ボリューム操作を実行する
Astra Trident

NetApp
November 20, 2023

This PDF was generated from https://docs.netapp.com/ja-jp/trident-2107/trident-use/csi-topology.html on November 20, 2023. Always check docs.netapp.com for the latest.

目次

ボリューム操作を実行する 1
CSI トポロジを使用します 1
スナップショットを操作します 8
ボリュームを展開します 12
ボリュームをインポート 19

ボリューム操作を実行する

Trident がボリュームを管理するための各種機能をご紹介します。

- "CSI トポロジを使用します"
- "スナップショットを操作します"
- "ボリュームを展開します"
- "ボリュームをインポート"

CSI トポロジを使用します

Astra Trident では、を使用して、 Kubernetes クラスタ内にあるノードにボリュームを選択的に作成して接続できます "CSI トポロジ機能"。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、アベイラビリティゾーン内の異なるリージョンに配置することも、さまざまなアベイラビリティゾーンに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をプロビジョニングするために、 Astra Trident は CSI トポロジを使用します。

> 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。

- を使用 `VolumeBindingMode` をに設定します `Immediate`トポロジを認識することなくボリュームを作成できます。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。これがデフォルトです `VolumeBindingMode` また、トポロジの制約を適用しないクラスタにも適しています。永続ボリュームは、要求側ポッドのスケジュール要件に依存せずに作成されます。
- を使用 `VolumeBindingMode` をに設定します `WaitForFirstConsumer`PVCの永続的ボリュームの作成とバインディングは、PVCを使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

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

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- 1.17 以降を実行する Kubernetes クラスタ。

```
$ 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、トポロジを認識するためのラベルが必要です (`topology.kubernetes.io/region` および `topology.kubernetes.io/zone`）。このラベル * は、 Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

```
$ 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"}]
```

手順 **1** ：トポロジ対応バックエンドを作成する

Astra 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伝送できます `supportedTopologies` サポートする必要があるゾーンおよび領域のリストを表すブロック。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。

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

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

を定義できます `supportedTopologies` ストレージプールごとに作成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": "Netapp123",
"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"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
         "region": "Iowa-DC",
         "zone": "Iowa-DC-B",
         "supportedTopologies": [
             {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
         ]
     }
]
}
```

この例では、を使用しています `region` および `zone` ラベルはストレージプールの場所を表します。`topology.kubernetes.io/region` および `topology.kubernetes.io/zone` ストレージプールの使用場所を指定します。

手順 2 ：トポロジを認識するストレージクラスを定義する

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

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
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、 `volumeBindingMode` がに設定されます `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。および、 `allowedTopologies` 使用するゾーンとリージョンを提供します。。 `netapp-san-us-east1` StorageClassがにPVCを作成します `san-backend-us-east1` 上で定義したバックエンド。

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。

例を参照 `spec` 下記：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
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により、Kubernetesは、にあるノードにPODをスケジュールするように指示されます `us-east1` リージョンを選択し、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追加 `supportedTopologies`

既存のバックエンドを更新して、のリストを追加することができます `supportedTopologies` を使用します `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、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

Astra Trident の 20.01 リリースから、 Kubernetes レイヤで PVS のスナップショットを作成できるようになりました。この Snapshot を使用して、 Astra Trident で作成されたボリュームのポイントインタイムコピーを管理し、追加のボリューム（クローン）の作成をスケジュールできます。ボリュームSnapshotは、でサポートされています `ontap-nas`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `aws-cvs`、 `gcp-cvs`および `azure-netapp-files` ドライバ。

> この機能は Kubernetes 1.17 （ベータ版）から提供され、 1.20 から GA になります。ベータ版から GA 版への移行に伴う変更点については、を参照してください "リリースのブログ"。一般に卒業したときに、 `v1` APIのバージョンが導入され、との後方互換性が確保されています `v1beta1` Snapshot：

必要なもの

- ボリューム Snapshot を作成するには、外部の Snapshot コントローラとカスタムリソース定義（ CRD ）を作成する必要があります。使用されている Kubernetes Orchestrator （例： Kubeadm 、 GKE 、 OpenShift ）の役割を担っています。

次のように、外部スナップショットコントローラとスナップショット作成 SSD を作成できます。

1. ボリューム Snapshot の作成：

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

2. 目的のネームスペースにスナップショットコントローラを作成します。以下の YAML マニフェストを編集して名前空間を変更します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

> CSI Snapshotter は、を提供します "webhook を検証しています" ユーザーが既存の v1beta1 スナップショットを検証し、有効なリソースオブジェクトであることを確認できるようにするため。検証中の webhook は、無効なスナップショットオブジェクトに自動的にラベルを付け、今後無効なオブジェクトが作成されないようにします。検証する webhook は Kubernetes Orchestrator によって導入されます。検証するウェブフックを手動で配備する手順を参照してください "こちらをご覧ください"。無効なスナップショットマニフェストの例を探します "こちらをご覧ください"。

以下に、スナップショットの操作に必要な構成要素と、スナップショットの作成方法および使用方法の例を示します。

手順1：を設定します `VolumeSnapshotClass`

ボリュームSnapshotを作成する前に、リンク./trident-reference/objects.htmlを設定します[`VolumeSnapshotClass`^]をクリックします。

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 `driver` Astra TridentのCSIドライバをポイントします。 `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。

手順 2 ：既存の PVC のスナップショットを作成します

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

という名前のPVCに対してスナップショットが作成されています `pvc1`および`Snapshot`の名前がに設定されます `pvc1-snap`。

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

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

これでが作成されました `VolumeSnapshot` オブジェクト。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。

を識別できます `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。

```
$ 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
.
.
```

。 `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、Snapshotを使用して新しいPVCを作成できることを示します。

手順 3 ：ボリューム Snapshot から PVC を作成します

スナップショットを使用して PVC を作成する例は、次のとおりです。

```
$ 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
```

`dataSource` に、という名前のボリュームSnapshotを使用してPVCを作成する必要があることを示します `pvc1-snap` データのソースとして。このコマンドを実行すると、 Astra Trident が Snapshot から PVC を作成するように指示します。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。

> スナップショット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「削除状態」に更新されます。Astra Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

詳細については、こちらをご覧ください

- "ボリューム Snapshot"
- リンク:./trident-reference/objects.html[`VolumeSnapshotClass`^]

ボリュームを展開します

Astra Trident により、 Kubernetes ユーザは作成後にボリュームを拡張できます。ここでは、 iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

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

> iSCSIボリューム拡張は、でサポートされます `ontap-san`、 `ontap-san-economy`、 `solidfire-san` ドライバとにはKubernetes 1.16以降が必要です。

概要

iSCSI PV の拡張には、次の手順が含まれます。

- StorageClass定義を編集してを設定します `allowVolumeExpansion` フィールドからに移動します `true`。
- PVC定義を編集してを更新します `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。
- サイズを変更するには、 PV をポッドに接続する必要があります。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、 Astra Trident はストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
  - 未接続の PV のサイズを変更しようとすると、 Astra Trident が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。

次の例は、 iSCSI PVS の仕組みを示しています。

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

```
$ 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
```

既存のストレージクラスの場合は、編集してを追加します `allowVolumeExpansion` パラメータ

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

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

Astra Trident が、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

この例では、を使用するポッドが作成されます `san-pvc`。

```
$  kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-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:    centos-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の定義を編集してを更新します `spec.resources.requests.storage` 2Giへ。

```
$ 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 、 Astra Trident の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ているかどうかを検証できます。

```
$ 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

Astra Tridentは、でプロビジョニングしたNFS PVSのボリューム拡張をサポートしています `ontap-nas`、`ontap-nas-economy`、 `ontap-nas-flexgroup`、 `aws-cvs`、 `gcp-cvs``および `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、管理者はまず、を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 `allowVolumeExpansion` フィールドからに移動します `true`：

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

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編

集するだけです `kubectl edit storageclass` ボリュームを拡張できるようにするため。

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
```

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

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

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       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20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します `spec.resources.requests.storage` 1 GBに設定する場合：

```
$ 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 、 Astra Trident の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ている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    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

ボリュームをインポート

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

ボリュームインポートをサポートするドライバ

次の表は、ボリュームのインポートをサポートするドライバと、それらのアップグレードが導入されたリリースを示しています。

ドライバ	リリース。
`ontap-nas`	19.04
`ontap-nas-flexgroup`	19.04
`solidfire-san`	19.04

ドライバ	リリース。
aws-cvs	19.04
azure-netapp-files	19.04
gcp-cvs	19.04
ontap-san	19.04

ボリュームをインポートする理由

Trident にボリュームをインポートするユースケースはいくつかあります。

- アプリケーションのコンテナ化と既存のデータセットの再利用
- エフェメラルアプリケーション用のデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタの再構築
- 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する

インポートはどのように機能しますか。

Persistent Volume Claim （ PVC ；永続ボリューム要求）ファイルは、ボリュームインポートプロセスで PVC を作成するために使用されます。少なくとも、次の例に示すように、 PVC ファイルには name 、 namespace 、 accessModes 、および storageClassName フィールドが含まれている必要があります。

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

。 `tridentctl` クライアントは、既存のストレージボリュームをインポートするために使用されます。Trident は、ボリュームのメタデータを保持し、 PVC と PV を作成することで、ボリュームをインポートします。

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

ストレージボリュームをインポートするには、ボリュームが含まれている Astra Trident バックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ ONTAP FlexVol 、 Element Volume 、 CVS ボリュー

ムパスなど）を指定します。ストレージボリュームは、読み取り / 書き込みアクセスを許可し、指定された Astra Trident バックエンドからアクセスできる必要があります。。 `-f` string引数は必須で、YAMLまたはJSON PVCファイルへのパスを指定します。

Astra Trident がインポートボリューム要求を受信すると、既存のボリュームサイズが決定され、 PVC で設定されます。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。

を使用してボリュームがインポートされる場合 `--no-manage` 引数として、Tridentはオブジェクトのライフサイクルに関してPVCまたはPVに対する追加の操作を実行しません。TridentはのPVイベントとPVCイベントを無視するため `--no-manage` オブジェクト。PVを削除してもストレージボリュームは削除されません。ボリュームのクローンやサイズ変更などの他の処理も無視されます。このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、 Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

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

Trident 19.07 以降では、 PVS の添付ファイルを処理し、ボリュームをインポートの一環としてマウントします。以前のバージョンの Astra Trident を使用しているインポートの場合、データパスに処理は存在しないため、ボリュームをマウントできるかどうかがボリュームインポートで検証されません。ストレージクラスが正しくない場合など、ボリュームのインポートでミスが発生した場合は、PVの再利用ポリシーをに変更することでリカバリできます `retain`をクリックしてPVCとPVを削除し、volume importコマンドを再試行します。

`ontap-nas` および `ontap-nas-flexgroup` インポート

を使用して作成した各ボリューム `ontap-nas` driverはONTAP クラスタ上のFlexVol です。を使用してFlexVolをインポートする `ontap-nas` ドライバも同じように動作します。ONTAP クラスタにすでに存在するFlexVolは、としてインポートできます `ontap-nas` PVC。同様に、FlexGroup ボリュームはとしてインポートできます `ontap-nas-flexgroup` PVC

> Trident がインポートする ONTAP のタイプは RW である必要があります。DP 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。 Trident にボリュームを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。

> 。 `ontap-nas` ドライバでqtreeをインポートおよび管理できない。。 `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。

たとえば、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`では、次のコマンドを使用します。

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

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

という名前のボリュームをインポートします `unmanaged_volume` （上 `ontap_nas backend`）を使用します。Tridentは管理しません。次のコマンドを使用します。

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

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

を使用する場合 `--no-manage` Tridentは、ボリュームの名前を変更したり、ボリュームがマウントされたかどうかを検証したりすることはありません。ボリュームが手動でマウントされていない場合、ボリュームインポート処理は失敗します。

> UnixPermissions カスタムのボリュームをインポートするという既存のバグが修正されました。PVC 定義またはバックエンド構成に unixPermissions を指定し、必要に応じて Astra Trident にボリュームをインポートするように指示できます。

`ontap-san` インポート

Astra Trident は、 1 つの LUN を含む ONTAP SAN FlexVol をインポートすることもできます。これはと同じです `ontap-san` ドライバ。FlexVol 内の各PVCおよびLUNにFlexVol を作成します。を使用できます `tridentctl import` 他の場合と同様にコマンドを実行します。

- の名前を含めます `ontap-san` バックエンド：

- インポートする必要がある FlexVol の名前を指定します。この FlexVol には、インポートが必要な LUN が 1 つしか含まれていないことに注意してください。
- とともに使用する必要があるPVC定義のパスを指定します `-f` フラグ。
- PVC を管理するか、管理対象外にするかを選択します。デフォルトでは、 Trident によって PVC が管理され、バックエンドの FlexVol と LUN の名前が変更されます。管理対象外のボリュームとしてインポートするには、を渡します `--no-manage` フラグ。

> 管理対象外のをインポートする場合 `ontap-san` ボリューム：FlexVol 内のLUNの名前がになっていることを確認します `lun0` とは、目的のイニシエータを含むigroupにマッピングされている。Trident が管理対象のインポートに対して自動的に処理します。

次に、 Astra Trident が FlexVol をインポートし、 PVC 定義に関連付けます。Astra Tridentは、FlexVol の名前もに変更します `pvc-<uuid>` およびFlexVol 内のLUNをからにフォーマットします `lun0`。

> 既存の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場合は、最初にボリュームをクローニングしてからインポートを実行します。

例

をインポートします `ontap-san-managed` にあるFlexVol `ontap_san_default` バックエンドでを実行します `tridentctl import` コマンドの形式：

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

> ONTAP ボリュームのタイプが RW であることが Astra Trident でインポートされる必要があります。DP 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 Astra 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。

`element` インポート

Trident を使用して、 NetApp Element ソフトウェア / NetApp HCI ボリュームを Kubernetes クラスタにインポートできます。必要に応じて、Astra Tridentバックエンドの名前、ボリュームとPVCファイルの一意の名前をの引数として指定します `tridentctl import` コマンドを実行します

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

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

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。ボリューム名が重複している場合、 Trident のボリュームインポートプロセス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ます。次に、クローンボリュームをインポートします。

`aws-cvs` インポート

NetApp Cloud Volumes Service がサポートするボリュームを AWS でインポートするには、名前ではなくボリュームパスでボリュームを特定します。

をインポートします `aws-cvs` バックエンドのボリュームの名前はです `awscvs_YEppr` を指定します `adroit-jolly-swift`では、次のコマンドを使用します。

```
$ tridentctl import volume awscvs_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 | aws-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
```

ボリュームパスは、 / のあとのボリュームのエクスポートパスの部分です。たとえば、エクスポートパスがの場合などです `10.0.0.1:/adroit-jolly-swift`、ボリュームのパスはです `adroit-jolly-swift`。

gcp-cvs インポート

をインポートする gcp-cvs ボリュームは、のインポートと同じように機能します aws-cvs ボリューム：

azure-netapp-files インポート

をインポートします azure-netapp-files バックエンドのボリュームの名前はです azurenetappfiles_40517 を指定します `importvol1`を使用して、次のコマンドを実行します。

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

ANF ボリュームのボリュームパスは、 / のあとのマウントパスにあります。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。

著作権に関する情報

Copyright © 2023 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

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

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

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

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

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

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

商標に関する情報

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