볼륨을 프로비저닝하고 관리합니다

Astra Trident

NetApp
April 03, 2024

목차

볼륨을 프로비저닝하고 관리합니다

볼륨을 프로비저닝합니다

구성된 Kubernetes StorageClass를 사용하여 PV에 대한 액세스를 요청하는 PersistentVolume(PV) 및 PersistentVolumeClaim(PVC)을 생성합니다. 그런 다음 PV를 포드에 장착할 수 있습니다.

개요

A "지속성 볼륨 _" (PV)는 Kubernetes 클러스터에서 클러스터 관리자가 프로비저닝한 물리적 스토리지 리소스입니다. 를 클릭합니다 "_PersistentVolumeClaim" (PVC)는 클러스터의 PersistentVolume에 대한 액세스 요청입니다.

PVC는 특정 크기 또는 액세스 모드의 저장을 요청하도록 구성할 수 있습니다. 클러스터 관리자는 연결된 StorageClass를 사용하여 PersistentVolume 크기 및 액세스 모드(예: 성능 또는 서비스 수준)를 제어할 수 있습니다.

PV 및 PVC를 생성한 후 포드에 볼륨을 장착할 수 있습니다.

샘플 매니페스트

PersistentVolume 샘플 매니페스트

이 샘플 매니페스트는 StorageClass와 연결된 10Gi의 기본 PV를 보여 줍니다 `basic-csi`.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaim 샘플 매니페스트

이러한 예는 기본적인 PVC 구성 옵션을 보여줍니다.

RWO 액세스 **PVC**

이 예에서는 이름이 인 StorageClass와 연결된 RWO 액세스 권한이 있는 기본 PVC를 보여 줍니다 `basic-csi`.

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

NVMe/TCP가 있는 **PVC**

이 예에서는 이름이 인 StorageClass와 연결된 RWO 액세스 권한이 있는 NVMe/TCP용 기본 PVC를 보여 줍니다 `protection-gold`.

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

POD 매니페스트 샘플

이 예는 PVC를 포드에 부착하기 위한 기본 구성을 보여줍니다.

기본 구성

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

기본 **NVMe/TCP** 구성

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      resources: {}
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: pvc-san-nvme
```

PV 및 PVC를 작성합니다

단계

1. PV를 만듭니다.

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

2. PV 상태를 확인한다.

```
kubectl get pv
NAME        CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. PVC를 작성합니다.

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

4. PVC 상태를 확인합니다.

```
kubectl get pvc
NAME          STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi       RWO                          5m
```

5. 볼륨을 Pod에 마운트합니다.

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

> 을 사용하여 진행 상황을 모니터링할 수 있습니다 `kubectl get pod --watch`.

6. 볼륨이 에 마운트되어 있는지 확인합니다 `/my/mount/path`.

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

7. 이제 Pod를 삭제할 수 있습니다. Pod 응용 프로그램은 더 이상 존재하지 않지만 볼륨은 유지됩니다.

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

을 참조하십시오 "Kubernetes 및 Trident 오브젝트" 스토리지 클래스가 와 상호 작용하는 방법에 대한 자세한 내용은 을 참조하십시오 `PersistentVolumeClaim` 및 Astra Trident가 볼륨을 프로비저닝하는 방법을 제어하는 매개 변수가 포함됩니다.

볼륨 확장

Astra Trident를 사용하면 Kubernetes 사용자가 볼륨을 생성한 후 확장할 수 있습니다. iSCSI 및 NFS 볼륨을 확장하는 데 필요한 구성에 대한 정보를 찾습니다.

iSCSI 볼륨을 확장합니다

CSI 프로비저닝을 사용하여 iSCSI PV(Persistent Volume)를 확장할 수 있습니다.

> iSCSI 볼륨 확장은 에서 지원합니다 `ontap-san`, `ontap-san-economy`, `solidfire-san` Kubernetes 1.16 이상이 필요합니다.

1단계: 볼륨 확장을 지원하도록 **StorageClass**를 구성합니다

StorageClass 정의를 편집하여 를 설정합니다 `allowVolumeExpansion` 필드를 눌러 로 이동합니다 `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
```

이미 존재하는 StorageClass 의 경우 를 포함하도록 편집합니다 `allowVolumeExpansion` 매개 변수.

2단계: 생성한 **StorageClass**를 사용하여 **PVC**를 생성합니다

PVC 정의를 편집하고 를 업데이트합니다 `spec.resources.requests.storage` 원래 크기보다 커야 하는 새로 원하는 크기를 반영합니다.

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

Astra Trident가 PV(Persistent Volume)를 생성하여 이 PVC(Persistent Volume Claim)와 연결합니다.

```
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
```

3단계**: PVC**를 부착하는 **POD**를 정의합니다

크기를 조정할 수 있도록 PV를 포드에 연결합니다. iSCSI PV의 크기를 조정할 때 두 가지 시나리오가 있습니다.

- PV가 포드에 연결된 경우 Astra Trident는 스토리지 백엔드의 볼륨을 확장하고 디바이스를 다시 검사하며 파일 시스템의 크기를 조정합니다.
- 연결되지 않은 PV의 크기를 조정하려고 하면 Astra Trident가 스토리지 백엔드의 볼륨을 확장합니다. PVC가 POD에 바인딩되면 Trident가 디바이스를 다시 검사해 파일 시스템의 크기를 조정합니다. 그런 다음 확장 작업이 성공적으로 완료된 후 Kubernetes에서 PVC 크기를 업데이트합니다.

이 예제에서는 을 사용하는 포드가 만들어집니다 `san-pvc`.

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

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

4단계: **PV**를 확장합니다

1Gi 에서 2Gi 로 생성된 PV 의 크기를 조정하려면 PVC 정의를 편집하고 를 업데이트합니다 `spec.resources.requests.storage` 2Gi

```
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
 ...
```

5단계: 확장 확인

PVC, PV, Astra Trident 볼륨의 크기를 확인하여 확장이 제대로 작동하는지 확인할 수 있습니다.

```
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 볼륨을 확장합니다

Astra Trident는 에 프로비저닝된 NFS PVS에 대한 볼륨 확장을 지원합니다 `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `gcp-cvs`, 및 `azure-netapp-files` 백엔드.

1단계: 볼륨 확장을 지원하도록 **StorageClass**를 구성합니다

NFS PV의 크기를 조정하려면 관리자가 먼저 을 설정하여 볼륨 확장을 허용하도록 스토리지 클래스를 구성해야 합니다 `allowVolumeExpansion` 필드를 눌러 로 이동합니다 `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
```

이 옵션 없이 스토리지 클래스를 이미 생성한 경우 를 사용하여 기존 스토리지 클래스를 간단히 편집할 수 있습니다 `kubectl edit storageclass` 볼륨 확장을 허용합니다.

2단계: 생성한 **StorageClass**를 사용하여 **PVC**를 생성합니다

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

Astra Trident는 이 PVC에 대해 20MiB NFS PV를 생성해야 합니다.

```
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
```

3단계: **PV**를 확장합니다

새로 생성된 20MiB PV의 크기를 1GiB로 조정하려면 PVC를 편집하고 설정합니다
`spec.resources.requests.storage` 1GiB 증가:

```
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
...
```

4단계**:** 확장을 확인합니다

PVC, PV, Astra Trident 볼륨의 크기를 확인하여 크기가 올바르게 작동하는지 확인할 수 있습니다.

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

볼륨 가져오기

를 사용하여 기존 스토리지 볼륨을 Kubernetes PV로 가져올 수 있습니다 `tridentctl import`.

개요 및 고려 사항

다음과 같은 작업을 위해 Astra Trident로 볼륨을 가져올 수 있습니다.

- 응용 프로그램을 Containerize 하고 기존 데이터 집합을 다시 사용합니다
- 수명이 짧은 애플리케이션에 사용할 데이터 세트의 클론을 사용합니다
- 오류가 발생한 Kubernetes 클러스터를 재구성합니다
- 재해 복구 중에 애플리케이션 데이터 마이그레이션

고려 사항

볼륨을 가져오기 전에 다음 고려 사항을 검토하십시오.

- Astra Trident는 RW(읽기-쓰기) 유형의 ONTAP 볼륨만 가져올 수 있습니다. DP(데이터 보호) 유형 볼륨은

SnapMirror 대상 볼륨입니다. Astra Trident로 볼륨을 가져오기 전에 미러 관계를 끊어야 합니다.

- 활성 연결이 없는 볼륨을 가져오는 것이 좋습니다. 활성 볼륨을 가져오려면 볼륨을 클론한 다음 가져오기를 수행합니다.

Kubernetes가 이전 연결을 인식하지 못하고 활성 볼륨을 POD에 쉽게 연결할 수 있기 때문에 블록 볼륨에서 특히 중요합니다. 이로 인해 데이터가 손상될 수 있습니다.

- 하지만 `StorageClass` PVC에 지정해야 하며 Astra Trident는 가져오는 동안 이 매개 변수를 사용하지 않습니다. 스토리지 클래스는 볼륨 생성 중에 스토리지 특성에 따라 사용 가능한 풀에서 선택하는 데 사용됩니다. 볼륨이 이미 있으므로 가져오는 동안 풀을 선택할 필요가 없습니다. 따라서 볼륨이 PVC에 지정된 스토리지 클래스와 일치하지 않는 백엔드 또는 풀에 있더라도 가져오기에 실패합니다.
- 기존 체적 크기는 PVC에서 결정되고 설정됩니다. 스토리지 드라이버에서 볼륨을 가져온 후 PV는 PVC에 대한 ClaimRef를 사용하여 생성됩니다.
  - 처음에 부가세 반환 청구액 정책이 로 설정되어 있습니다 `retain` PV에서 Kubernetes에서 PVC 및 PV를 성공적으로 바인딩하면 스토리지 클래스의 부가세 반환 청구액 정책에 맞게 부가세 반환 청구액 정책이 업데이트됩니다.
  - 스토리지 클래스의 부가세 반환 청구액 정책이 인 경우 `delete`, PV 삭제 시 저장 볼륨이 삭제된다.
- 기본적으로 Astra Trident는 PVC를 관리하고 백엔드에서 FlexVol 및 LUN의 이름을 바꿉니다. 을(를) 통과할 수 있습니다 `--no-manage` 관리되지 않는 볼륨을 가져오려면 플래그를 지정합니다. 를 사용하는 경우 `--no-manage`, Astra Trident는 개체의 수명 주기 동안 PVC 또는 PV에 대한 추가 작업을 수행하지 않습니다. PV가 삭제되어도 스토리지 볼륨은 삭제되지 않으며 볼륨 클론 및 볼륨 크기 조정과 같은 다른 작업도 무시됩니다.

이 옵션은 컨테이너화된 워크로드에 Kubernetes를 사용하고, 그렇지 않고 Kubernetes 외부 스토리지 볼륨의 라이프사이클을 관리하려는 경우에 유용합니다.

- PVC 및 PV에 주석이 추가되어 용적을 가져온 후 PVC와 PV가 관리되었는지 여부를 나타내는 두 가지 목적으로 사용됩니다. 이 주석은 수정하거나 제거할 수 없습니다.

볼륨을 가져옵니다

을 사용할 수 있습니다 `tridentctl import` 를 눌러 볼륨을 가져옵니다.

단계

1. 영구 볼륨 클레임(PVC) 파일(예: `pvc.yaml`)를 사용하여 PVC를 생성합니다. PVC 파일에는 다음이 포함되어야 합니다 `name`, `namespace`, `accessModes`, 및 `storageClassName`. 선택적으로 을 지정할 수 있습니다 `unixPermissions` PVC 정의.

   다음은 최소 사양의 예입니다.

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

PV 이름 또는 볼륨 크기와 같은 추가 매개 변수는 포함하지 마십시오. 이로 인해 가져오기 명령이 실패할 수 있습니다.

2. 를 사용합니다 `tridentctl import` 볼륨을 포함하는 Astra Trident 백엔드의 이름과 스토리지에서 볼륨을 고유하게 식별하는 이름(예: ONTAP FlexVol, Element Volume, Cloud Volumes Service 경로)을 지정하는 명령입니다. 를 클릭합니다 `-f` PVC 파일의 경로를 지정하려면 인수가 필요합니다.

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

예

지원되는 드라이버에 대한 다음 볼륨 가져오기 예를 검토하십시오.

ONTAP NAS 및 ONTAP NAS FlexGroup를 지원합니다

Astra Trident는 를 사용하여 볼륨 가져오기를 지원합니다 `ontap-nas` 및 `ontap-nas-flexgroup` 드라이버.

- 를 클릭합니다 `ontap-nas-economy` 드라이버가 qtree를 가져오고 관리할 수 없습니다.
- 를 클릭합니다 `ontap-nas` 및 `ontap-nas-flexgroup` 드라이버는 중복 볼륨 이름을 허용하지 않습니다.

로 생성된 각 볼륨입니다 `ontap-nas` 드라이버는 ONTAP 클러스터의 FlexVol입니다. 를 사용하여 FlexVol을 가져옵니다 `ontap-nas` 드라이버는 동일하게 작동합니다. ONTAP 클러스터에 이미 있는 FlexVol를 로 가져올 수 있습니다 `ontap-nas` PVC. 마찬가지로 FlexGroup vols는 로 가져올 수 있습니다 `ontap-nas-flexgroup` PVC

ONTAP NAS의 예

다음은 관리되는 볼륨 및 관리되지 않는 볼륨 가져오기의 예입니다.

관리 볼륨

다음 예에서는 라는 볼륨을 가져옵니다 `managed_volume` 백엔드에서 을(를) 선택합니다 `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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

관리되지 않는 볼륨

를 사용할 때 `--no-manage` argument, Astra Trident는 볼륨의 이름을 바꾸지 않습니다.

다음 예에서는 를 가져옵니다 `unmanaged_volume` 를 누릅니다 `ontap_nas` 백엔드:

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

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

ONTAP SAN

Astra Trident는 를 사용하여 볼륨 가져오기를 지원합니다 `ontap-san` 드라이버. 에서는 볼륨 가져오기가 지원되지 않습니다 `ontap-san-economy` 드라이버.

Astra Trident는 단일 LUN이 포함된 ONTAP SAN FlexVol을 가져올 수 있습니다. 이는 와 일치합니다 `ontap-san` 드라이버 - 각 PVC 및 FlexVol 내의 LUN에 대한 FlexVol를 생성합니다. Astra Trident는 FlexVol를 불러와 PVC 정의와 연결합니다.

ONTAP SAN 예

다음은 관리되는 볼륨 및 관리되지 않는 볼륨 가져오기의 예입니다.

관리 볼륨

관리 볼륨의 경우 Astra Trident가 FlexVol의 이름을 로 바꿉니다 `pvc-<uuid>` 및 FlexVol 내의 LUN을 에 포맷합니다 `lun0`.

다음 예제에서는 을 가져옵니다 `ontap-san-managed` 에 있는 FlexVol입니다 `ontap_san_default` 백엔드:

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

관리되지 않는 볼륨

다음 예에서는 를 가져옵니다 `unmanaged_example_volume` 를 누릅니다 `ontap_san` 백엔드:

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

다음 예에 표시된 것처럼 IQN을 Kubernetes 노드 IQN과 공유하는 igroup에 LUN이 매핑되어 있는 경우 오류가 발생합니다. `LUN already mapped to initiator(s) in this group`. 볼륨을 가져오려면 이니시에이터를 제거하거나 LUN 매핑을 해제해야 합니다.

```
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
```

요소

Astra Trident는 를 사용하여 NetApp Element 소프트웨어 및 NetApp HCI 볼륨 가져오기를 지원합니다 `solidfire-san` 드라이버.

> Element 드라이버는 중복 볼륨 이름을 지원합니다. 그러나 중복 볼륨 이름이 있는 경우 Astra Trident에서 오류를 반환합니다. 이 문제를 해결하려면 볼륨을 클론하고 고유한 볼륨 이름을 제공한 다음 복제된 볼륨을 가져옵니다.

요소 예제

다음 예제에서는 을 가져옵니다 `element-managed` 백엔드의 볼륨 `element_default`.

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

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

Google 클라우드 플랫폼

Astra Trident는 를 사용하여 볼륨 가져오기를 지원합니다 `gcp-cvs` 드라이버.

> Google Cloud Platform에서 NetApp Cloud Volumes Service가 지원하는 볼륨을 가져오려면 해당 볼륨 경로를 기준으로 볼륨을 식별합니다. 볼륨 경로는 이후 볼륨 내보내기 경로의 일부입니다 `:/`. 예를 들어, 내보내기 경로가 인 경우 `10.0.0.1:/adroit-jolly-swift`, 볼륨 경로는 입니다 `adroit-jolly-swift`.

Google Cloud Platform의 예

다음 예제에서는 을 가져옵니다 `gcp-cvs` 백엔드의 볼륨 `gcpcvs_YEppr` 볼륨 경로 포함 `adroit-jolly-swift`.

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

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

Azure NetApp Files

Astra Trident는 를 사용하여 볼륨 가져오기를 지원합니다 `azure-netapp-files` 드라이버.

> Azure NetApp Files 볼륨을 가져오려면 해당 볼륨 경로를 기준으로 볼륨을 식별합니다. 볼륨 경로는 이후 볼륨 내보내기 경로의 일부입니다 `:/`. 예를 들어, 마운트 경로가 인 경우 `10.0.0.2:/importvol1`, 볼륨 경로는 입니다 `importvol1`.

Azure NetApp Files의 예

다음 예제에서는 을 가져옵니다 `azure-netapp-files` 백엔드의 볼륨 `azurenetappfiles_40517` 볼륨 경로 포함 `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    |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
```

네임스페이스 전체에서 **NFS** 볼륨을 공유합니다

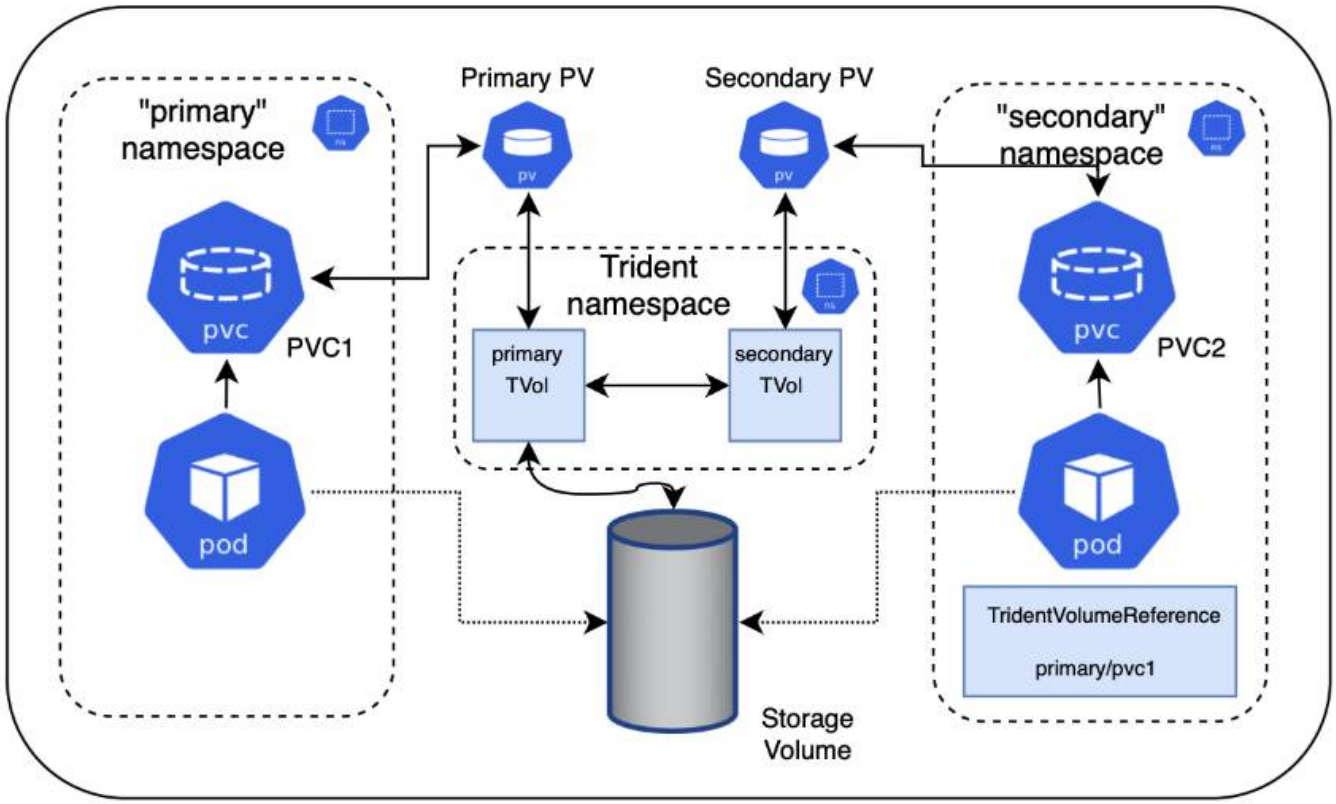
Astra Trident를 사용하면 기본 네임스페이스에서 볼륨을 생성하고 하나 이상의 보조 네임스페이스에서 공유할 수 있습니다.

피처

Astra 트리펜볼륨 레퍼런스 CR을 사용하면 하나 이상의 Kubernetes 네임스페이스에서 ReadWriteMany(rwx) NFS 볼륨을 안전하게 공유할 수 있습니다. 이 Kubernetes 네이티브 솔루션은 다음과 같은 이점을 제공합니다.

- 보안을 보장하기 위한 다양한 수준의 액세스 제어
- 모든 Trident NFS 볼륨 드라이버와 호환됩니다
- tridentctl 또는 기타 기본 Kubernetes 기능이 아닌 기능에 의존하지 않습니다

이 다이어그램은 2개의 Kubernetes 네임스페이스에서 NFS 볼륨 공유를 보여 줍니다.



빠른 시작

몇 단계만으로 NFS 볼륨 공유를 설정할 수 있습니다.

1 볼륨을 공유하도록 소스 **PVC**를 구성합니다

소스 네임스페이스 소유자는 소스 PVC의 데이터에 액세스할 수 있는 권한을 부여합니다.

2 대상 네임스페이스에서 **CR**을 만들 수 있는 권한을 부여합니다

클러스터 관리자는 대상 네임스페이스의 소유자에게 트리엔VolumeReference CR을 생성할 수 있는 권한을 부여합니다.

3 대상 네임스페이스에서 트리젠**VolumeReference** 를 생성합니다

대상 네임스페이스의 소유자는 소스 PVC를 참조하기 위해 트리엔VolumeReference CR을 생성합니다.

4 대상 네임스페이스에서 하위 **PVC**를 만듭니다

대상 네임스페이스의 소유자는 원본 PVC의 데이터 소스를 사용하기 위해 하위 PVC를 만듭니다.

소스 및 대상 네임스페이스를 구성합니다

보안을 보장하기 위해 네임스페이스 간 공유는 소스 네임스페이스 소유자, 클러스터 관리자 및 대상 네임스페이스 소유자의 협업 및 조치가 필요합니다. 사용자 역할은 각 단계에서 지정됩니다.

단계

1. * 원본 네임스페이스 소유자: * PVC를 만듭니다 (`pvc1`)를 대상 네임스페이스와 공유할 수 있는 권한을 부여하는 소스 네임스페이스의 경우 (`namespace2`)를 사용합니다 `shareToNamespace` 주석.

```
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
```

Astra Trident가 PV 및 백엔드 NFS 스토리지 볼륨을 생성합니다.

- 쉼표로 구분된 목록을 사용하여 PVC를 여러 네임스페이스에 공유할 수 있습니다. 예를 들면, 다음과 같습니다. `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`.
- 를 사용하여 모든 네임스페이스에 공유할 수 있습니다 `*`. 예를 들면, 다음과 같습니다. `trident.netapp.io/shareToNamespace: *`
- PVC를 업데이트하여 를 포함할 수 있습니다 `shareToNamespace` 언제든지 주석을 추가할 수 있습니다.

2. * 클러스터 관리자: * 대상 네임스페이스 소유자에게 대상 네임스페이스에서 트리젠VolumeReference CR을 생성할 수 있는 권한을 부여하기 위해 사용자 지정 역할을 생성하고 kubecon무화하십시오.
3. * 대상 네임스페이스 소유자: * 소스 네임스페이스를 참조하는 대상 네임스페이스에서 트리젠VolumeReference CR을 만듭니다 `pvc1`.

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

4. * 대상 네임스페이스 소유자: * PVC를 만듭니다 (`pvc2`)를 대상 네임스페이스에서 사용합니다 (`namespace2`)를 사용합니다 `shareFromPVC` 원본 PVC를 지정하는 주석.

```
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
```

대상 PVC의 크기는 소스 PVC보다 작거나 같아야 합니다.

결과

Astra Trident가 을 읽습니다 `shareFromPVC` 대상 PVC에 주석을 추가하여 대상 PV를 원본 PV를 가리키는 자체 스토리지 리소스가 없는 하위 볼륨으로 생성하고 소스 PV 스토리지 리소스를 공유합니다. 대상 PVC와 PV가 정상으로 표시됩니다.

공유 볼륨을 삭제합니다

여러 네임스페이스에서 공유되는 볼륨을 삭제할 수 있습니다. Astra Trident는 소스 네임스페이스에서 볼륨에 대한 액세스를 제거하고 볼륨을 공유하는 다른 네임스페이스에 대한 액세스를 유지 관리합니다. 볼륨을 참조하는 모든 네임스페이스가 제거되면 Astra Trident가 볼륨을 삭제합니다.

사용 `tridentctl get` 하위 볼륨을 쿼리합니다

를 사용합니다[`tridentctl` 유틸리티, 를 실행할 수 있습니다 `get` 하위 볼륨을 가져오는 명령입니다. 자세한 내용은 다음 링크를 참조하십시오…/triment-reference/tridentctl.html[`tridentctl` 명령 및 옵션].

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

플래그:

- `-h, --help`: 볼륨에 대한 도움말입니다.
- `--parentOfSubordinate string`: 하위 원본 볼륨으로 쿼리를 제한합니다.
- `--subordinateOf string`: 볼륨 부하로 쿼리 제한.

제한 사항

- Astra Trident는 대상 네임스페이스가 공유 볼륨에 쓰는 것을 막을 수 없습니다. 파일 잠금 또는 기타 프로세스를 사용하여 공유 볼륨 데이터를 덮어쓰지 않도록 해야 합니다.
- 를 제거하여 원본 PVC에 대한 액세스를 취소할 수 없습니다 `shareToNamespace` 또는 `shareFromNamespace` 주석 또는 삭제 `TridentVolumeReference` 있습니다. 액세스 권한을 취소하려면 하위 PVC를 삭제해야 합니다.
- 하위 볼륨에서는 스냅샷, 클론 및 미러링을 사용할 수 없습니다.

를 참조하십시오

네임스페이스 간 볼륨 액세스에 대한 자세한 내용은 다음을 참조하십시오.

- 를 방문하십시오 "네임스페이스 간 볼륨 공유: 네임스페이스 간 볼륨 액세스를 위해 hello를 사용합니다".
- 데모를 시청해보시기 바랍니다 "NetAppTV를 참조하십시오".

CSI 토폴로지를 사용합니다

Astra Trident는 을 사용하여 Kubernetes 클러스터에 있는 노드를 선택적으로 생성하여 연결할 수 있습니다 "CSI 토폴로지 기능".

개요

CSI 토폴로지 기능을 사용하면 지역 및 가용성 영역에 따라 볼륨에 대한 액세스가 노드의 하위 집합으로 제한될 수 있습니다. 오늘날의 클라우드 공급자는 Kubernetes 관리자가 영역 기반의 노드를 생성할 수 있습니다. 노드는 지역 내 또는 여러 지역의 여러 가용성 영역에 위치할 수 있습니다. Astra Trident는 다중 영역 아키텍처에서 워크로드용 볼륨 프로비저닝을 지원하기 위해 CSI 토폴로지를 사용합니다.

CSI 토폴로지 기능에 대해 자세히 알아보십시오 "여기".

Kubernetes는 두 가지 고유한 볼륨 바인딩 모드를 제공합니다.

- 와 함께 `VolumeBindingMode` 를 로 설정합니다 `Immediate`, Astra Trident는 토폴로지 인식 없이 볼륨을 생성합니다. 볼륨 바인딩 및 동적 프로비저닝은 PVC가 생성될 때 처리됩니다. 이것이 기본값입니다 `VolumeBindingMode` 또한 토폴로지 제약 조건을 적용하지 않는 클러스터에 적합합니다. 영구 볼륨은 요청 Pod의 예약 요구사항에 종속되지 않고 생성됩니다.
- 와 함께 `VolumeBindingMode` 를 로 설정합니다 `WaitForFirstConsumer`PVC에 대한 영구 볼륨의 생성 및 바인딩은 PVC를 사용하는 POD가 예약 및 생성될 때까지 지연됩니다. 이렇게 하면 토폴로지 요구 사항에 따라 적용되는 일정 제한을 충족하기 위해 볼륨이 생성됩니다.

를 클릭합니다 `WaitForFirstConsumer` 바인딩 모드에서는 토폴로지 레이블이 필요하지 않습니다. 이 기능은 CSI 토폴로지 기능과 독립적으로 사용할 수 있습니다.

필요한 것

CSI 토폴로지를 사용하려면 다음이 필요합니다.

- 를 실행하는 Kubernetes 클러스터 "지원되는 Kubernetes 버전"

```
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"}
```

- 클러스터의 노드에는 토폴로지를 인식하는 레이블이 있어야 합니다 (`topology.kubernetes.io/region` 및 `topology.kubernetes.io/zone`)를 클릭합니다. Astra Trident가 토폴로지 인식을 위해 설치되기 전에 클러스터의 노드에 이러한 레이블 * 이 있어야 합니다.

```
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"}]
```

1단계: 토폴로지 인식 백엔드 생성

Astra Trident 스토리지 백엔드는 가용성 영역에 따라 볼륨을 선택적으로 프로비저닝하도록 설계할 수 있습니다. 각 백엔드에는 선택 사항이 포함될 수 있습니다 `supportedTopologies` 지원해야 하는 영역 및 영역 목록을 나타내는 블록입니다. 이러한 백엔드를 사용하는 StorageClasses의 경우 지원되는 영역/영역에서 예약된 애플리케이션에서 요청하는 경우에만 볼륨이 생성됩니다.

다음은 백엔드 정의의 예입니다.

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
```

JSON을 참조하십시오

```
{
 "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` 백엔드 당 지역 및 영역 목록을 제공하는 데 사용됩니다. 이러한 영역 및 영역은 StorageClass 에서 제공할 수 있는 허용 가능한 값의 목록을 나타냅니다. 백엔드에서 제공되는 지역 및 영역의 하위 집합이 포함된 StorageClasses의 경우 Astra Trident는 백엔드에 볼륨을 생성합니다.

을 정의할 수 있습니다 `supportedTopologies` 스토리지 풀당 다음 예를 참조하십시오.

```
---
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
  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
```

이 예에서 는 입니다 `region` 및 `zone` 레이블은 스토리지 풀의 위치를 나타냅니다. `topology.kubernetes.io/region` 및 `topology.kubernetes.io/zone` 스토리지 풀을 사용할 수 있는 위치를 지정합니다.

2단계: 토폴로지를 인식하는 StorageClasses를 정의합니다

클러스터의 노드에 제공되는 토폴로지 레이블을 기반으로 StorageClasses를 정의하여 토폴로지 정보를 포함할 수 있습니다. 이렇게 하면 PVC 요청에 대한 후보 역할을 하는 스토리지 풀과 Trident에서 제공하는 볼륨을 사용할 수 있는 노드의 하위 세트가 결정됩니다.

다음 예를 참조하십시오.

```
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"
```

위에서 제공한 StorageClass 정의에서 `volumeBindingMode` 가 로 설정되어 있습니다 `WaitForFirstConsumer`. 이 StorageClass에 요청된 PVC는 POD에서 참조될 때까지 작동하지 않습니다. 그리고, `allowedTopologies` 사용할 영역 및 영역을 제공합니다. 를 클릭합니다 `netapp-san-us-east1` StorageClass가 에 PVC를 생성합니다 `san-backend-us-east1` 백엔드가 위에서 정의되었습니다.

3단계: PVC 생성 및 사용

StorageClass가 생성되어 백엔드에 매핑되면 PVC를 생성할 수 있습니다.

예를 참조하십시오 `spec` 아래:

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

이 매니페스트를 사용하여 PVC를 만들면 다음과 같은 결과가 발생합니다.

```
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
```

Trident에서 볼륨을 생성하여 PVC에 바인딩하려면 POD에서 PVC를 사용합니다. 다음 예를 참조하십시오.

```
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
```

이 podSpec은 에 있는 노드에서 Pod를 예약하도록 Kubernetes에 지시합니다 `us-east1` 영역 을 클릭하고 에 있는 노드 중에서 선택합니다 `us-east1-a` 또는 `us-east1-b` 존.

다음 출력을 참조하십시오.

```
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
```

포함할 백엔드를 업데이트합니다 `supportedTopologies`

기존 백엔드는 목록을 포함하도록 업데이트할 수 있습니다 `supportedTopologies` 사용 `tridentctl backend update`. 이는 이미 프로비저닝된 체적에 영향을 주지 않으며 후속 PVC에만 사용됩니다.

자세한 내용을 확인하십시오

- "컨테이너에 대한 리소스를 관리합니다"
- "노드 선택기"
- "친화성 및 반친화성"
- "오염과 내약입니다"

스냅샷 작업

영구 볼륨(PVS)의 Kubernetes 볼륨 스냅샷은 볼륨의 시점 복사본을 지원합니다. Astra Trident를 사용하여 생성된 볼륨의 스냅샷을 생성하고, Astra Trident 외부에서 생성된 스냅샷을 가져오고, 기존 스냅샷에서 새 볼륨을 생성하고, 스냅샷에서 볼륨 데이터를 복구할 수 있습니다.

개요

에서 볼륨 스냅샷을 지원합니다 `ontap-nas`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`, `solidfire-san`, `gcp-cvs`, 및 `azure-netapp-files` 드라이버.

시작하기 전에

스냅샷을 사용하려면 외부 스냅샷 컨트롤러와 CRD(사용자 정의 리소스 정의)가 있어야 합니다. Kubernetes Orchestrator의 책임입니다(예: Kubeadm, GKE, OpenShift).

Kubernetes 배포 시 스냅샷 컨트롤러 및 CRD가 포함되지 않은 경우 를 참조하십시오 볼륨 스냅샷 컨트롤러를 배포합니다.

> GKE 환경에서 필요 시 볼륨 스냅샷을 생성할 경우 스냅샷 컨트롤러를 생성하지 마십시오. GKE는 내장된 숨겨진 스냅샷 컨트롤러를 사용합니다.

볼륨 스냅샷을 생성합니다

단계

1. 을 생성합니다 `VolumeSnapshotClass`. 자세한 내용은 을 참조하십시오 "VolumeSnapshotClass".
   - 를 클릭합니다 `driver` Astra Trident CSI 드라이버를 가리킵니다.
   - `deletionPolicy` 있을 수 있습니다 `Delete` 또는 `Retain`. 를 로 설정한 경우 `Retain`, 스토리지 클러스터의 기본 물리적 스냅샷은 가 있는 경우에도 유지됩니다 `VolumeSnapshot` 객체가 삭제되었습니다.

   예

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

2. 기존 PVC의 스냅샷을 생성합니다.

   예
   - 이 예에서는 기존 PVC의 스냅샷을 생성합니다.

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

   - 이 예에서는 라는 PVC에 대한 볼륨 스냅샷 객체를 생성합니다 `pvc1` 스냅샷 이름이 로 설정되어 있습니다 `pvc1-snap`. VolumeSnapshot은 PVC와 유사하며 와 관련이 있습니다 `VolumeSnapshotContent` 실제 스냅샷을 나타내는 객체입니다.

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

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

◦ 를 식별할 수 있습니다 `VolumeSnapshotContent` 의 개체 `pvc1-snap` VolumeSnapshot을 설명합니다. 를 클릭합니다 `Snapshot Content Name` 이 스냅샷을 제공하는 VolumeSnapshotContent 객체를 식별합니다. 를 클릭합니다 `Ready To Use` 매개 변수는 스냅샷을 사용하여 새 PVC를 생성할 수 있음을 나타냅니다.

```
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
.
.
```

볼륨 스냅샷에서 **PVC**를 생성합니다

을 사용할 수 있습니다 `dataSource` 이름이 인 VolumeSnapshot을 사용하여 PVC를 생성합니다 `<pvc-name>` 데이터 소스로 사용됩니다. PVC가 생성된 후 POD에 부착하여 다른 PVC와 마찬가지로 사용할 수 있습니다.

PVC는 소스 볼륨과 동일한 백엔드에서 생성됩니다. 을 참조하십시오 "KB: Trident PVC 스냅샷에서 PVC를 생성하는 것은 대체 백엔드에서 생성할 수 없습니다".

다음 예에서는 를 사용하여 PVC를 작성합니다 `pvc1-snap` 를 데이터 소스로 사용합니다.

```
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
```

볼륨 스냅샷을 가져옵니다

Astra Trident가 를 지원합니다 "Kubernetes 사전 프로비저닝된 스냅샷 프로세스" 클러스터 관리자가 을(를) 생성할 수 있도록 하려면 `VolumeSnapshotContent` Astra Trident 외부에 생성된 개체 및 스냅샷 가져오기

시작하기 전에

Astra Trident가 스냅샷의 상위 볼륨을 생성하거나 가져와야 합니다.

단계

1. * 클러스터 관리자: * 를 생성합니다 `VolumeSnapshotContent` 백엔드 스냅샷을 참조하는 객체입니다. 그러면 Astra Trident에서 스냅샷 워크플로우가 시작됩니다.
   - 에서 백엔드 스냅샷의 이름을 지정합니다 `annotations` 현재 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`.
   - 를 지정합니다 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` 인치 `snapshotHandle`. 이 정보는 의 외부 스냅샷 생성자가 Astra Trident에 제공하는 유일한 정보입니다 `ListSnapshots` 통화.

     > ⓘ 를 클릭합니다 `<volumeSnapshotContentName>` CR 명명 제한으로 인해 백엔드 스냅샷 이름과 항상 일치할 수 없습니다.

     예

     다음 예제에서는 을 만듭니다 `VolumeSnapshotContent` 백엔드 스냅샷을 참조하는 객체입니다 `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>
```

2. * 클러스터 관리자: * 를 생성합니다 `VolumeSnapshot` 을 참조하는 CR `VolumeSnapshotContent` 오브젝트. 그러면 를 사용할 수 있는 액세스가 필요합니다 `VolumeSnapshot` 지정된 네임스페이스에서.

   예

   다음 예제에서는 을 만듭니다 `VolumeSnapshot` CR 이름 `import-snap` 을 참조합니다 `VolumeSnapshotContent` 이름 지정 `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. * 내부 처리 (아무 조치 필요 없음) : * 외부 snapshotter가 새로 만든 것을 인식합니다 `VolumeSnapshotContent` 를 실행합니다 `ListSnapshots` 통화. Astra Trident가 을 생성합니다 `TridentSnapshot`.
   - 외부 스냅숏 작성기가 를 설정합니다 `VolumeSnapshotContent` 를 선택합니다 `readyToUse` 및 `VolumeSnapshot` 를 선택합니다 `true`.
   - Trident가 돌아왔습니다 `readyToUse=true`.
4. * 모든 사용자: * 를 생성합니다 `PersistentVolumeClaim` 를 눌러 새 를 참조합니다 `VolumeSnapshot`, 위치 `spec.dataSource` (또는 `spec.dataSourceRef`) name 은 입니다 `VolumeSnapshot` 이름.

   예

   다음 예에서는 를 참조하는 PVC를 작성합니다 `VolumeSnapshot` 이름 지정 `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
```

스냅샷을 사용하여 볼륨 데이터를 복구합니다

스냅샷 디렉토리는 를 사용하여 프로비저닝된 볼륨의 최대 호환성을 지원하기 위해 기본적으로 숨겨져 있습니다 `ontap-nas` 및 `ontap-nas-economy` 드라이버. 를 활성화합니다 `.snapshot` 스냅샷으로부터 직접 데이터를 복구할 디렉토리입니다.

볼륨 스냅샷 복원 ONTAP CLI를 사용하여 볼륨을 이전 스냅숏에 기록된 상태로 복원합니다.

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

스냅샷 복사본을 복원하면 기존 볼륨 구성이 덮어쓰여집니다. 스냅샷 복사본이 생성된 후 볼륨 데이터에 대한 변경 사항은 손실됩니다.

연결된 스냅샷이 있는 **PV**를 삭제합니다

연결된 스냅샷이 있는 영구 볼륨을 삭제하면 해당 Trident 볼륨이 "삭제 상태"로 업데이트됩니다. Astra Trident 볼륨을 삭제하려면 볼륨 스냅샷을 제거하십시오.

볼륨 스냅샷 컨트롤러를 배포합니다

Kubernetes 배포 시 스냅샷 컨트롤러와 CRD가 포함되지 않은 경우 다음과 같이 배포할 수 있습니다.

단계

1. 볼륨 스냅샷 CRD를 생성합니다.

```
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. 스냅샷 컨트롤러를 생성합니다.

```
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
```

> 필요한 경우 를 엽니다 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` 및 업데이트 `namespace` 네임스페이스로.

관련 링크

- "볼륨 스냅숏"
- "VolumeSnapshotClass"