミラー関係に含まれていました。
  - デスティネーションボリュームはデスティネーションボリュームの内容を上書きするため、ソースボリュームとの関係が確立されたことがある場合は、reestablished状態を使用する必要があります。
  - ボリュームが以前にソースとの関係になかった場合、再確立状態は失敗します。

計画外フェールオーバー時にセカンダリPVCを昇格する

セカンダリKubernetesクラスタで次の手順を実行します。

- TridentMirrorRelationshipの_spec.state_フィールド をに更新します `promoted`。

計画的フェイルオーバー中にセカンダリPVCを昇格

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

手順

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

   例

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

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

フェイルオーバー後にミラー関係をリストアする

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

手順

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

その他の処理

Tridentでは、プライマリボリュームとセカンダリボリュームで次の処理がサポートされます。

新しいセカンダリ**PVC**へのプライマリ**PVC**の複製

プライマリPVCとセカンダリPVCがすでに存在していることを確認します。

手順

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

ミラー、プライマリ、またはセカンダリ**PVC**のサイズ変更

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

PVCからのレプリケーションの削除

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

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

（以前にミラーリングされていた）**PVC**の削除

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

TMRの削除

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

ONTAPがオンラインのときにミラー関係を更新

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

ONTAPがオフラインの場合にミラー関係を更新

CRDを使用すると、TridentがONTAPクラスタに直接接続されていなくてもSnapMirror更新を実行できます。次のTridentActionMirrorUpdateの形式例を参照してください。

例

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

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

CSI トポロジを使用します

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

概要

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

CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

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

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

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

必要なもの

CSI トポロジを使用するには、次のものが必要です。

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

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"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、トポロジ対応と `topology.kubernetes.io/zone`を示すラベルを付ける必要があり(`topology.kubernetes.io/region`ます。これらのラベル*は、Tridentをトポロジ対応にするためにTridentをインストールする前に、クラスタ内のノード*に設定しておく必要があります。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.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"}]
```

手順 1 ：トポロジ対応バックエンドを作成する

Tridentストレージバックエンドは、アベイラビリティゾーンに基づいて選択的にボリュームをプロビジョニングするように設計できます。各バックエンドは、サポートされているゾーンとリージョンのリストを表すオプ

ションのブロックを運ぶことができます `supportedTopologies` 。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

YAML

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

また ' ストレージ・プールごとに 'upportedTopologies を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、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 2 ：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  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
```

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

ステップ 3 ： PVC を作成して使用する

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

以下の例「 PEC 」を参照してください。

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

このマニフェストを使用して PVC を作成すると、次のような結果になります。

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

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

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

この podSpec は 'us-east1` 領域に存在するノード上のポッドをスケジュールするよう Kubernetes に指示し 'us-east1-a' または `us-east1-b' ゾーン内に存在する任意のノードから選択します

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追加 `supportedTopologies`

既存のバックエンドは 'tridentctl backend update を使用して 'upportedTopologies のリストを含むように更新できますこれは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

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

スナップショットを操作します

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

概要

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

作業を開始する前に

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

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

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

ボリューム **Snapshot** を作成します

手順

1. を作成します `VolumeSnapshotClass`。詳細については、を参照してください "ボリュームSnapshotクラス"。
   - は `driver`Trident CSIドライバを示しています。
   - `deletionPolicy` は、です `Delete` または `Retain`。に設定すると `Retain`を使用すると、ストレージクラスタの基盤となる物理Snapshotが、の場合でも保持されます `VolumeSnapshot` オブジェクトが削除された。

   例

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

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

2. 既存のPVCのスナップショットを作成します。

   例

   - 次に、既存のPVCのスナップショットを作成する例を示します。

   ```
   cat snap.yaml
   ```

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

   - 次の例は、という名前のPVCのボリュームSnapshotオブジェクトを作成します。 `pvc1` Snapshotの名前はに設定されます `pvc1-snap`。ボリュームSnapshotはPVCに似ており、に関連付けられています

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
```

- 次の情報を確認できます。 VolumeSnapshotContent のオブジェクト pvc1-snap ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。。 Snapshot Content Name このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 Ready To Use パラメータは、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SnapshotからPVCを作成

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

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

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

```
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のインポート

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

作業を開始する前に

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

手順

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

     。 `<volumeSnapshotContentName>` CRの命名規則のため、バックエンドスナップショット名が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` を参照するCR `VolumeSnapshotContent` オブジェクト。これにより、 `VolumeSnapshot` 指定された名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap` を参照しています。 `VolumeSnapshotContent` 名前付き `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（アクション不要）：*外部スナップショットは、新しく作成されたを認識して `VolumeSnapshotContent`呼び出しを実行します `ListSnapshots`。Tridentによってが作成され`TridentSnapshot`ます。
   - 外部スナップショットは、 `VolumeSnapshotContent` 終了： `readyToUse` および `VolumeSnapshot` 終了： `true`。
   - Tridentのリターン `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim` 新しい `VolumeSnapshot`を参照してください `spec.dataSource` （または `spec.dataSourceRef`）nameは `VolumeSnapshot` 名前。

例

次に、を参照するPVCを作成する例を示します。 `VolumeSnapshot` 名前付き `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
```

Snapshotを使用してボリュームデータをリカバリします

snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` スナップショットからデータを直接リカバリするディレクトリ。

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

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

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

Snapshotからのインプレースボリュームのリストア

Tridentでは、（TASR）CRを使用してSnapshotからボリュームをインプレースで迅速にリストアできます `TridentActionSnapshotRestore`。このCRはKubernetesの必須アクションとして機能し、処理の完了後も維持されません。

Tridentは、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` 、 `google-cloud-netapp-volumes` 、 そして `solidfire-san`ドライバー。

作業を開始する前に

バインドされたPVCと使用可能なボリュームSnapshotが必要です。

- PVCステータスがバインドされていることを確認します。

```
kubectl get pvc
```

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

```
kubectl get vs
```

手順

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

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

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

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

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

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

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

結果

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

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

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

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

Snapshotが関連付けられているPVを削除する

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

ボリュームSnapshotコントローラの導入

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

手順

1. ボリュームのSnapshot作成

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

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.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、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

関連リンク

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

ボリュームグループスナップショットの操作

Kubernetes の永続ボリューム（PV）の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、同じ時点で作成された複数のボリューム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、次のストレージ ドライバーでサポートされます。

- `ontap-san`ドライバー - 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 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。

作業を開始する前に

- 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とカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

  Kubernetesディストリビューションに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Snapshotコントローラの導入 。

  > GKE 環境でオンデマンド ボリューム 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では、内蔵の非表示のスナップショットコントローラを使用します。

- 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、を参照してください "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。

。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ユニットとして復元することはできません。

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

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

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

Snapshotからのインプレースボリュームのリストア

Tridentでは、（TASR）CRを使用してSnapshotからボリュームをインプレースで迅速にリストアできます `TridentActionSnapshotRestore`。このCRはKubernetesの必須アクションとして機能し、処理の完了後も維持されません。

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

関連付けられた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、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Snapshotコントローラの導入

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

手順

1. ボリュームのSnapshot作成

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

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
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、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

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 Snapshot"