



Astra Trident 23.01.1 설명서

Astra Trident

NetApp
April 16, 2024

목차

Astra Trident 23.01.1 설명서	1
릴리즈 노트	2
23.01.1의 새로운 기능	2
23.01의 변경 사항	2
22.10의 변경 사항	3
22.07 변경	4
22.04의 변경	5
22.01.1의 변경 사항	6
22.01.0의 변경 사항	6
21.10.1 변경	6
21.10.0 변경	7
알려진 문제	8
자세한 내용을 확인하십시오	9
개념	10
Astra Trident에 대해 자세히 알아보십시오	10
ONTAP 드라이버	11
프로비저닝	12
볼륨 스냅샷	13
가상 풀	13
볼륨 액세스 그룹	15
시작하십시오	16
시도해 보십시오	16
요구 사항	16
Astra Trident를 설치합니다	21
다음 단계	52
Astra Trident 관리	57
Astra Trident를 업그레이드합니다	57
Astra Trident를 제거합니다	70
Astra Trident를 다운그레이드하십시오	72
Astra Trident를 사용해 보십시오	76
작업자 노드를 준비합니다	76
백엔드 구성	80
kubectrl로 백엔드를 만듭니다	169
kubectrl을 사용하여 백엔드 관리 수행	176
tridentctl을 사용하여 백엔드 관리를 수행합니다	177
백엔드 관리 옵션 간 이동	179
스토리지 클래스를 관리합니다	185
볼륨 작업을 수행합니다	187
네임스페이스 전체에서 NFS 볼륨을 공유합니다	212

Astra Trident를 모니터링합니다	215
Docker를 위한 Astra Trident	220
배포를 위한 사전 요구 사항	220
Astra Trident 구축	223
Astra Trident를 업그레이드하거나 제거합니다	227
볼륨 작업	229
로그를 수집합니다	237
여러 Astra Trident 인스턴스를 관리합니다	238
스토리지 구성 옵션	239
알려진 문제 및 제한 사항	247
자주 묻는 질문	249
일반적인 질문	249
Kubernetes 클러스터에 Astra Trident를 설치 및 사용합니다	249
문제 해결 및 지원	251
Astra Trident를 업그레이드합니다	252
백엔드 및 볼륨 관리	252
지원	257
문제 해결	258
일반 문제 해결	258
연산자를 사용하여 실패한 Trident 배포 문제 해결	259
를 사용하여 Trident 배포가 성공하지 못한 경우 문제 해결 <code>tridentctl</code>	261
모범 사례 및 권장사항	262
구축	262
스토리지 구성	262
Astra Trident 통합	268
데이터 보호	278
보안	283
참조하십시오	289
Astra Trident 포트	289
Astra Trident REST API	289
명령줄 옵션	290
Kubernetes와 통합된 NetApp 제품	291
Kubernetes 및 Trident 오브젝트	292
tridentctl 명령 및 옵션	303
POD 보안 표준(PSS) 및 보안 컨텍스트 제약(SCC)	308
이전 버전의 문서	314
법적 고지	315
저작권	315
상표	315
특허	315
개인 정보 보호 정책	315

Astra Trident 23.01.1 설명서

릴리즈 노트

릴리즈 노트 최신 버전의 Astra Trident에 새로운 기능, 개선 사항 및 버그 수정 사항에 대한 정보를 제공합니다.



를 클릭합니다 tridentctl 설치 프로그램 zip 파일에 제공된 Linux용 바이너리는 테스트되고 지원되는 버전입니다. 에 유의하십시오 macos 에 제공된 바이너리 /extras zip 파일의 일부가 테스트되지 않았거나 지원되지 않습니다.

23.01.1의 새로운 기능

수정

- SPEC에 지정된 경우 설치에 IPv6 localhost를 사용하도록 고정 Trident Operator가 수정되었습니다.
- 번들 권한과 동기화되는 고정 Trident 운영자 클러스터 역할 권한 "[문제 #799](#)".
- 외부 프로세스가 완료될 때까지 실행되도록 하는 수정 사항이 추가되었습니다.
- rwx 모드에서 여러 노드에 원시 블록 볼륨을 연결하는 문제 해결
- SMB 볼륨에 대한 FlexGroup 클론 복제 지원 및 볼륨 가져오기 수정

23.01의 변경 사항



이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Astra Trident를 업그레이드하십시오.

수정

- Kubernetes: Helm ("[문제 #783](#), [#794](#)")를 클릭합니다.

향상된 기능

쿠버네티스

- Kubernetes 1.26에 대한 지원이 추가되었습니다.
- 전체 Trident RBAC 리소스 활용률 향상 ("[문제 #757](#)")를 클릭합니다.
- 호스트 노드에서 손상되거나 유효하지 않은 iSCSI 세션을 감지하고 수정하는 자동화 기능 추가
- LUKS 암호화 볼륨 확장을 위한 지원이 추가되었습니다.
- Kubernetes: LUKS 암호화 볼륨에 대한 자격 증명 회전 지원 추가.

아스트라 트리덴트

- ONTAP용 Amazon FSx를 사용하는 SMB 볼륨에 대한 지원을 ONTAP-NAS 스토리지 드라이버에 추가했습니다.
- SMB 볼륨을 사용할 때 NTFS 권한에 대한 지원이 추가되었습니다.
- CVS 서비스 수준이 있는 GCP 볼륨에 대한 스토리지 풀에 대한 지원이 추가되었습니다.

- ONTAP-NAS-flexgroup 스토리지 드라이버를 사용하여 FlexGroups를 생성할 때 flexgroupAggregateList의 선택적 사용에 대한 지원이 추가되었습니다.
- 여러 개의 FlexVols를 관리할 때 ONTAP-NAS-이코노미 스토리지 드라이버의 성능이 향상되었습니다.
- 모든 ONTAP NAS 스토리지 드라이버에 대해 데이터 LIF 업데이트를 사용하도록 설정했습니다.
- 호스트 노드 OS를 반영하도록 Trident 배포 및 DemonSet 명명 규칙을 업데이트했습니다.

사용 중단

- Kubernetes: 지원되는 최소 Kubernetes를 1.21로 업데이트했습니다.
- 구성 시 데이터 LIF를 더 이상 지정하지 않아야 합니다 `ontap-san` 또는 `ontap-san-economy` 드라이버.

22.10의 변경 사항

- Astra Trident 22.10으로 업그레이드하기 전에 다음 중요 정보를 읽어야 합니다. *

>Astra Trident 22.10에 대한 중요 정보

- 이제 Trident에서 Kubernetes 1.25가 지원됩니다. Kubernetes 1.25로 업그레이드하기 전에 Astra Trident를 22.10으로 업그레이드해야 합니다.
- Astra Trident는 이제 SAN 환경에서 다중 경로 구성을 사용하도록 엄격히 적용되며 권장값은 `find_multipaths: no` 다중 경로 .conf 파일



비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart multipath.conf` 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다 `find_multipaths: no` 21.07 릴리스 이후.

수정

- 을 사용하여 생성된 ONTAP 백엔드에 대한 특정 문제를 해결했습니다 `credentials` 22.07.0 업그레이드 중 온라인 상태로 전환되지 않는 필드("문제 #759")를 클릭합니다.
- Docker: 일부 환경에서 Docker 볼륨 플러그인을 시작하지 못하는 문제 해결("문제 #548" 및 "문제 # 760")를 클릭합니다.
- 보고 노드에 속하는 데이터 LIF의 일부만 게시되도록 ONTAP SAN 백엔드에 특정한 SLM 문제를 수정했습니다.
- 볼륨을 연결할 때 iSCSI LUN에 대한 불필요한 검사가 발생하는 성능 문제를 해결했습니다.
- Astra Trident iSCSI 워크플로우 내에서 세분화된 재시도 횟수를 제거하여 빠르게 실패하고 외부 재시도 간격을 줄일 수 있습니다.
- 해당 다중 경로 장치가 이미 플래시되었을 때 iSCSI 장치를 플래싱할 때 오류가 반환되는 문제를 해결했습니다.

향상된 기능

- 쿠버네티스:
 - Kubernetes 1.25 지원 추가. Kubernetes 1.25로 업그레이드하기 전에 Astra Trident를 22.10으로 업그레이드해야 합니다.
 - Trident Deployment 및 DemonSet에 대해 별도의 ServiceAccount, ClusterRole 및 ClusterRoleBinding을

추가하여 이후의 사용 권한 개선을 허용합니다.

- 에 대한 지원이 추가되었습니다 "[네임스페이스 간 볼륨 공유](#)".
- 모든 Trident `ontap-*` 이제 스토리지 드라이버가 ONTAP REST API에서 작동합니다.
- 새로운 운영자 YAML 추가 (`bundle_post_1_25.yaml`)를 사용하지 않을 경우 PodSecurityPolicy Kubernetes 1.25를 지원합니다.
- 추가되었습니다 "[LUKS 암호화 볼륨 지원](#)" 용 `ontap-san` 및 `ontap-san-economy` 스토리지 드라이버.
- Windows Server 2019 노드에 대한 지원이 추가되었습니다.
- 추가되었습니다 "[Windows 노드에서 SMB 볼륨 지원](#)" 를 통해 `azure-netapp-files` 스토리지 드라이버.
- 이제 ONTAP 드라이버에 대한 자동 MetroCluster 전환 감지 기능을 사용할 수 있습니다.

사용 중단

- **Kubernetes:** 최소 지원 Kubernetes를 1.20으로 업데이트했습니다.
- ADS(Astra Data Store) 드라이버를 제거했습니다.
- 에 대한 지원이 제거되었습니다 `yes` 및 `smart` 에 대한 옵션 `find_multipaths` iSCSI에 대한 작업자 노드 경로 다중화를 구성하는 경우

22.07 변경

수정

- Kubernetes**
 - Hrom 또는 Trident 연산자를 사용하여 Trident를 구성할 때 노드 선택기에 대한 부울 및 숫자 값을 처리하는 문제가 해결되었습니다. ("[GitHub 문제 #700](#)")
 - CHAP가 아닌 경로에서 발생하는 오류를 처리하는 문제를 수정함으로써 kubelet이 실패한 경우 다시 시도합니다. ("[GitHub 문제 #736](#)")

향상된 기능

- `k8s.gcr.io`에서 `registry.k8s.io`로 CSI 이미지의 기본 레지스트리로 전환합니다
- ONTAP-SAN 볼륨은 이제 노드별 `igroup`을 사용하며 해당 노드에 능동적으로 게시되는 LUN만 `igroup`에 매핑하여 보안 상태를 향상합니다. Trident가 활성 워크로드에 영향을 주지 않고 안전하게 수행할 수 있다고 판단하면 기존 볼륨이 새로운 `igroup` 체계로 자동으로 전환됩니다.
- Trident 설치에 리소스 할당량을 포함함으로써 PriorityClass 소비가 기본적으로 제한될 때 Trident DemonSet이 예약되도록 합니다.
- ANF 드라이버에 네트워크 기능에 대한 지원이 추가되었습니다. ("[GitHub 문제 #717](#)")
- ONTAP 드라이버에 기술 미리 보기 자동 MetroCluster 전환 감지 기능이 추가되었습니다. ("[GitHub 문제 #228](#)")

사용 중단

- **Kubernetes:** 최소 지원 Kubernetes를 1.19로 업데이트했습니다.
- 백엔드 구성은 더 이상 단일 구성에서 여러 인증 유형을 사용할 수 없습니다.

제거

- AWS CVS 드라이버(22.04 이후 더 이상 사용되지 않음)가 제거되었습니다.
- 쿠버네티스
 - 노드 포드에 불필요한 SYS_ADMIN 기능이 제거되었습니다.
 - 노드 준비 작업을 간단한 호스트 정보로 줄이고 활성 서비스 검색을 통해 작업 노드에서 NFS/iSCSI 서비스를 사용할 수 있다는 최선의 확인 작업을 수행할 수 있습니다.

문서화

새로운 소식 "[POD 보안 표준](#)" (PSS) 섹션에 설치 시 Astra Trident에서 활성화한 상세 권한이 추가되었습니다.

22.04의 변경

NetApp은 제품과 서비스를 지속적으로 개선 및 개선하고 있습니다. Astra Trident의 최신 기능 몇 가지를 소개합니다. 이전 릴리스는 를 참조하십시오 "[이전 버전의 문서](#)".



이전 Trident 릴리즈에서 업그레이드하고 Azure NetApp Files를 사용하는 경우, 를 참조하십시오 `location` 구성 매개 변수는 이제 필수 싱글톤 필드입니다.

수정

- iSCSI 이니시에이터 이름의 구문 분석 기능이 향상되었습니다. ("[GitHub 문제 #681](#)")
- CSI 스토리지 클래스 매개 변수가 허용되지 않는 문제를 해결했습니다. ("[GitHub 문제 #598](#)")
- Trident CRD에서 중복 키 선언을 수정했습니다. ("[GitHub 문제 #671](#)")
- 부정확한 CSI 스냅샷 로그를 수정했습니다. ("[GitHub 문제 #629](#)")
- 삭제된 노드에서 볼륨 계시를 취소하는 문제 해결 ("[GitHub 문제 #691](#)")
- 블록 디바이스에서 파일 시스템 불일치를 처리하는 기능이 추가되었습니다. ("[GitHub 문제 #656](#)")
- 를 설정할 때 자동 지원 이미지를 가져오는 문제가 해결되었습니다 imageRegistry 설치 중 플래그. ("[GitHub 문제 #715](#)")
- ANF 드라이버가 여러 내보내기 규칙을 사용하여 볼륨을 클론하지 못한 문제 해결

향상된 기능

- 이제 Trident의 보안 끝점에 대한 인바운드 연결에는 TLS 1.3 이상이 필요합니다. ("[GitHub 문제 #698](#)")
- 이제 Trident는 보안 엔드포인트의 응답에 HSTS 헤더를 추가합니다.
- 이제 Trident는 Azure NetApp Files UNIX 사용 권한 기능을 자동으로 활성화하려고 시도합니다.
- * Kubernetes *: Trident가 이제 시스템 노드 크리티컬 우선 순위 클래스에서 실행됩니다. ("[GitHub 문제 #694](#)")

제거

E-Series 드라이버(20.07 이후 비활성화됨)가 제거되었습니다.

22.01.1의 변경 사항

수정

- 삭제된 노드에서 볼륨 게시를 취소하는 문제 해결 ("[GitHub 문제 #691](#)")
- ONTAP API 응답에서 공간 집계에 대한 nil 필드에 액세스할 때 패닉이 수정되었습니다.

22.01.0의 변경 사항

수정

- * Kubernetes: * 대규모 클러스터의 노드 등록 백오프 재시도 시간을 늘립니다.
- 동일한 이름의 여러 리소스가 Azure-NetApp-files 드라이버를 혼동할 수 있는 문제 해결
- ONTAP SAN IPv6 데이터 LIF는 이제 대괄호와 함께 지정된 경우 작동합니다.
- 이미 가져온 볼륨을 가져오려고 하면 PVC가 보류 상태로 남겨둔 EOF가 반환되는 문제가 해결되었습니다. ("[GitHub 문제 #489](#)")
- SolidFire 볼륨에 32개 이상의 스냅샷을 생성할 때 Astra Trident 성능이 느려지는 문제를 해결했습니다.
- SHA-1을 SSL 인증서 생성에서 SHA-256으로 교체했습니다.
- 중복된 리소스 이름을 허용하고 작업을 단일 위치로 제한하도록 ANF 드라이버를 수정했습니다.
- 중복된 리소스 이름을 허용하고 작업을 단일 위치로 제한하도록 ANF 드라이버를 수정했습니다.

향상된 기능

- Kubernetes의 향상된 기능:
 - Kubernetes 1.23 지원 추가.
 - Trident Operator 또는 Hrom을 통해 설치된 Trident Pod에 대한 예약 옵션을 추가합니다. ("[GitHub 문제 #651](#)")
- GCP 드라이버에서 지역 간 볼륨을 허용합니다. ("[GitHub 문제 #633](#)")
- ANF 볼륨에 'unixPermissions' 옵션에 대한 지원이 추가되었습니다. ("[GitHub 문제 #666](#)")

사용 중단

Trident REST 인터페이스는 127.0.0.1 또는 [::1] 주소에서만 수신 및 제공할 수 있습니다

21.10.1 변경



v21.10.0 릴리즈에는 노드를 제거한 다음 Kubernetes 클러스터에 다시 추가할 때 Trident 컨트롤러를 CrashLoopBackOff 상태로 전환할 수 있는 문제가 있습니다. 이 문제는 v21.10.1([GitHub 문제 669](#))에서 해결되었습니다.

수정

- GCP CVS 백엔드에서 볼륨을 가져올 때 잠재적인 경쟁 조건이 수정되어 가져오지 못했습니다.
- 노드를 제거할 때 Trident 컨트롤러를 CrashLoopBackOff 상태로 전환할 수 있는 문제를 해결한 다음 Kubernetes 클러스터(GitHub 문제 669)에 다시 추가되었습니다.
- SVM 이름이 지정되지 않은 경우 SVM이 더 이상 검색되지 않는 문제 해결(GitHub 문제 612)

21.10.0 변경

수정

- XFS 볼륨의 클론을 소스 볼륨과 동일한 노드에 마운트할 수 없는 문제(GitHub 문제 514)가 해결되었습니다.
- Astra Trident에서 종료 시 심각한 오류를 기록한 문제 해결(GitHub 문제 597)
- Kubernetes 관련 수정 사항:
 - 에서 스냅샷을 생성할 때 볼륨의 사용된 공간을 최소 restoreSize로 반환합니다 `ontap-nas` 및 `ontap-nas-flexgroup` 드라이버(GitHub 문제 645).
 - 문제 해결 위치 `Failed to expand filesystem` 볼륨 크기 조정 후 오류가 기록되었습니다(GitHub 문제 560).
 - 포드가 끼일 수 있는 문제 해결 `Terminating State`(GitHub 문제 572):
 - 에서 케이스를 수정했습니다 `ontap-san-economy FlexVol`에 스냅샷 LUN이 가득 찼을 수 있습니다(GitHub 문제 533).
 - 다른 이미지의 사용자 지정 YAML 설치 프로그램 문제 해결(GitHub 문제 613)
 - 스냅샷 크기 계산 수정(GitHub 문제 611)
 - 모든 Astra Trident 설치 관리자가 일반 Kubernetes를 OpenShift로 식별할 수 있는 문제 해결(GitHub 문제 639)
 - Kubernetes API 서버에 연결할 수 없는 경우 조정을 중지하도록 Trident 연산자를 수정했습니다(GitHub 문제 599).

향상된 기능

- 에 대한 지원이 추가되었습니다 `unixPermissions` GCP-CVS 성능 볼륨 옵션.
- 600GiB~1TiB 범위의 GCP에서 확장성 최적화 CVS 볼륨 지원 추가
- Kubernetes 관련 개선사항:
 - Kubernetes 1.22에 대한 지원이 추가되었습니다.
 - Trident 운영자 및 제어 차트를 Kubernetes 1.22(GitHub 문제 628)와 함께 사용할 수 있도록 했습니다.
 - 에 운영자 이미지가 추가되었습니다 `tridentctl images` 명령(GitHub 문제 570).

실험적인 개선

- 에서 볼륨 복제에 대한 지원이 추가되었습니다 `ontap-san` 드라이버.
- 에 대한 * 기술 미리보기 * REST 지원 추가 `ontap-nas-flexgroup`, `ontap-san`, 및 `ontap-nas-economy`

드라이버.

알려진 문제

알려진 문제점은 제품을 성공적으로 사용하지 못하게 만들 수 있는 문제를 식별합니다.

- Astra Trident가 설치된 Kubernetes 클러스터를 1.24에서 1.25 이상으로 업그레이드할 경우 Values.YAML을 업데이트해야 합니다 `excludePodSecurityPolicy` 를 선택합니다 `true` 또는 을 추가합니다 `--set excludePodSecurityPolicy=true` 를 누릅니다 `helm upgrade` 명령을 먼저 실행한 후 클러스터를 업그레이드하십시오.
- Astra Trident가 이제 공란을 적용합니다 `fsType` (`fsType=""`)이 없는 볼륨의 경우 `fsType StorageClass`에 지정됩니다. Kubernetes 1.17 이상으로 작업할 경우 Trident는 공백을 제공할 수 있도록 지원합니다 `fsType NFS` 볼륨용. iSCSI 볼륨의 경우 를 설정해야 합니다 `fsType` 를 적용할 때 `StorageClass`에 대해 `fsGroup` 보안 컨텍스트 사용.
- 여러 Astra Trident 인스턴스에서 백엔드를 사용할 경우 각 백엔드 구성 파일은 서로 달라야 합니다 `storagePrefix` ONTAP 백엔드에 대한 값이거나 다른 값을 사용합니다 `TenantName` SolidFire 백엔드의 경우. Astra Trident는 Astra Trident의 다른 인스턴스가 생성한 볼륨을 감지할 수 없습니다. Astra Trident가 볼륨 생성을 idempotent 작업으로 처리하므로 ONTAP 또는 SolidFire 백엔드에서 기존 볼륨을 생성하려고 하면 성공합니다. If(경우 `storagePrefix` 또는 `TenantName` 서로 다를 수 없습니다. 동일한 백엔드에서 생성된 볼륨의 이름 충돌이 있을 수 있습니다.
- Astra Trident 설치 시(사용 `tridentctl` 또는 `Trident Operator`)를 사용하여 사용할 수 있습니다 `tridentctl` Astra Trident를 관리하려면 을(를) 확인해야 합니다 `KUBECONFIG` 환경 변수가 설정되었습니다. 이는 Kubernetes 클러스터를 나타내는 데 필요합니다 `tridentctl` 에 대해 작업해야 합니다. 여러 Kubernetes 환경을 사용할 때는 가 반드시 에 적합해야 합니다 `KUBECONFIG` 파일을 정확하게 소싱합니다.
- iSCSI PVS에 대해 온라인 공간 재확보를 수행하려면 작업자 노드의 기본 OS에 볼륨에 마운트 옵션을 전달해야 할 수 있습니다. 이 내용은 가 필요한 RHEL/RedHat CoreOS 인스턴스에 적용됩니다 `discard` "마운트 옵션"에 `mountmounted` 폐기 옵션이 포함되어 있는지 확인합니다[`StorageClass^`] 를 클릭하여 온라인 블록 버기를 지원합니다.
- Kubernetes 클러스터당 Astra Trident 인스턴스가 두 개 이상 있는 경우, Astra Trident가 다른 인스턴스와 통신할 수 없고 자신이 생성한 다른 볼륨을 검색할 수 없기 때문에 클러스터 내에서 둘 이상의 인스턴스가 실행될 경우 예기치 않거나 잘못된 동작이 발생합니다. Kubernetes 클러스터당 하나의 Astra Trident 인스턴스만 있어야 합니다.
- Astra Trident 기반인 경우 `StorageClass` Astra Trident가 오프라인인 동안 Kubernetes에서 객체가 삭제되며, Astra Trident는 온라인 상태로 돌아올 때 데이터베이스에서 해당 스토리지 클래스를 제거하지 않습니다. 를 사용하여 이러한 스토리지 클래스를 삭제해야 합니다 `tridentctl` 또는 REST API를 선택합니다.
- 사용자가 해당 PVC를 삭제하기 전에 Astra Trident가 프로비저닝한 PV를 삭제하는 경우 Astra Trident는 백업 볼륨을 자동으로 삭제하지 않습니다. 를 통해 볼륨을 제거해야 합니다 `tridentctl` 또는 REST API를 선택합니다.
- `FlexGroup`은 애그리게이트 세트가 각 프로비저닝 요청에서 고유하지 않으면 한 번에 둘 이상의 ONTAP를 동시에 프로비저닝할 수 없습니다.
- IPv6를 통해 Astra Trident를 사용할 때는 을 지정해야 합니다 `managementLIF` 및 `dataLIF` 백엔드 정의에서 대괄호로 묶습니다. 예를 들면, 다음과 같습니다. [`fd20:8b1e:b258:2000:f816:3eff:feec:0`].



지정할 수 없습니다 `dataLIF` ONTAP SAN 백엔드에서 Astra Trident는 사용 가능한 모든 iSCSI LIF를 검색하여 다중 경로 세션을 설정하는 데 사용합니다.

- 를 사용하는 경우 `solidfire-san` OpenShift 4.5를 사용하는 드라이버 기본 작업자 노드가 CHAP 인증 알고리즘으로 MD5를 사용하는지 확인합니다. 보안 FIPS 호환 CHAP 알고리즘 SHA1, SHA-256 및 SHA3-256은 Element 12.7에서 사용할 수 있습니다.

자세한 내용을 확인하십시오

- ["Astra Trident GitHub를 참조하십시오"](#)
- ["Astra Trident 블로그"](#)

개념

Astra Trident에 대해 자세히 알아보십시오

Astra Trident는 NetApp이 의 일부로 유지 관리하는 완전히 지원되는 오픈 소스 프로젝트입니다 ["Astra 제품군"](#). 컨테이너 스토리지 인터페이스(CSI)와 같은 업계 표준 인터페이스를 사용하여 컨테이너화된 애플리케이션의 지속성 요구를 충족하도록 설계되었습니다.

개요

Astra Trident는 Kubernetes 클러스터에 Pod로 구축하고 Kubernetes 워크로드에 동적 스토리지 오케스트레이션 서비스를 제공합니다. 컨테이너화된 애플리케이션을 통해 ONTAP(AFF/FAS/Select/Cloud/Amazon FSx for NetApp ONTAP), Element 소프트웨어(NetApp HCI/SolidFire), Azure NetApp Files 서비스 및 Google Cloud의 Cloud Volumes Service를 비롯한 NetApp의 광범위한 포트폴리오에서 영구 스토리지를 빠르고 쉽게 사용할 수 있습니다.

또한 Astra Trident는 NetApp Astra의 기본 기술로서 스냅샷, 백업, 복제 및 복제를 위한 NetApp의 업계 최고 데이터 관리 기술을 활용하여 Kubernetes 워크로드의 데이터 보호, 재해 복구, 이동성 및 마이그레이션 사용 사례를 다룹니다.

지원되는 **Kubernetes** 클러스터 아키텍처

Astra Trident는 다음 Kubernetes 아키텍처에서 지원됩니다.

Kubernetes 클러스터 아키텍처	지원	기본 설치
단일 마스터, 컴퓨팅	예	예
다중 마스터, 컴퓨팅	예	예
마스터, `etcd` 컴퓨팅	예	예
마스터, 인프라, 컴퓨팅	예	예

아스트라(Astra)란?

Astra를 사용하면 퍼블릭 클라우드와 온프레미스 모두에서 Kubernetes에서 실행되는 데이터 기반의 컨테이너 워크로드를 손쉽게 관리, 보호 및 이동할 수 있습니다. Astra는 퍼블릭 클라우드 및 온프레미스에 있는 NetApp의 검증되고 광범위한 스토리지 포트폴리오의 Astra Trident를 사용하여 영구 컨테이너 스토리지를 프로비저닝하고 제공합니다. 또한, 스냅샷, 백업 및 복원, 활동 로그, 데이터 보호, 재해/데이터 복구, 데이터 감사, Kubernetes 워크로드의 마이그레이션 사용 사례를 위한 액티브 클론 복제와 같은 풍부한 고급 애플리케이션 인식 데이터 관리 기능을 제공합니다.

Astra 페이지에서 무료 평가판을 신청하실 수 있습니다.

를 참조하십시오

- ["NetApp Astra 제품군"](#)
- ["Astra Control Service 문서"](#)

- ["Astra Control Center 문서"](#)
- ["Astra API 설명서"](#)

ONTAP 드라이버

Astra Trident는 ONTAP 클러스터와 통신할 수 있도록 5개의 고유한 ONTAP 스토리지 드라이버를 제공합니다. 각 드라이버가 볼륨 생성, 액세스 제어 및 해당 기능을 처리하는 방법에 대해 자세히 알아보십시오.

ONTAP 스토리지 드라이버에 대해 알아보십시오

Astra Control은 로 생성한 볼륨을 위해 원활한 보호, 재해 복구, 이동성(Kubernetes 클러스터 간에 볼륨 이동)을 제공합니다 `ontap-nas`, `ontap-nas-flexgroup`, 및 `ontap-san` 드라이버. 을 참조하십시오 ["Astra Control 복제 사전 요구 사항"](#) 를 참조하십시오.



- 를 사용해야 합니다 `ontap-nas` 데이터 보호, 재해 복구 및 이동성을 필요로 하는 운영 워크로드에 적합합니다.
- 사용 `ontap-san-economy` 예상되는 볼륨 사용량이 ONTAP가 지원하는 것보다 훨씬 많을 것으로 예상됩니다.
- 사용 `ontap-nas-economy` 예상되는 볼륨 사용량이 ONTAP에서 지원하는 것보다 훨씬 많을 것으로 예상되는 경우에만 및 입니다 `ontap-san-economy` 드라이버를 사용할 수 없습니다.
- 사용하지 마십시오 `ontap-nas-economy` 데이터 보호, 재해 복구 또는 이동성이 필요할 것으로 예상되는 경우

드라이버	프로토콜	볼륨 모드	액세스 모드가 지원됩니다	지원되는 파일 시스템
<code>ontap-nas</code>	NFS 를 참조하십시오	파일 시스템	RWO, ROX, <code>rwX</code>	"", NFS
<code>ontap-nas-economy</code>	NFS 를 참조하십시오	파일 시스템	RWO, ROX, <code>rwX</code>	"", NFS
<code>ontap-nas-flexgroup</code>	NFS 를 참조하십시오	파일 시스템	RWO, ROX, <code>rwX</code>	"", NFS
<code>ontap-san</code>	iSCSI	블록	RWO, ROX, <code>rwX</code>	파일 시스템이 없습니다. 원시 블록 디바이스입니다
<code>ontap-san</code>	iSCSI	파일 시스템	RWO, ROX 파일 시스템 볼륨 모드에서는 <code>rwX</code> 를 사용할 수 없습니다.	<code>xfS</code> , <code>ext3</code> , <code>ext4</code>

드라이버	프로토콜	볼륨 모드	액세스 모드가 지원됩니다	지원되는 파일 시스템
ontap-san-economy	iSCSI	블록	RWO, ROX, rwx	파일 시스템이 없습니다. 원시 블록 디바이스입니다
ontap-san-economy	iSCSI	파일 시스템	RWO, ROX 파일 시스템 볼륨 모드에서는 rwx를 사용할 수 없습니다.	xf _s , ext3, ext4



ONTAP 백엔드는 보안 역할(사용자 이름/암호)에 대한 로그인 자격 증명을 사용하거나 ONTAP 클러스터에 설치된 개인 키와 인증서를 사용하여 인증할 수 있습니다. 기존 백엔드를 업데이트하여 를 사용하여 한 인증 모드에서 다른 인증 모드로 이동할 수 있습니다 `tridentctl update backend`.

프로비저닝

Astra Trident의 프로비저닝에는 두 가지 주요 단계가 있습니다. 첫 번째 단계에서는 스토리지 클래스를 적절한 백엔드 스토리지 풀 세트와 연결하고 용량 할당 전에 필요한 준비 작업으로 수행됩니다. 두 번째 단계에서는 볼륨 생성 자체를 포함하며 보류 중인 볼륨의 스토리지 클래스와 연결된 스토리지 풀에서 스토리지 풀을 선택해야 합니다.

스토리지 클래스 연결

백엔드 스토리지 풀을 스토리지 클래스와 연결하려면 스토리지 클래스의 요청된 속성 및 해당 특성이 모두 필요합니다 `storagePools`, `additionalStoragePools`, 및 `excludeStoragePools` 목록. 스토리지 클래스를 생성할 때 Trident는 각 백엔드에서 제공하는 특성과 풀을 스토리지 클래스에서 요청한 속성과 비교합니다. 스토리지 풀의 속성과 이름이 요청된 모든 속성 및 풀 이름과 일치하면 Astra Trident는 해당 스토리지 클래스에 적합한 스토리지 풀 세트에 해당 스토리지 풀을 추가합니다. 또한 Astra Trident는 에 나열된 모든 스토리지 풀을 추가합니다 `additionalStoragePools` 특성이 요청된 스토리지 클래스의 모든 속성 또는 특성을 충족하지 않는 경우에도 해당 집합에 대해 나열합니다. 를 사용해야 합니다 `excludeStoragePools` 스토리지 클래스에 사용할 스토리지 풀을 재정의하고 제거할 목록입니다. Astra Trident는 새 백엔드를 추가할 때마다 유사한 프로세스를 수행하여 스토리지 풀이 기존 스토리지 클래스의 풀을 충족하는지 확인하고 제외로 표시된 항목을 제거합니다.

볼륨 생성

그런 다음 Astra Trident는 스토리지 클래스와 스토리지 풀 간의 연결을 사용하여 볼륨을 프로비저닝할 위치를 결정합니다. 볼륨을 생성할 때 Astra Trident는 먼저 해당 볼륨의 스토리지 클래스에 대한 스토리지 풀 세트를 가져옵니다. 볼륨의 프로토콜을 지정한 경우 Astra Trident는 요청된 프로토콜을 제공할 수 없는 스토리지 풀을 제거합니다(예: NetApp HCI/SolidFire 백엔드는 파일 기반 볼륨을 제공할 수 없고 ONTAP NAS 백엔드는 블록 기반 볼륨을 제공할 수 없음). Astra Trident는 이 결과 집합의 순서를 무작위로 생성하여 볼륨을 균일하게 분산시킨 다음 이를 반복함으로써 각 스토리지 풀에서 볼륨을 차례로 프로비저닝합니다. 이 오류가 1에서 성공하면 프로세스에서 발생한 모든 오류를 로깅하여 성공적으로 반환됩니다. Astra Trident는 * 요청된 스토리지 클래스 및 프로토콜에 대해 사용 가능한 모든 * 스토리지 풀을 프로비저닝하지 못한 경우에만 실패 * 를 반환합니다.

볼륨 스냅샷

Astra Trident가 드라이버를 위한 볼륨 스냅샷 생성을 어떻게 처리하는지 자세히 알아보십시오.

볼륨 스냅샷 생성에 대해 자세히 알아보십시오

- 의 경우 `ontap-nas`, `ontap-san`, `gcp-cvs`, 및 `azure-netapp-files` 드라이버, 각 PV(영구 볼륨)는 FlexVol에 매핑됩니다. 따라서 볼륨 스냅샷이 NetApp 스냅샷으로 생성됩니다. NetApp의 스냅샷 기술은 경쟁 스냅샷 기술보다 뛰어난 안정성, 확장성, 복구 가능성 및 성능을 제공합니다. 이러한 스냅샷 복사본은 생성하는 데 필요한 시간과 스토리지 공간 모두에서 매우 효율적입니다.
- 의 경우 `ontap-nas-flexgroup` 드라이버, 각 PV(영구 볼륨)는 FlexGroup에 매핑됩니다. 따라서 볼륨 스냅샷이 NetApp FlexGroup 스냅샷으로 생성됩니다. NetApp의 스냅샷 기술은 경쟁 스냅샷 기술보다 뛰어난 안정성, 확장성, 복구 가능성 및 성능을 제공합니다. 이러한 스냅샷 복사본은 생성하는 데 필요한 시간과 스토리지 공간 모두에서 매우 효율적입니다.
- 의 경우 `ontap-san-economy` 드라이버, PVS는 공유 FlexVol에 생성된 LUN에 매핑됩니다. PVS의 볼륨 스냅샷은 연결된 LUN의 FlexClone을 수행하여 구현됩니다. ONTAP의 FlexClone 기술을 사용하면 최대 규모의 데이터 세트의 복사본을 거의 즉시 생성할 수 있습니다. 복사본은 데이터 블록을 부모님과 공유하므로 메타데이터에 필요한 것만 빼고 스토리지를 사용하지 않습니다.
- 의 경우 `solidfire-san` 드라이버, 각 PV는 NetApp Element 소프트웨어/NetApp HCI 클러스터에서 생성된 LUN에 매핑됩니다. 볼륨 스냅샷은 기본 LUN의 요소 스냅샷으로 표시됩니다. 이러한 스냅샷은 특정 시점 복제본이며 소량의 시스템 리소스와 공간만 차지합니다.
- 를 사용하여 작업하는 경우 `ontap-nas` 및 `ontap-san` 드라이버 및 ONTAP 스냅샷은 FlexVol의 시점 복사본으로 FlexVol 자체에서 공간을 사용합니다. 따라서 스냅샷이 생성/예약될 때 시간이 경과함에 따라 볼륨의 쓰기 가능한 공간이 줄어들 수 있습니다. 이 문제를 해결하는 간단한 방법 중 하나는 Kubernetes를 통해 크기를 조정하여 볼륨을 늘리는 것입니다. 또 다른 옵션은 더 이상 필요하지 않은 스냅샷을 삭제하는 것입니다. Kubernetes를 통해 생성된 VolumeSnapshot이 삭제되면 Astra Trident가 연결된 ONTAP 스냅샷을 삭제합니다. Kubernetes를 통해 생성되지 않은 ONTAP 스냅샷도 삭제할 수 있습니다.

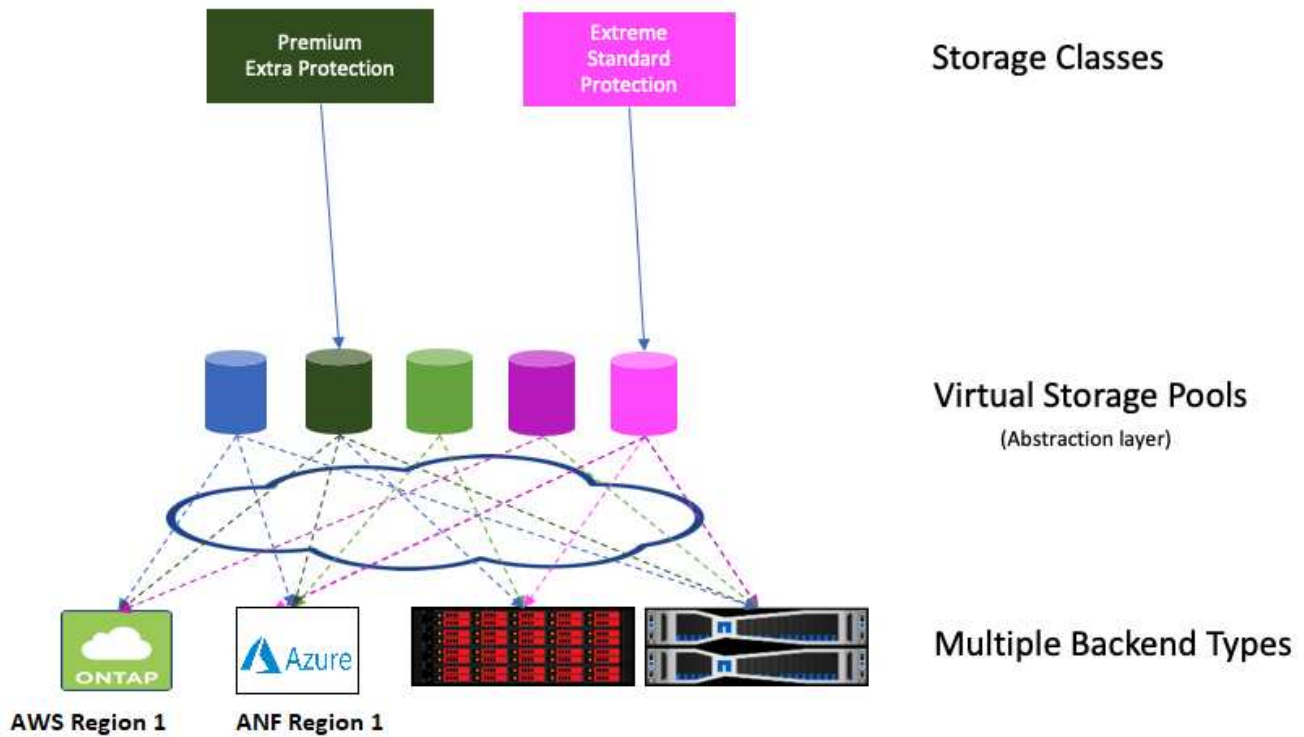
Astra Trident를 사용하면 VolumeSnapshots을 사용하여 새 PVS를 생성할 수 있습니다. 이러한 스냅샷에서 PVS를 생성하는 작업은 지원되는 ONTAP 및 CVS 백엔드에 FlexClone 기술을 사용하여 수행됩니다. 스냅샷으로부터 PV를 생성할 때, 백업 볼륨은 스냅샷의 상위 볼륨의 FlexClone입니다. 를 클릭합니다 `solidfire-san` 드라이버는 Element 소프트웨어 볼륨 클론을 사용하여 스냅샷에서 PVS를 생성합니다. 여기서 Element 스냅샷으로부터 클론을 생성합니다.

가상 풀

가상 풀은 Astra Trident 스토리지 백엔드와 Kubernetes 간의 추상화 계층을 제공합니다 StorageClasses. 관리자는 을 만들지 않고 각 백엔드에 대한 위치, 성능 및 보호와 같은 측면을 일반적인 백엔드 방식으로 정의할 수 있습니다 StorageClass 원하는 기준을 충족하는 데 사용할 물리적 백엔드, 백엔드 풀 또는 백엔드 유형을 지정합니다.

가상 풀에 대해 알아보십시오

스토리지 관리자는 Astra Trident 백엔드의 가상 풀을 JSON 또는 YAML 정의 파일로 정의할 수 있습니다.



가상 풀 목록 외부에서 지정된 모든 측면은 백엔드에 대해 전역적이며 모든 가상 풀에 적용되지만, 각 가상 풀은 하나 이상의 측면을 개별적으로 지정할 수 있습니다(백엔드-글로벌 측면 재정의).



- 가상 풀을 정의할 때 백엔드 정의에서 기존 가상 풀의 순서를 재정렬하지 마십시오.
- 기존 가상 풀에 대한 속성을 수정하지 않는 것이 좋습니다. 변경하려면 새 가상 풀을 정의해야 합니다.

대부분의 측면은 백엔드 관련 용어로 지정됩니다. 결정적으로, Aspect 값은 백엔드의 드라이버 외부에 노출되지 않으며에서 일치시킬 수 없습니다 StorageClasses. 대신 관리자는 각 가상 풀에 대해 하나 이상의 레이블을 정의합니다. 각 레이블은 키, 즉 값 쌍이며 레이블은 고유한 백엔드에서 공통될 수 있습니다. 측면과 마찬가지로 레이블을 풀별로 지정하거나 백엔드에 대해 전역으로 지정할 수 있습니다. 미리 정의된 이름과 값이 있는 측면과 달리 관리자는 필요에 따라 레이블 키와 값을 정의할 수 있습니다. 편의를 위해 스토리지 관리자는 가상 풀 및 그룹 볼륨별로 레이블을 레이블별로 정의할 수 있습니다.

A StorageClass 선택기 매개 변수 내의 레이블을 참조하여 사용할 가상 풀을 식별합니다. 가상 풀 선택기는 다음 연산자를 지원합니다.

운영자	예	풀의 레이블 값은 다음과 같아야 합니다.
=	성능 = 프리미엄	일치
!=	성능!=최고	일치하지 않습니다
in	위치(동쪽, 서쪽)	값 집합에 있어야 합니다
notin	성능 노트(실버, 브론즈)	값 집합에 있지 않습니다
<key>	보호	모든 값과 함께 존재합니다

운영자	예	플의 레이블 값은 다음과 같아야 합니다.
!<key>	! 보호	존재하지 않습니다

볼륨 액세스 그룹

Astra Trident가 사용하는 방법에 대해 자세히 알아보십시오 ["볼륨 액세스 그룹"](#).



CHAP를 사용하는 경우 이 섹션을 무시하십시오. CHAP는 관리를 단순화하고 아래 설명된 배울 제한을 피하는 것이 좋습니다. 또한 CSI 모드에서 Astra Trident를 사용하는 경우 이 섹션을 무시할 수 있습니다. Astra Trident는 향상된 CSI 구축 기능으로 설치된 경우 CHAP를 사용합니다.

볼륨 액세스 그룹에 대해 알아보십시오

Astra Trident는 볼륨 액세스 그룹을 사용하여 프로비저닝하는 볼륨에 대한 액세스를 제어할 수 있습니다. CHAP가 해제되어 있으면 `trident` 라는 액세스 그룹을 찾을 것으로 예상합니다 `trident` 구성에 하나 이상의 액세스 그룹 ID를 지정하지 않은 경우

Astra Trident는 새 볼륨을 구성된 액세스 그룹에 연결하지만 액세스 그룹 자체를 생성하거나 관리하지 않습니다. 스토리지 백엔드를 Astra Trident에 추가하기 전에 액세스 그룹이 있어야 하며, 해당 백엔드에서 프로비저닝한 볼륨을 마운트할 수 있는 Kubernetes 클러스터의 모든 노드에서 iSCSI IQN을 포함해야 합니다. 대부분의 설치에서는 클러스터의 모든 작업자 노드를 포함합니다.

64개 이상의 노드가 있는 Kubernetes 클러스터의 경우 다중 액세스 그룹을 사용해야 합니다. 각 액세스 그룹에는 최대 64개의 IQN이 포함될 수 있으며 각 볼륨은 4개의 액세스 그룹에 속할 수 있습니다. 최대 4개의 액세스 그룹이 구성되어 있는 경우, 클러스터의 모든 노드에서 최대 256개의 노드 크기로 모든 볼륨에 액세스할 수 있습니다. 볼륨 액세스 그룹에 대한 최신 제한은 ["여기"](#) 를 참조하십시오.

기본값을 사용하는 구성에서 구성을 수정하는 경우 `trident` 다른 사람을 사용하는 그룹에 대한 액세스 그룹에는 의 ID도 포함됩니다 `trident` 목록에 있는 액세스 그룹.

시작하십시오

시도해 보십시오

NetApp은 사용자가 요청할 수 있는 즉시 사용 가능한 연구소 이미지를 제공합니다 "[NetApp 시험 구동](#)".

시험 구동에 대해 자세히 알아보십시오

테스트 드라이브는 3노드 Kubernetes 클러스터 및 Astra Trident가 설치 및 구성된 샌드박스 환경을 제공합니다. Astra Trident에 대해 알아보고 기능을 둘러보는 것도 좋습니다.

또 다른 옵션은 을 보는 것입니다 "[kubeadm 설치 가이드](#)" Kubernetes에서 제공:



이러한 지침을 운영 환경에 사용하여 구축한 Kubernetes 클러스터를 사용해서는 안 됩니다. 배포 시 제공되는 운영 구축 가이드를 사용하여 즉시 프로덕션할 수 있는 클러스터를 생성할 수 있습니다.

Kubernetes를 처음 사용하는 경우 개념 및 툴에 대해 자세히 알아보십시오 "[여기](#)".

요구 사항

Astra Trident를 설치하기 전에 이러한 일반 시스템 요구 사항을 검토해야 합니다. 특정 백엔드에 추가 요구 사항이 있을 수 있습니다.

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

** 중요 정보 Astra Trident **

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 입니다
`find_multipaths: no` 다중 경로 .conf 파일

비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart`
`multipath.conf` 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다
`find_multipaths: no` 21.07 릴리스 이후.

지원되는 프론트엔드(오케스트레이터)

Astra Trident는 다음을 비롯한 여러 컨테이너 엔진과 오케스트레이터가 지원됩니다.

- Anthos On-Premise(VMware) 및 Anthos의 Bare Metal 1.9, 1.10, 1.11
- Kubernetes 1.21-1.26

- Mirantis Kubernetes 엔진 3.5
- OpenShift 4.9-4.12

Trident 연산자는 다음 릴리즈에서 지원됩니다.

- Anthos On-Premise(VMware) 및 Anthos의 Bare Metal 1.9, 1.10, 1.11
- Kubernetes 1.21-1.26
- OpenShift 4.9-4.12

Astra Trident는 GKE(Google Kubernetes Engine), EKS(Amazon Elastic Kubernetes Service), AKS(Azure Kubernetes Service), Rancher, VMware Tanzu Portfolio를 비롯한 기타 완전 관리형 및 자체 관리 Kubernetes 오픈링과도 작동합니다.



Astra Trident가 설치된 Kubernetes 클러스터를 1.24에서 1.25 이상으로 업그레이드하기 전에 을 참조하십시오 **"제어 기반 작업자 설치를 업그레이드합니다"**.

지원되는 백엔드(스토리지)

Astra Trident를 사용하려면 다음 중 하나 이상의 지원되는 백엔드가 필요합니다.

- NetApp ONTAP용 Amazon FSx
- Azure NetApp Files
- Cloud Volumes ONTAP
- GCP용 Cloud Volumes Service
- FAS/AFF/9.5 이상을 선택합니다
- NetApp All SAN 어레이(ASA)
- NetApp HCI/Element 소프트웨어 11 이상

피처 요구 사항

아래 표에는 Astra Trident의 이번 릴리즈와 함께 사용할 수 있는 기능과 지원하는 Kubernetes 버전이 요약되어 있습니다.

피처	Kubernetes 버전	기능 게이트가 필요합니까?
CSI Trident	1.21-1.26	아니요
볼륨 스냅샷	1.21-1.26	아니요
체적 스냅샷의 PVC	1.21-1.26	아니요
iSCSI PV 크기 조정	1.21-1.26	아니요
ONTAP 양방향 CHAP	1.21-1.26	아니요

피처	Kubernetes 버전	기능 게이트가 필요합니까?
동적 내보내기 정책	1.21-1.26	아니요
Trident 연산자	1.21-1.26	아니요
CSI 토폴로지	1.21-1.26	아니요

호스트 운영 체제를 테스트했습니다

Astra Trident가 공식적으로 특정 운영 체제를 지원하지 않지만 다음과 같은 운영 체제가 작동하는 것으로 알려져 있습니다.

- OpenShift Container Platform에서 지원하는 RedHat CoreOS(RHCOS) 버전
- RHEL 8+
- Ubuntu 22.04 이상
- Windows Server 2019

기본적으로 Astra Trident는 컨테이너에서 실행되므로 모든 Linux 작업자에서 실행됩니다. 그러나 이러한 작업자들은 표준 NFS 클라이언트 또는 iSCSI 이니시에이터를 사용하여 Astra Trident가 제공하는 볼륨을 사용 중인 백엔드에 따라 마운트할 수 있어야 합니다.

를 클릭합니다 `tridentctl` 이 유틸리티는 이러한 Linux 배포판에서도 실행됩니다.

호스트 구성

Kubernetes 클러스터의 모든 작업자 노드는 Pod용으로 프로비저닝된 볼륨을 마운트할 수 있어야 합니다. 작업자 노드를 준비하려면 드라이버 선택에 따라 NFS 또는 iSCSI 도구를 설치해야 합니다.

["작업자 노드를 준비합니다"](#)

스토리지 시스템 구성

Astra Trident는 백엔드 구성에서 사용하기 전에 스토리지 시스템을 변경해야 할 수 있습니다.

["백엔드 구성"](#)

Astra Trident 포트

Astra Trident는 통신을 위해 특정 포트에 액세스해야 합니다.

["Astra Trident 포트"](#)

컨테이너 이미지 및 해당 Kubernetes 버전

공기 박형 설치의 경우 다음 목록은 Astra Trident를 설치하는 데 필요한 컨테이너 이미지의 참조입니다. 를 사용합니다 `tridentctl images` 명령을 사용하여 필요한 컨테이너 이미지 목록을 확인합니다.

Kubernetes 버전	컨테이너 이미지
v1.21.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)
v1.22.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)
v1.23.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)

Kubernetes 버전	컨테이너 이미지
v1.24.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)
v1.25.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)
v1.26.0	<ul style="list-style-type: none"> • Docker.IO/NetApp/트리덴트: 23.01.1 • Docker.IO/NetApp/트리덴트 - 자동 지원: 23.01 • registry.k8s.io/SIG-storage/CSI-v3.4.0 • registry.k8s.io/SIG-storage/CSI-attacher:v4.1.0 • registry.k8s.io/SIG-storage/CSI-resizer: v1.7.0 • registry.k8s.io/SIG-storage/CSI-snapshotter:v6.2.1 • registry.k8s.io/SIG-storage/CSI-node-driver-registrar: v2.7.0 • Docker.IO/NetApp/트리덴트 - 운영자: 23.01.1(선택 사항)



Kubernetes 버전 1.21 이상에서는 검증된 을 사용합니다 registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x ?곡깊?? v1 에서 지원하는 버전입니다 volumesnapshots.snapshot.storage.k8s.gcr.io CRD 를 누릅니다 v1beta1 에서 CRD를 지원하는 버전입니다 v1 버전, 검증된 을 사용합니다 registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x 이미지.

Astra Trident를 설치합니다

Astra Trident 설치에 대해 자세히 알아보십시오

Astra Trident를 다양한 환경과 조직에 설치할 수 있도록 NetApp은 다양한 설치 옵션을 제공합니다. Trident 연산자(수동 또는 Helm 사용) 또는 를 사용하여 Astra Trident를 설치할 수 있습니다 `tridentctl`. 이 항목에서는 적합한 설치 프로세스를 선택하는 데 필요한 중요한 정보를 제공합니다.

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

 중요 정보 Astra Trident

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 `입니다`
`find_multipaths: no` 다중 경로 `.conf` 파일

비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart`
`multipath.conf` 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다
`find_multipaths: no` 21.07 릴리스 이후.

시작하기 전에

설치 경로에 관계없이 다음 항목이 있어야 합니다.

- 지원되는 버전의 Kubernetes 및 기능 요구 사항을 실행하는 지원되는 Kubernetes 클러스터에 대한 모든 권한이 활성화됩니다. 를 검토합니다 ["요구 사항"](#) 를 참조하십시오.
- 지원되는 NetApp 스토리지 시스템에 대한 액세스
- 모든 Kubernetes 작업자 노드에서 볼륨을 마운트할 수 있습니다.
- 가 설치된 Linux 호스트 `kubectl` (또는 `oc`, OpenShift를 사용하는 경우) 사용하려는 Kubernetes 클러스터를 관리하도록 설치 및 구성한 것입니다.
- 를 클릭합니다 `KUBECONFIG` Kubernetes 클러스터 구성을 가리키도록 설정된 환경 변수입니다.
- Docker Enterprise와 함께 Kubernetes를 사용하는 경우, ["다음 단계에 따라 CLI 액세스를 설정합니다"](#).



에 익숙하지 않은 경우 ["기본 개념"](#)이제 아주 좋은 시간입니다.

설치 방법을 선택합니다

적합한 설치 방법을 선택합니다. 의 고려 사항도 검토해야 합니다 ["방법 간 이동"](#) 결정을 내리기 전에

Trident 연산자 사용

수동으로 배포하던 Hrom을 사용하면 Trident 운영자는 설치를 단순화하고 Astra Trident 리소스를 동적으로 관리할 수 있는 훌륭한 방법입니다. 물론 가능합니다 "[Trident 운영자 배포를 사용자 지정합니다](#)"의 속성을 사용합니다 TridentOrchestrator 사용자 지정 리소스(CR).

Trident 연산자를 사용하면 다음과 같은 이점이 있습니다.

 Astra Trident 객체 생성

Trident 운영자가 Kubernetes 버전에 대해 다음 오브젝트를 자동으로 생성합니다.

- 운영자용 ServiceAccount입니다
- ServiceAccount에 대한 ClusterRole 및 ClusterRoleBinding
- 전용 PodSecurityPolicy(Kubernetes 1.25 이하)
- 작업자 자체

 자동 복구 기능

운영자는 Astra Trident 설치를 모니터링하고 구축이 삭제되거나 실수로 수정된 경우와 같은 문제를 해결하기 위한 조치를 적극적으로 수행합니다. A trident-operator-`<generated-id>` 를 연결하는 POD가 생성됩니다 TridentOrchestrator Astra Trident가 설치된 CR. 이렇게 하면 클러스터에 Astra Trident 인스턴스가 하나만 있고 설치가 제어되므로 설치가 매우 강력합니다. 설치 변경(예: 배포 또는 노드 반점 삭제)이 수행되면 운영자가 이를 식별하고 개별적으로 수정합니다.

 기존 설치에 대한 간편한 업데이트

기존 배포를 운영자로 쉽게 업데이트할 수 있습니다. 를 편집하기만 하면 됩니다 TridentOrchestrator CR을 사용하여 설치를 업데이트합니다.

예를 들어, Astra Trident를 활성화하여 디버그 로그를 생성해야 하는 시나리오를 생각해 보십시오. 이렇게 하려면 에 패치를 적용합니다 TridentOrchestrator 를 눌러 설정합니다 spec.debug 를 선택합니다 true:

```
kubectl patch torc <trident-orchestrator-name> -n trident --type=merge  
-p '{"spec":{"debug":true}}'
```

이후 TridentOrchestrator 이 업데이트되면 운영자가 업데이트를 처리하고 기존 설치를 패치합니다. 이렇게 하면 새 포드가 생성되어 설치를 적절하게 수정할 수 있습니다.

를 처리하는 자동 Kubernetes 업그레이드

클러스터의 Kubernetes 버전이 지원되는 버전으로 업그레이드되면 운영자는 기존 Astra Trident 설치를 자동으로 업데이트하고 Kubernetes 버전 요구사항을 충족하도록 변경합니다.



클러스터가 지원되지 않는 버전으로 업그레이드되면 운영자는 Astra Trident를 설치할 수 없습니다. Astra Trident가 운영자와 함께 이미 설치된 경우 Astra Trident가 지원되지 않는 Kubernetes 버전에 설치되었음을 나타내는 경고가 표시됩니다.

BlueXP(이전의 Cloud Manager) 를 사용한 Kubernetes 클러스터 관리

와 함께 "BlueXP를 사용하는 Astra Trident" Astra Trident의 최신 버전으로 업그레이드하고, 스토리지 클래스를 추가 및 관리하고, 작업 환경에 연결한 다음, Cloud Backup Service를 사용하여 영구 볼륨을 백업할 수 있습니다. BlueXP는 Trident 연산자를 사용하여 수동으로 또는 Helm을 사용하여 Astra Trident 구축을 지원합니다.

사용 `tridentctl`

업그레이드해야 하는 기존 배포가 있거나 배포를 사용자 지정하려는 경우 고려해야 합니다. Astra Trident를 구축하는 기존 방법입니다.

가능합니다 Trident 리소스에 대한 매니페스트를 생성합니다. 여기에는 구축, 개발/제거, 서비스 계정, Astra Trident가 설치의 일부로 생성한 클러스터 역할 등이 포함됩니다.



22.04 릴리즈부터는 Astra Trident가 설치될 때마다 AES 키가 더 이상 다시 생성되지 않습니다. 이 릴리즈를 통해 Astra Trident는 설치 전반에 걸쳐 유지되는 새로운 비밀 객체를 설치합니다. 즉, `tridentctl` 22.04에서는 Trident의 이전 버전을 제거할 수 있지만 이전 버전에서는 22.04 설치를 제거할 수 없습니다. 적절한 `installation_method_`를 선택합니다.

설치 모드를 선택합니다

조직에서 요구하는 *installation mode*(Standard, Offline 또는 Remote)를 기반으로 배포 프로세스를 결정합니다.

표준 설치

이것은 Astra Trident를 설치하는 가장 쉬운 방법이며 네트워크 제한이 없는 대부분의 환경에서 작동합니다. 표준 설치 모드에서는 기본 레지스트리를 사용하여 필요한 Trident를 저장합니다 (docker.io)와 CSI를 참조하십시오 (registry.k8s.io) 이미지.

표준 모드를 사용하는 경우 Astra Trident 설치 프로그램이 다음을 수행합니다.

- 인터넷을 통해 컨테이너 이미지를 가져옵니다
- Kubernetes 클러스터의 모든 적격 노드에서 Astra Trident Pod를 가동하는 구축 또는 노드 데모시작을 생성합니다

오프라인 설치

오프라인 설치 모드는 공기 박수나 안전한 위치에 필요할 수 있습니다. 이 시나리오에서는 필요한 Trident 및 CSI 이미지를 저장하기 위해 단일 전용 미러링 레지스트리 또는 두 개의 미러링된 레지스트리를 만들 수 있습니다.



레지스트리 구성에 관계없이 CSI 이미지는 하나의 레지스트리에 있어야 합니다.

원격 설치

다음은 원격 설치 프로세스에 대한 상위 수준의 개요입니다.

- 적절한 버전의 를 배포합니다 kubectl Astra Trident를 구축하려는 원격 머신
- Kubernetes 클러스터에서 구성 파일을 복사하고 를 설정합니다 KUBECONFIG 원격 시스템의 환경 변수.
- 를 시작합니다 kubectl get nodes 명령을 실행하여 필요한 Kubernetes 클러스터에 연결할 수 있는지 확인하십시오.
- 표준 설치 단계를 사용하여 원격 컴퓨터에서 배포를 완료합니다.

방법 및 모드에 따라 프로세스를 선택합니다

결정을 내린 후 적절한 프로세스를 선택합니다.

방법	설치 모드
Trident 운영자(수동)	"표준 설치" "오프라인 설치"
Trident 운영자(제어)	"표준 설치" "오프라인 설치"
tridentctl	"표준 또는 오프라인 설치"

설치 방법 간 이동

설치 방법을 변경할 수 있습니다. 이렇게 하기 전에 다음 사항을 고려하십시오.

- Astra Trident를 설치 및 제거할 때는 항상 동일한 방법을 사용하십시오. 을(를) 배포한 경우 `tridentctl`, 의 해당 버전을 사용해야 합니다 `tridentctl` Astra Trident를 제거하는 바이너리. 마찬가지로 연산자를 사용하여 를 배포하는 경우에는 를 편집해야 합니다 `TridentOrchestrator` CR 및 `SET spec.uninstall=true` Astra Trident를 제거합니다.
- 대신 를 제거하고 대신 사용할 운영자 기반 배포가 있는 경우 `tridentctl` Astra Trident를 배포하려면 먼저 편집해야 합니다 `TridentOrchestrator` 그리고 설정합니다 `spec.uninstall=true` Astra Trident를 제거합니다. 그런 다음 삭제합니다 `TridentOrchestrator` 및 작업자 배포. 그런 다음 를 사용하여 를 설치할 수 있습니다 `tridentctl`.
- 작업자 기반의 수동 배포를 사용하고 H제어 기반 Trident 연산자 배포를 사용하려는 경우 먼저 수동으로 연산자를 제거한 다음 Helm 설치를 수행해야 합니다. 이를 통해 Helm은 필요한 레이블 및 주석을 사용하여 Trident 연산자를 배포할 수 있습니다. 이렇게 하지 않으면 레이블 유효성 검사 오류 및 주석 유효성 검사 오류와 함께 H제어 기반 Trident 연산자 배포가 실패합니다. 가 있는 경우 `tridentctl` 기반 배포에서는 문제 없이 Helm 기반 배포를 사용할 수 있습니다.

기타 알려진 구성 옵션

VMware Tanzu 포트폴리오 제품에 Astra Trident를 설치할 경우:

- 클러스터는 권한이 있는 워크로드를 지원해야 합니다.
- 를 클릭합니다 `--kubelet-dir` 플래그를 `kubelet` 디렉터리의 위치로 설정해야 합니다. 기본적으로 이 값은 `/var/vcap/data/kubelet`.

를 사용하여 `kubelet` 위치 지정 `--kubelet-dir` Trident Operator, Helm 및 에 대해 작업하는 것으로 알려져 있습니다 `tridentctl` 적합합니다.

Trident 연산자를 사용하여 설치합니다

Trident 연산자 수동 배포(표준 모드)

Trident 연산자를 수동으로 구축하여 Astra Trident를 설치할 수 있습니다. 이 프로세스는 Astra Trident에 필요한 컨테이너 이미지가 개인 레지스트리에 저장되어 있지 않은 설치에 적용됩니다. 개인 이미지 레지스트리가 있는 경우 를 사용합니다 **"오프라인 배포를 위한 프로세스입니다"**.

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

 중요 정보 Astra Trident

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 `입니다` `find_multipaths: no` 다중 경로 `.conf` 파일

비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart` `multipath.conf` 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다 `find_multipaths: no` 21.07 릴리스 이후.

Trident 연산자를 수동으로 구축하고 Trident를 설치합니다

검토 "설치 개요" 설치 사전 요구 사항을 충족하고 환경에 맞는 올바른 설치 옵션을 선택했는지 확인합니다.

시작하기 전에

설치를 시작하기 전에 Linux 호스트에 로그인하여 작업 관리 여부를 확인합니다. "지원되는 Kubernetes 클러스터" 필요한 권한이 있어야 합니다.



OpenShift에서는 을 사용합니다 oc 대신 kubectl 다음 모든 예에서 를 실행하여 먼저 * system:admin * 으로 로그인합니다 oc login -u system:admin 또는 oc login -u kube-admin.

1. Kubernetes 버전 확인:

```
kubectl version
```

2. 클러스터 관리자 권한 확인:

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. Docker Hub의 이미지를 사용하는 Pod를 시작하고 Pod 네트워크를 통해 스토리지 시스템에 연결할 수 있는지 확인합니다.

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
ping <management IP>
```

1단계: Trident 설치 프로그램 패키지를 다운로드합니다

Astra Trident 설치 프로그램 패키지에는 Trident 운영자를 구축하고 Astra Trident를 설치하는 데 필요한 모든 것이 들어 있습니다. 에서 최신 버전의 Trident 설치 프로그램을 다운로드하고 압축을 풉니다 "GitHub의 [_Assets_](#) 섹션".

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

2단계: 을 작성합니다 TridentOrchestrator CRD

를 생성합니다 TridentOrchestrator 사용자 정의 리소스 정의(CRD). 을(를) 생성합니다 TridentOrchestrator 나중에 사용자 지정 리소스. 에서 적절한 CRD YAML 버전을 사용하십시오 deploy/crds 를 작성합니다 TridentOrchestrator CRD

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3단계: Trident 연산자를 배포합니다

Astra Trident 설치 관리자는 연산자를 설치하고 관련 개체를 만드는 데 사용할 수 있는 번들 파일을 제공합니다. 번들 파일은 기본 구성을 사용하여 운영자를 구축하고 Astra Trident를 설치하는 간편한 방법입니다.

- Kubernetes 1.24 이하 를 실행하는 클러스터의 경우, 를 사용합니다 bundle_pre_1_25.yaml.
- Kubernetes 1.25 이상을 실행하는 클러스터의 경우 를 사용합니다 bundle_post_1_25.yaml.

Trident 설치 관리자는 에 연산자를 배포합니다 trident 네임스페이스. 를 누릅니다 trident 네임스페이스가 없습니다. 를 사용하십시오 kubectl apply -f deploy/namespace.yaml 를 눌러 만듭니다.

단계

1. 리소스를 생성하고 연산자를 배포합니다.

```
kubectl create -f deploy/<bundle>.yaml
```



를 제외한 네임스페이스에 연산자를 배포합니다 trident 네임스페이스, 업데이트 serviceaccount.yaml, clusterrolebinding.yaml 및 operator.yaml 을 사용하여 번들 파일을 생성합니다 kustomization.yaml:

```
kubectl kustomize deploy/ > deploy/<bundle>.yaml
```

2. 작업자가 배치되었는지 확인합니다.

```
kubectl get deployment -n <operator-namespace>
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
trident-operator	1/1	1	1	3m



Kubernetes 클러스터에는 운영자의 인스턴스 * 하나가 있어야 합니다. Trident 연산자의 여러 배포를 생성하지 마십시오.

4단계: 을 작성합니다 TridentOrchestrator **Trident**를 설치합니다

이제 를 만들 수 있습니다 TridentOrchestrator Astra Trident를 설치합니다. 필요에 따라 할 수 있습니다 "Trident 설치를 사용자 지정합니다"의 속성을 사용합니다 TridentOrchestrator 사양

```

kubect1 create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

kubect1 describe torc trident

Name:          trident
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Spec:
  Debug:       true
  Namespace:   trident
Status:
  Current Installation Params:
    IPv6:                false
    Autosupport Hostname:
    Autosupport Image:   netapp/trident-autosupport:23.01
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:          30
    Kubelet Dir:         /var/lib/kubelet
    Log Format:           text
    Silence Autosupport: false
    Trident Image:       netapp/trident:23.01.1
  Message:             Trident installed Namespace:
trident
  Status:              Installed
  Version:             v23.01.1
Events:
  Type Reason Age From Message ---- -
-----
Normal
Installing 74s trident-operator.netapp.io Installing Trident Normal
Installed 67s trident-operator.netapp.io Trident installed

```

설치를 확인합니다

설치를 확인하는 방법에는 여러 가지가 있습니다.

사용 TridentOrchestrator 상태

의 상태입니다 TridentOrchestrator 설치가 성공적으로 완료되었는지 여부를 나타내고 설치된 Trident의 버전을 표시합니다. 설치하는 동안 의 상태입니다 TridentOrchestrator 변경 시작 Installing 를 선택합니다 Installed. 을(를) 관찰하면 Failed 상태 및 운영자가 자체적으로 복구할 수 없습니다. "로그를 확인합니다".

상태	설명
설치 중	이 옵션을 사용하여 Astra Trident를 설치합니다 TridentOrchestrator 있습니다.
설치되어 있습니다	Astra Trident가 성공적으로 설치되었습니다.
제거 중	그 이유는 운영자가 Astra Trident를 제거하는 중입니다 spec.uninstall=true.
제거되었습니다	Astra Trident가 제거되었습니다.
실패했습니다	운영자가 Astra Trident를 설치, 패치, 업데이트 또는 제거할 수 없습니다. 이 상태에서 자동으로 복구를 시도합니다. 이 상태가 지속되면 문제 해결이 필요합니다.
업데이트 중	운영자가 기존 설치를 업데이트하고 있습니다.
오류	를 클릭합니다 TridentOrchestrator 사용되지 않습니다. 다른 파일이 이미 있습니다.

POD 생성 상태 사용

생성된 Pod의 상태를 검토하여 Astra Trident 설치가 완료되었는지 확인할 수 있습니다.

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
AGE			
trident-controller-7d466bf5c7-v4cpw 1m	6/6	Running	0
trident-node-linux-mr6zc 1m	2/2	Running	0
trident-node-linux-xrp7w 1m	2/2	Running	0
trident-node-linux-zh2jt 1m	2/2	Running	0
trident-operator-766f7b8658-ldzsv 3m	1/1	Running	0

사용 tridentctl

을 사용할 수 있습니다 tridentctl 설치된 Astra Trident의 버전을 확인합니다.

```
./tridentctl -n trident version
```

```
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 23.01.1        | 23.01.1        |
+-----+-----+
```

다음 단계

이제 가능합니다 "백엔드 및 스토리지 클래스를 생성하고, 볼륨을 프로비저닝하고, POD에 볼륨을 마운트합니다".

Trident 연산자 수동 배포(오프라인 모드)

Trident 연산자를 수동으로 구축하여 Astra Trident를 설치할 수 있습니다. 이 프로세스는 Astra Trident에 필요한 컨테이너 이미지가 개인 레지스트리에 저장된 설치에 적용됩니다. 개인 이미지 레지스트리가 없는 경우 를 사용합니다 "표준 배포 프로세스".

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

 중요 정보 Astra Trident

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 입니다
find_multipaths: no 다중 경로 .conf 파일

비 경로 다중화 구성 또는 의 사용 find_multipaths: yes 또는 find_multipaths: smart multipath.conf 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다 find_multipaths: no 21.07 릴리스 이후.

Trident 연산자를 수동으로 구축하고 Trident를 설치합니다

검토 "설치 개요" 설치 사전 요구 사항을 충족하고 환경에 맞는 올바른 설치 옵션을 선택했는지 확인합니다.

시작하기 전에

Linux 호스트에 로그인하여 작업 및 을 관리하고 있는지 확인합니다 "지원되는 Kubernetes 클러스터" 필요한 권한이 있어야 합니다.



OpenShift에서는 을 사용합니다 oc 대신 kubectl 다음 모든 예에서 를 실행하여 먼저 * system:admin * 으로 로그인합니다 oc login -u system:admin 또는 oc login -u kube-admin.

1. Kubernetes 버전 확인:

```
kubectl version
```

2. 클러스터 관리자 권한 확인:

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. Docker Hub의 이미지를 사용하는 Pod를 시작하고 Pod 네트워크를 통해 스토리지 시스템에 연결할 수 있는지 확인합니다.

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
ping <management IP>
```

1단계: Trident 설치 프로그램 패키지를 다운로드합니다

Astra Trident 설치 프로그램 패키지에는 Trident 운영자를 구축하고 Astra Trident를 설치하는 데 필요한 모든 것이 들어 있습니다. 에서 최신 버전의 Trident 설치 프로그램을 다운로드하고 압축을 풉니다 "[GitHub의 _Assets_ 섹션](#)".

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

2단계: 을 작성합니다 TridentOrchestrator CRD

를 생성합니다 TridentOrchestrator 사용자 정의 리소스 정의(CRD). 을(를) 생성합니다 TridentOrchestrator 나중에 사용자 지정 리소스. 에서 적절한 CRD YAML 버전을 사용하십시오 deploy/crds 를 작성합니다 TridentOrchestrator CRD:

```
kubectl create -f deploy/crds/<VERSION>.yaml
```

3단계: 운영자의 레지스트리 위치를 업데이트합니다

인치 /deploy/operator.yaml, 업데이트 image: docker.io/netapp/trident-operator:23.01.1 이미지 레지스트리의 위치를 반영합니다. 귀사의 "[Trident 및 CSI 이미지](#)" 하나의 레지스트리 또는 다른 레지스트리에 있을 수 있지만 모든 CSI 이미지는 동일한 레지스트리에 있어야 합니다. 예를 들면 다음과 같습니다.

- image: <your-registry>/trident-operator:23.01.1 이미지가 모두 하나의 레지스트리에 있는 경우
- image: <your-registry>/netapp/trident-operator:23.01.1 Trident 이미지가 CSI 이미지와

다른 레지스트리에 있는 경우

4단계: Trident 연산자를 배포합니다

Trident 설치 관리자는 에 연산자를 배포합니다 trident 네임스페이스. 를 누릅니다 trident 네임스페이스가 없습니다. 를 사용하십시오 kubectl apply -f deploy/namespace.yaml 를 눌러 만듭니다.

를 제외한 네임스페이스에 연산자를 배포합니다 trident 네임스페이스, 업데이트 serviceaccount.yaml, clusterrolebinding.yaml 및 operator.yaml 오퍼레이터 배치 전.

1. 리소스를 생성하고 연산자를 배포합니다.

```
kubectl kustomize deploy/ > deploy/<BUNDLE>.yaml
```

Astra Trident 설치 관리자는 연산자를 설치하고 관련 개체를 만드는 데 사용할 수 있는 번들 파일을 제공합니다. 번들 파일은 기본 구성을 사용하여 운영자를 구축하고 Astra Trident를 설치하는 간편한 방법입니다.



- Kubernetes 1.24 이하 를 실행하는 클러스터의 경우, 를 사용합니다 bundle_pre_1_25.yaml.
- Kubernetes 1.25 이상을 실행하는 클러스터의 경우 를 사용합니다 bundle_post_1_25.yaml.

2. 작업자가 배치되었는지 확인합니다.

```
kubectl get deployment -n <operator-namespace>
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
trident-operator	1/1	1	1	3m



Kubernetes 클러스터에는 운영자의 인스턴스 * 하나가 있어야 합니다. Trident 연산자의 여러 배포를 생성하지 마십시오.

5단계: 에서 이미지 레지스트리 위치를 업데이트합니다 TridentOrchestrator

귀사의 "Trident 및 CSI 이미지" 하나의 레지스트리 또는 다른 레지스트리에 있을 수 있지만 모든 CSI 이미지는 동일한 레지스트리에 있어야 합니다. 업데이트 deploy/crds/tridentorchestrator_cr.yaml 레지스트리 구성에 따라 추가 위치 사양을 추가하려면 다음을 수행합니다.

하나의 레지스트리에 있는 이미지

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:23.01"
tridentImage: "<your-registry>/trident:23.01.1"
```

다른 레지스트리의 이미지

추가해야 합니다 sig-storage 를 누릅니다 imageRegistry 다른 레지스트리 위치를 사용합니다.

```
imageRegistry: "<your-registry>/sig-storage"
autosupportImage: "<your-registry>/netapp/trident-autosupport:23.01"
tridentImage: "<your-registry>/netapp/trident:23.01.1"
```

6단계: 을 작성합니다 TridentOrchestrator **Trident**를 설치합니다

이제 를 만들 수 있습니다 TridentOrchestrator Astra Trident를 설치합니다. 원하는 경우 더 추가할 수 있습니다 **"Trident 설치를 사용자 지정합니다"**의 속성을 사용합니다 TridentOrchestrator 사양 다음 예에서는 Trident 및 CSI 이미지가 다른 레지스트리에 있는 설치를 보여 줍니다.

```

kubect1 create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

kubect1 describe torc trident

Name:          trident
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Spec:
  Autosupport Image: <your-registry>/netapp/trident-autosupport:23.01
  Debug:              true
  Image Registry:     <your-registry>/sig-storage
  Namespace:          trident
  Trident Image:      <your-registry>/netapp/trident:23.01.1
Status:
  Current Installation Params:
    IPv6:              false
    Autosupport Hostname:
    Autosupport Image: <your-registry>/netapp/trident-
autosupport:23.01
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:              true
    Http Request Timeout: 90s
    Image Pull Secrets:
    Image Registry:     <your-registry>/sig-storage
    k8sTimeout:         30
    Kubelet Dir:        /var/lib/kubelet
    Log Format:          text
    Probe Port:         17546
    Silence Autosupport: false
    Trident Image:      <your-registry>/netapp/trident:23.01.1
  Message:            Trident installed
  Namespace:          trident
  Status:              Installed
  Version:             v23.01.1
Events:
  Type Reason Age From Message ---- -
-----
Normal
Installing 74s trident-operator.netapp.io Installing Trident Normal
Installed 67s trident-operator.netapp.io Trident installed

```

설치를 확인합니다

설치를 확인하는 방법에는 여러 가지가 있습니다.

사용 TridentOrchestrator 상태

의 상태입니다 TridentOrchestrator 설치가 성공적으로 완료되었는지 여부를 나타내고 설치된 Trident의 버전을 표시합니다. 설치하는 동안 의 상태입니다 TridentOrchestrator 변경 시작 Installing 를 선택합니다 Installed. 을(를) 관찰하면 Failed 상태 및 운영자가 자체적으로 복구할 수 없습니다. ["로그를 확인합니다"](#).

상태	설명
설치 중	이 옵션을 사용하여 Astra Trident를 설치합니다 TridentOrchestrator 있습니다.
설치되어 있습니다	Astra Trident가 성공적으로 설치되었습니다.
제거 중	그 이유는 운영자가 Astra Trident를 제거하는 중입니다 spec.uninstall=true.
제거되었습니다	Astra Trident가 제거되었습니다.
실패했습니다	운영자가 Astra Trident를 설치, 패치, 업데이트 또는 제거할 수 없습니다. 이 상태에서 자동으로 복구를 시도합니다. 이 상태가 지속되면 문제 해결이 필요합니다.
업데이트 중	운영자가 기존 설치를 업데이트하고 있습니다.
오류	를 클릭합니다 TridentOrchestrator 사용되지 않습니다. 다른 파일이 이미 있습니다.

POD 생성 상태 사용

생성된 Pod의 상태를 검토하여 Astra Trident 설치가 완료되었는지 확인할 수 있습니다.

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
AGE			
trident-controller-7d466bf5c7-v4cpw	6/6	Running	0
1m			
trident-node-linux-mr6zc	2/2	Running	0
1m			
trident-node-linux-xrp7w	2/2	Running	0
1m			
trident-node-linux-zh2jt	2/2	Running	0
1m			
trident-operator-766f7b8658-ldzsv	1/1	Running	0
3m			

사용 `tridentctl`

을 사용할 수 있습니다 `tridentctl` 설치된 Astra Trident의 버전을 확인합니다.

```
./tridentctl -n trident version

+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 23.01.1        | 23.01.1        |
+-----+-----+
```

다음 단계

이제 가능합니다 "백엔드 및 스토리지 클래스를 생성하고, 볼륨을 프로비저닝하고, POD에 볼륨을 마운트합니다".

H제어(표준 모드)를 사용하여 **Trident** 연산자 배포

Trident 연산자를 구축하고 Hrom을 사용하여 Astra Trident를 설치할 수 있습니다. 이 프로세스는 Astra Trident에 필요한 컨테이너 이미지가 개인 레지스트리에 저장되어 있지 않은 설치에 적용됩니다. 개인 이미지 레지스트리가 있는 경우 를 사용합니다 "오프라인 배포를 위한 프로세스입니다".

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

** 중요 정보 Astra Trident **

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 입니다
`find_multipaths: no` 다중 경로 .conf 파일

비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart`
`multipath.conf` 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다
`find_multipaths: no` 21.07 릴리스 이후.

Trident 연산자를 구축하고 **Hrom**을 사용하여 **Astra Trident**를 설치합니다

Trident 사용 "**Helm 차트**" Trident 연산자를 구축하고 Trident를 한 번에 설치할 수 있습니다.

검토 "**설치 개요**" 설치 사전 요구 사항을 충족하고 환경에 맞는 올바른 설치 옵션을 선택했는지 확인합니다.

시작하기 전에

또한 "**구축 사전 요구 사항**" 필요한 것입니다 "**Helm 버전 3**".

단계

1. Astra Trident Helm 리포지토리를 추가합니다.

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

2. 사용 `helm install` 다음 예제에서와 같이 배포 이름을 지정합니다 23.01.1 는 설치 중인 Astra Trident의 버전입니다.

```
helm install <name> netapp-trident/trident-operator --version 23.01.1  
--create-namespace --namespace <trident-namespace>
```



Trident에 대한 네임스페이스를 이미 만든 경우 를 참조하십시오 --create-namespace 매개 변수는 추가 네임스페이스를 만들지 않습니다.

을 사용할 수 있습니다 `helm list` 이름, 네임스페이스, 차트, 상태, 앱 버전과 같은 설치 세부 정보를 검토하려면 수정본 번호.

설치 중에 구성 데이터를 전달합니다

설치 중에 구성 데이터를 전달하는 방법에는 두 가지가 있습니다.

옵션을 선택합니다	설명
<code>--values</code> (또는 <code>-f</code>)	재정의가 있는 YAML 파일을 지정합니다. 이 옵션은 여러 번 지정할 수 있으며 가장 오른쪽 파일이 우선 적용됩니다.
<code>--set</code>	명령줄에 overrides를 지정합니다.

예를 들어, 의 기본값을 변경합니다 `debug``에서 다음을 실행합니다 `--set` 명령 위치 23.01.1 설치 중인 Astra Trident의 버전입니다.

```
helm install <name> netapp-trident/trident-operator --version 23.01.1  
--create-namespace --namespace --set tridentDebug=true
```

구성 옵션

이 표와 `values.yaml` 제어 차트의 일부인 파일 에는 키 목록과 해당 기본값이 나와 있습니다.

옵션을 선택합니다	설명	기본값
<code>nodeSelector</code>	POD 할당을 위한 노드 레이블입니다	
<code>podAnnotations</code>	창 주석	
<code>deploymentAnnotations</code>	배포 주석	

옵션을 선택합니다	설명	기본값
tolerations	POD 지정에 대한 공차	
affinity	POD 할당에 대한 선호도	
tridentControllerPluginNodeSelector	Pod용 추가 노드 선택기를 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentControllerPluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentNodePluginNodeSelector	Pod용 추가 노드 선택기를 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentNodePluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
imageRegistry	에 대한 레지스트리를 식별합니다 trident-operator, `trident` 및 기타 이미지. 기본값을 그대로 사용하려면 비워 두십시오.	""
imagePullPolicy	에 대한 이미지 풀 정책을 설정합니다 trident-operator.	IfNotPresent
imagePullSecrets	의 이미지 풀 비밀을 설정합니다 trident-operator, `trident` 및 기타 이미지.	
kubeletDir	kubelet 내부 상태의 호스트 위치를 재정의할 수 있습니다.	"/var/lib/kubelet"
operatorLogLevel	Trident 연산자의 로그 수준을 다음으로 설정할 수 있습니다. trace, debug, info, warn, error, 또는 fatal.	"info"
operatorDebug	Trident 연산자의 로그 수준을 디버깅으로 설정할 수 있습니다.	true
operatorImage	에 대한 이미지를 완전히 재정의할 수 있습니다 trident-operator.	""
operatorImageTag	의 태그를 재정의할 수 있습니다 trident-operator 이미지.	""
tridentIPv6	Astra Trident가 IPv6 클러스터에서 작동하도록 허용합니다.	false
tridentK8sTimeout	대부분의 Kubernetes API 작업에 대한 기본 30초 시간 초과(0이 아닌 경우 초)를 재정의합니다.	0

옵션을 선택합니다	설명	기본값
tridentHttpRequestTimeout	에서는 HTTP 요청에 대한 기본 90초 제한 시간을 재정의합니다 0s 제한 시간 동안 무한 지속 시간입니다. 음수 값은 허용되지 않습니다.	"90s"
tridentSilenceAutosupport	Astra Trident Periodic AutoSupport 보고를 비활성화할 수 있습니다.	false
tridentAutosupportImageTag	Astra Trident AutoSupport 컨테이너의 이미지 태그를 재정의할 수 있습니다.	<version>
tridentAutosupportProxy	Astra Trident AutoSupport 컨테이너가 HTTP 프록시를 통해 집에 전화를 걸 수 있도록 허용합니다.	""
tridentLogFormat	Astra Trident 로깅 형식을 설정합니다 (text 또는 json)를 클릭합니다.	"text"
tridentDisableAuditLog	Astra Trident 감사 로거를 비활성화합니다.	true
tridentLogLevel	Astra Trident의 로그 수준을 다음과 같이 설정할 수 있습니다. trace, debug, info, warn, error, 또는 fatal.	"info"
tridentDebug	Astra Trident의 로그 수준을 로 설정할 수 있습니다 debug.	false
tridentLogWorkflows	추적 로깅 또는 로그 억제를 위해 특정 Astra Trident 워크플로우를 활성화할 수 있습니다.	""
tridentLogLayers	추적 로깅 또는 로그 억제를 위해 특정 Astra Trident 계층을 활성화할 수 있습니다.	""
tridentImage	Astra Trident의 이미지를 완전히 재정의할 수 있습니다.	""
tridentImageTag	Astra Trident에 대한 이미지 태그를 재정의할 수 있습니다.	""
tridentProbePort	Kubernetes 활성/준비 프로브에 사용되는 기본 포트를 재정의할 수 있습니다.	""
windows	Windows 작업자 노드에 Astra Trident를 설치할 수 있습니다.	false
enableForceDetach	힘 분리 기능을 활성화합니다.	false
excludePodSecurityPolicy	운영자 POD 보안 정책을 생성할 수 없습니다.	false

컨트롤러 Pod 및 노드 포드 이해

Astra Trident는 단일 컨트롤러 POD와 클러스터의 각 작업자 노드에 노드 POD를 더한 형태로 실행됩니다. Astra Trident 볼륨을 마운트하려는 호스트에서 노드 포드가 실행되고 있어야 합니다.

쿠버네티스 "노드 선택기" 및 "관용과 오해" 포드를 특정 노드 또는 기본 노드에서 실행하도록 제한하는 데 사용됩니다. ControllerPlugin과 을 사용합니다 `NodePlugin`구속 조건과 덮어쓰기를 지정할 수 있습니다.

- 컨트롤러 플러그인은 스냅샷 및 크기 조정과 같은 볼륨 프로비저닝 및 관리를 처리합니다.
- 노드 플러그인은 스토리지에 노드를 연결하는 작업을 처리합니다.

다음 단계

이제 가능합니다 "백엔드 및 스토리지 클래스를 생성하고, 볼륨을 프로비저닝하고, POD에 볼륨을 마운트합니다".

H제어(오프라인 모드)를 사용하여 Trident 연산자 배포

Trident 연산자를 구축하고 Hrom을 사용하여 Astra Trident를 설치할 수 있습니다. 이 프로세스는 Astra Trident에 필요한 컨테이너 이미지가 개인 레지스트리에 저장된 설치에 적용됩니다. 개인 이미지 레지스트리가 없는 경우 를 사용합니다 "표준 배포 프로세스".

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

 중요 정보 Astra Trident

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 입니다
`find_multipaths: no` 다중 경로 .conf 파일

비 경로 다중화 구성 또는 의 사용 `find_multipaths: yes` 또는 `find_multipaths: smart`
multipath.conf 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다
`find_multipaths: no` 21.07 릴리스 이후.

Trident 연산자를 구축하고 Hrom을 사용하여 Astra Trident를 설치합니다

Trident 사용 "Helm 차트" Trident 연산자를 구축하고 Trident를 한 번에 설치할 수 있습니다.

검토 "설치 개요" 설치 사전 요구 사항을 충족하고 환경에 맞는 올바른 설치 옵션을 선택했는지 확인합니다.

시작하기 전에

또한 "구축 사전 요구 사항" 필요한 것입니다 "Helm 버전 3".

단계

1. Astra Trident Helm 리포지토리를 추가합니다.

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

2. 사용 `helm install` 배포 및 이미지 레지스트리 위치의 이름을 지정합니다. 귀사의 "Trident 및 CSI 이미지" 하나의 레지스트리 또는 다른 레지스트리에 있을 수 있지만 모든 CSI 이미지는 동일한 레지스트리에 있어야 합니다. 예를 들어, 23.01.1 는 설치 중인 Astra Trident의 버전입니다.

하나의 레지스트리에 있는 이미지

```
helm install <name> netapp-trident/trident-operator --version  
23.01.1 --set imageRegistry=<your-registry> --create-namespace  
--namespace <trident-namespace>
```

다른 레지스트리의 이미지

추가해야 합니다 `sig-storage` 를 누릅니다 `imageRegistry` 다른 레지스트리 위치를 사용합니다.

```
helm install <name> netapp-trident/trident-operator --version  
23.01.1 --set imageRegistry=<your-registry>/sig-storage --set  
operatorImage=<your-registry>/netapp/trident-operator:23.01.1 --set  
tridentAutosupportImage=<your-registry>/netapp/trident-  
autosupport:23.01 --set tridentImage=<your-  
registry>/netapp/trident:23.01.1 --create-namespace --namespace  
<trident-namespace>
```



Trident에 대한 네임스페이스를 이미 만든 경우 를 참조하십시오 `--create-namespace` 매개 변수는 추가 네임스페이스를 만들지 않습니다.

을 사용할 수 있습니다 `helm list` 이름, 네임스페이스, 차트, 상태, 앱 버전과 같은 설치 세부 정보를 검토하려면 수정본 번호.

설치 중에 구성 데이터를 전달합니다

설치 중에 구성 데이터를 전달하는 방법에는 두 가지가 있습니다.

옵션을 선택합니다	설명
<code>--values</code> (또는 <code>-f</code>)	재정의가 있는 YAML 파일을 지정합니다. 이 옵션은 여러 번 지정할 수 있으며 가장 오른쪽 파일이 우선 적용됩니다.
<code>--set</code>	명령줄에 overrides를 지정합니다.

예를 들어, 의 기본값을 변경합니다 `debug``에서 다음을 실행합니다 `--set` 명령 위치 23.01.1 설치 중인 Astra Trident의 버전입니다.

```
helm install <name> netapp-trident/trident-operator --version 23.01.1
--create-namespace --namespace --set tridentDebug=true
```

구성 옵션

이 표와 values.yaml 제어 차트의 일부인 파일 에는 키 목록과 해당 기본값이 나와 있습니다.

옵션을 선택합니다	설명	기본값
nodeSelector	POD 할당을 위한 노드 레이블입니다	
podAnnotations	창 주석	
deploymentAnnotations	배포 주석	
tolerations	POD 지정에 대한 공차	
affinity	POD 할당에 대한 선호도	
tridentControllerPluginNodeSelector	Pod용 추가 노드 선택기 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentControllerPluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentNodePluginNodeSelector	Pod용 추가 노드 선택기 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
tridentNodePluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. 을 참조하십시오 컨트롤러 Pod 및 노드 포드 이해 를 참조하십시오.	
imageRegistry	에 대한 레지스트리를 식별합니다 trident-operator, `trident` 및 기타 이미지. 기본값을 그대로 사용하려면 비워 두십시오.	""
imagePullPolicy	에 대한 이미지 풀 정책을 설정합니다 trident-operator.	IfNotPresent
imagePullSecrets	의 이미지 풀 비밀을 설정합니다 trident-operator, `trident` 및 기타 이미지.	
kubeletDir	kubelet 내부 상태의 호스트 위치를 재정의할 수 있습니다.	"/var/lib/kubelet"
operatorLogLevel	Trident 연산자의 로그 수준을 다음으로 설정할 수 있습니다. trace, debug, info, warn, error, 또는 fatal.	"info"

옵션을 선택합니다	설명	기본값
operatorDebug	Trident 연산자의 로그 수준을 디버깅으로 설정할 수 있습니다.	true
operatorImage	에 대한 이미지를 완전히 재정의할 수 있습니다 trident-operator.	""
operatorImageTag	의 태그를 재정의할 수 있습니다 trident-operator 이미지.	""
tridentIPv6	Astra Trident가 IPv6 클러스터에서 작동하도록 허용합니다.	false
tridentK8sTimeout	대부분의 Kubernetes API 작업에 대한 기본 30초 시간 초과(0이 아닌 경우 초)를 재정의합니다.	0
tridentHttpRequestTimeout	에서는 HTTP 요청에 대한 기본 90초 제한 시간을 재정의합니다 0s 제한 시간 동안 무한 지속 시간입니다. 음수 값은 허용되지 않습니다.	"90s"
tridentSilenceAutosupport	Astra Trident Periodic AutoSupport 보고를 비활성화할 수 있습니다.	false
tridentAutosupportImageTag	Astra Trident AutoSupport 컨테이너의 이미지 태그를 재정의할 수 있습니다.	<version>
tridentAutosupportProxy	Astra Trident AutoSupport 컨테이너가 HTTP 프록시를 통해 집에 전화를 걸 수 있도록 허용합니다.	""
tridentLogFormat	Astra Trident 로깅 형식을 설정합니다 (text 또는 json)를 클릭합니다.	"text"
tridentDisableAuditLog	Astra Trident 감사 로거를 비활성화합니다.	true
tridentLogLevel	Astra Trident의 로그 수준을 다음과 같이 설정할 수 있습니다. trace, debug, info, warn, error, 또는 fatal.	"info"
tridentDebug	Astra Trident의 로그 수준을 로 설정할 수 있습니다 debug.	false
tridentLogWorkflows	추적 로깅 또는 로그 억제를 위해 특정 Astra Trident 워크플로우를 활성화할 수 있습니다.	""
tridentLogLayers	추적 로깅 또는 로그 억제를 위해 특정 Astra Trident 계층을 활성화할 수 있습니다.	""
tridentImage	Astra Trident의 이미지를 완전히 재정의할 수 있습니다.	""
tridentImageTag	Astra Trident에 대한 이미지 태그를 재정의할 수 있습니다.	""

옵션을 선택합니다	설명	기본값
tridentProbePort	Kubernetes 활성/준비 프로브에 사용되는 기본 포트를 재정의할 수 있습니다.	""
windows	Windows 작업자 노드에 Astra Trident를 설치할 수 있습니다.	false
enableForceDetach	힘 분리 기능을 활성화합니다.	false
excludePodSecurityPolicy	운영자 POD 보안 정책을 생성할 수 없습니다.	false

컨트롤러 Pod 및 노드 포드 이해

Astra Trident는 단일 컨트롤러 POD와 클러스터의 각 작업자 노드에 노드 POD를 더한 형태로 실행됩니다. Astra Trident 볼륨을 마운트하려는 호스트에서 노드 포드가 실행되고 있어야 합니다.

쿠버네티스 "[노드 선택기](#)" 및 "[관용과 오해](#)" 포드를 특정 노드 또는 기본 노드에서 실행하도록 제한하는 데 사용됩니다. ControllerPlugin과 을 사용합니다 `NodePlugin`구속 조건과 덮어쓰기를 지정할 수 있습니다.

- 컨트롤러 플러그인은 스냅샷 및 크기 조정과 같은 볼륨 프로비저닝 및 관리를 처리합니다.
- 노드 플러그인은 스토리지에 노드를 연결하는 작업을 처리합니다.

다음 단계

이제 가능합니다 "[백엔드 및 스토리지 클래스를 생성하고, 볼륨을 프로비저닝하고, POD에 볼륨을 마운트합니다](#)".

Trident 운영자 설치를 사용자 지정합니다

Trident 운영자는 의 특성을 사용하여 Astra Trident 설치를 사용자 지정할 수 있습니다 TridentOrchestrator 사양 설치를 사용자 지정하려면 다음을 선택합니다 TridentOrchestrator 인수를 사용할 수 있습니다. 을 사용하는 것이 좋습니다 tridentctl 필요에 따라 수정할 사용자 지정 YAML 매니페스트를 생성합니다.

컨트롤러 Pod 및 노드 포드 이해

Astra Trident는 단일 컨트롤러 POD와 클러스터의 각 작업자 노드에 노드 POD를 더한 형태로 실행됩니다. Astra Trident 볼륨을 마운트하려는 호스트에서 노드 포드가 실행되고 있어야 합니다.

쿠버네티스 "[노드 선택기](#)" 및 "[관용과 오해](#)" 포드를 특정 노드 또는 기본 노드에서 실행하도록 제한하는 데 사용됩니다. ControllerPlugin과 을 사용합니다 `NodePlugin`구속 조건과 덮어쓰기를 지정할 수 있습니다.

- 컨트롤러 플러그인은 스냅샷 및 크기 조정과 같은 볼륨 프로비저닝 및 관리를 처리합니다.
- 노드 플러그인은 스토리지에 노드를 연결하는 작업을 처리합니다.

구성 옵션



spec.namespace 에 지정됩니다 TridentOrchestrator Astra Trident가 설치된 네임스페이스를 나타냅니다. Astra Trident가 설치된 후에는 이 매개 변수 * 를 업데이트할 수 없습니다. 이렇게 하려고 하면 가 발생합니다 TridentOrchestrator 변경할 상태입니다 Failed. Astra Trident는 네임스페이스 간에 마이그레이션되지 않습니다.

이 표에 자세히 나와 있습니다 TridentOrchestrator 속성.

매개 변수	설명	기본값
namespace	Astra Trident를 설치할 네임스페이스입니다	"기본값"
debug	Astra Trident에 대한 디버깅을 활성화합니다	거짓
windows	를 로 설정합니다 true Windows 작업자 노드에 설치할 수 있습니다.	거짓
IPv6	IPv6를 통해 Astra Trident를 설치합니다	거짓
k8sTimeout	Kubernetes 작업 시간이 초과되었습니다	30초
silenceAutosupport	AutoSupport 번들을 NetApp에 자동으로 보내지 않습니다	거짓
enableNodePrep	작업자 노드 종속성 자동 관리(* beta*)	거짓
autosupportImage	AutoSupport 텔레메트리 컨테이너 이미지입니다	"NetApp/트리덴트 - AutoSupport: 23.01"
autosupportProxy	AutoSupport 텔레메트리 전송을 위한 프록시의 주소/포트입니다	"http://proxy.example.com:8888""
uninstall	Astra Trident를 제거하는 데 사용되는 플래그입니다	거짓
logFormat	사용할 Astra Trident 로깅 형식[text,json]	"텍스트"
tridentImage	설치할 Astra Trident 이미지	"NetApp/트리덴트: 21.04"
imageRegistry	형식의 내부 레지스트리 경로입니다 <registry FQDN>[:port] [/subpath]	"k8s.gcr.io/sig-storage(k8s 1.19 이상) 또는 quay.io/k8scloud" 또는 quay.io/k8scloud"
kubeletDir	호스트의 kubelet 디렉토리에 대한 경로입니다	"/var/lib/kubelet"
wipeout	Astra Trident를 완전히 제거하기 위해 삭제할 리소스 목록입니다	
imagePullSecrets	내부 레지스트리에서 이미지를 가져올 수 있는 비밀	

매개 변수	설명	기본값
imagePullPolicy	Trident 운영자의 이미지 풀 정책을 설정합니다. 유효한 값은 다음과 같습니다. Always 항상 이미지를 당깁니다. IfNotPresent 이미지가 아직 노드에 없는 경우에만 이미지를 가져옵니다. Never 이미지를 당기지 않습니다.	IfNotPresent
controllerPluginNodeSelector	Pod용 추가 노드 선택기 pod.spec.nodeSelector 과 동일한 형식을 따릅니다.	기본값 없음, 선택 사항
controllerPluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. pod.spec.Tolerations와 같은 형식을 따릅니다.	기본값 없음, 선택 사항
nodePluginNodeSelector	Pod용 추가 노드 선택기 pod.spec.nodeSelector 과 동일한 형식을 따릅니다.	기본값 없음, 선택 사항
nodePluginTolerations	Pod에 대한 Kubernetes 허용 설정을 재정의합니다. pod.spec.Tolerations와 같은 형식을 따릅니다.	기본값 없음, 선택 사항



포드 매개 변수 포맷에 대한 자세한 내용은 을 참조하십시오 "[노드에 Pod 할당](#)".

샘플 구성

정의할 때 위에서 언급한 속성을 사용할 수 있습니다 TridentOrchestrator 를 눌러 설치를 사용자 정의합니다.

예 1: 기본 사용자 정의 구성

다음은 기본 사용자 지정 구성의 예입니다.

```
cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
    - thisisasecret
```

예 2: 노드 선택기를 사용하여 배포

이 예제에서는 노드 선택기를 사용하여 Trident를 배포하는 방법을 보여 줍니다.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  controllerPluginNodeSelector:
    nodetype: master
  nodePluginNodeSelector:
    storage: netapp
```

예 3: Windows 작업자 노드에 배포

이 예제에서는 Windows 작업자 노드에 대한 배포를 보여 줍니다.

```
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true
```

tridentctl을 사용하여 설치합니다

tridentctl을 사용하여 설치합니다

을 사용하여 Astra Trident를 설치할 수 있습니다 tridentctl. 이 프로세스는 Astra Trident에 필요한 컨테이너 이미지가 개인 레지스트리에 저장되거나 저장되지 않은 설치에 적용됩니다. 를 사용자 지정합니다 tridentctl 구축 방법은 을 참조하십시오 "[tridentctl 배포를 사용자 지정합니다](#)".

Astra Trident 23.01에 대한 중요 정보입니다

- Astra Trident * 에 대한 다음 중요 정보를 읽어야 합니다

 중요 정보 Astra Trident

- 이제 Trident에서 Kubernetes 1.26이 지원됩니다. Kubernetes를 업그레이드하기 전에 Trident를 업그레이드하십시오.
- Astra Trident는 SAN 환경에서 다중 경로 구성을 엄격하게 사용하며 권장 값은 입니다
find_multipaths: no 다중 경로 .conf 파일

비 경로 다중화 구성 또는 의 사용 find_multipaths: yes 또는 find_multipaths: smart multipath.conf 파일의 값으로 인해 마운트 오류가 발생합니다. Trident에서 의 사용을 권장했습니다
find_multipaths: no 21.07 릴리스 이후.

를 사용하여 **Astra Trident**를 설치합니다 tridentctl

검토 "[설치 개요](#)" 설치 사전 요구 사항을 충족하고 환경에 맞는 올바른 설치 옵션을 선택했는지 확인합니다.

시작하기 전에

설치를 시작하기 전에 Linux 호스트에 로그인하여 작업 관리 여부를 확인합니다. "[지원되는 Kubernetes 클러스터](#)" 필요한 권한이 있어야 합니다.



OpenShift에서는 을 사용합니다 oc 대신 kubectl 다음 모든 예에서 를 실행하여 먼저 * system:admin * 으로 로그인합니다 oc login -u system:admin 또는 oc login -u kube-admin.

1. Kubernetes 버전 확인:

```
kubectl version
```

2. 클러스터 관리자 권한 확인:

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. Docker Hub의 이미지를 사용하는 Pod를 시작하고 Pod 네트워크를 통해 스토리지 시스템에 연결할 수 있는지 확인합니다.

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
ping <management IP>
```

1단계: Trident 설치 프로그램 패키지를 다운로드합니다

Astra Trident 설치 프로그램 패키지는 Trident Pod를 생성하고 상태를 유지하는 데 사용되는 CRD 객체를 구성하며, CSI 사이드카를 초기화하여 클러스터 호스트에 볼륨 프로비저닝 및 연결과 같은 작업을 수행합니다. 에서 최신 버전의 Trident 설치 프로그램을 다운로드하고 압축을 풉니다 "[GitHub의 Assets_섹션](#)". [선택한 Astra Trident 버전을 사용하여](#)

예제에서 `update<trident-installer-XX.XX.X.tar.gz>_`를 선택합니다.

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

2단계: Astra Trident 설치

를 실행하여 원하는 네임스페이스에 Astra Trident를 설치합니다 `tridentctl install` 명령. 추가 인수를 추가하여 이미지 레지스트리 위치를 지정할 수 있습니다.



Astra Trident를 Windows 노드에서 실행하도록 설정하려면 `el`을 추가합니다 `--windows` 설치 명령에 플래그 지정: `$./tridentctl install --windows -n trident.`

표준 모드

```
./tridentctl install -n trident
```

하나의 레지스트리에 있는 이미지

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:23.01 --trident
-image <your-registry>/trident:23.01.1
```

다른 레지스트리의 이미지

추가해야 합니다 `sig-storage` 를 누릅니다 `imageRegistry` 다른 레지스트리 위치를 사용합니다.

```
./tridentctl install -n trident --image-registry <your-registry>/sig-
storage --autosupport-image <your-registry>/netapp/trident-
autosupport:23.01 --trident-image <your-
registry>/netapp/trident:23.01.1
```

설치 상태는 다음과 같습니다.

```

.....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                        namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.              version=23.01.1
INFO Trident installation succeeded.
.....

```

설치를 확인합니다

POD 생성 상태 또는 `tridentctl` 를 사용하여 설치를 확인할 수 있습니다 `tridentctl`.

POD 생성 상태 사용

생성된 Pod의 상태를 검토하여 Astra Trident 설치가 완료되었는지 확인할 수 있습니다.

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS	AGE
trident-controller-679648bd45-cv2mx	6/6	Running	0	5m29s
trident-node-linux-vgc8n	2/2	Running	0	5m29s



설치 프로그램이 성공적으로 완료되지 않거나 `trident-controller-<generated id>` (`trident-csi-<generated id>` 23.01 이전 버전에서는 *Running* 상태가 없으며 플랫폼이 설치되지 않았습니다. 사용 `-d` 를 선택합니다 "디버그 모드를 켭니다" 문제를 해결합니다.

사용 `tridentctl`

을 사용할 수 있습니다 `tridentctl` 설치된 Astra Trident의 버전을 확인합니다.

```
./tridentctl -n trident version
```

```
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 23.01.1        | 23.01.1        |
+-----+-----+
```

다음 단계

이제 가능합니다 "백엔드 및 스토리지 클래스를 생성하고, 볼륨을 프로비저닝하고, POD에 볼륨을 마운트합니다".

tridentctl 설치를 사용자 지정합니다

Astra Trident 설치 프로그램을 사용하여 설치를 사용자 지정할 수 있습니다.

설치 프로그램에 대해 알아보십시오

Astra Trident 설치 프로그램을 사용하여 특성을 사용자 지정할 수 있습니다. 예를 들어, Trident 이미지를 개인 저장소에 복사한 경우 `trident-image`를 사용하여 이미지 이름을 지정할 수 있습니다. Trident 이미지와 필요한 CSI 사이드카 이미지를 개인 저장소에 복사한 경우 `image-registry` 스위치를 누릅니다 `<registry FQDN>[:port]`.

Kubernetes 배포를 사용 중인 경우 `kubelet` 일반적인 경로 이외의 경로에 데이터를 보관합니다 `/var/lib/kubelet`, 을 사용하여 대체 경로를 지정할 수 있습니다 `--kubelet-dir`.

설치 관리자의 인수 이외에 설치를 사용자 지정해야 하는 경우 배포 파일을 사용자 지정할 수도 있습니다. `tridentctl`를 사용합니다 `--generate-custom-yaml` 매개 변수는 설치 관리자의 `setup` 디렉터리:

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`
- `trident-service.yaml`
- `trident-namespace.yaml`
- `trident-serviceaccount.yaml`
- `trident-resourcequota.yaml`

이러한 파일을 생성한 후 필요에 따라 수정한 다음 `tridentctl`을 사용할 수 있습니다 `--use-custom-yaml` 사용자 지정 배포를 설치합니다.

```
./tridentctl install -n trident --use-custom-yaml
```

다음 단계

Astra Trident를 설치한 후 백엔드 생성, 스토리지 클래스 생성, 볼륨 프로비저닝 및 POD에 볼륨 마운팅을 진행할 수 있습니다.

1단계: 백엔드를 생성합니다

이제 Astra Trident에서 볼륨을 프로비저닝하는 데 사용할 백엔드를 만들 수 있습니다. 이렇게 하려면 을 만듭니다 backend.json 필요한 매개 변수가 포함된 파일입니다. 다양한 백엔드 유형에 대한 샘플 구성 파일은 에서 찾을 수 있습니다 sample-input 디렉토리.

을 참조하십시오 ["여기"](#) 백엔드 유형에 맞게 파일을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오.

```
cp sample-input/<backend template>.json backend.json
vi backend.json
```

```
./tridentctl -n trident create backend -f backend.json
+-----+-----+-----+-----+
+-----+-----+
|   NAME   | STORAGE DRIVER |           UUID           |
STATE | VOLUMES |
+-----+-----+-----+-----+
+-----+-----+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |         0 |
+-----+-----+-----+-----+
+-----+-----+
```

생성에 실패하면 백엔드 구성에 문제가 있는 것입니다. 다음 명령을 실행하여 로그를 보고 원인을 확인할 수 있습니다.

```
./tridentctl -n trident logs
```

문제를 해결한 후 이 단계의 처음으로 돌아가서 다시 시도하십시오. 자세한 문제 해결 팁은 를 참조하십시오 ["문제 해결"](#) 섹션을 참조하십시오.

2단계: 스토리지 클래스를 생성합니다

Kubernetes 사용자는 가 지정된 영구 PVC(Volume Claim)를 사용하여 볼륨을 프로비저닝합니다 ["스토리지 클래스"](#) 이름별. 세부 정보는 사용자로부터 숨겨지지만 스토리지 클래스는 해당 클래스에 사용되는 공급자(이 경우 Trident)와 해당 클래스가 프로비저닝자로부터 의미하는 바를 식별합니다.

Kubernetes 스토리지 클래스를 생성할 때 볼륨을 지정할 시기를 지정하십시오. 수업 구성에서는 이전 단계에서 생성한 백엔드를 모델링해야 하므로 Astra Trident가 이를 사용하여 새 볼륨을 프로비저닝합니다.

가장 간단한 스토리지 클래스는 을 기반으로 합니다 `sample-input/storage-class-csi.yaml.template` 설치 프로그램과 함께 제공되는 파일로 대체합니다 `BACKEND_TYPE` 스토리지 드라이버 이름을 사용합니다.

```
./tridentctl -n trident get backend
+-----+-----+-----+
+-----+
|   NAME   | STORAGE DRIVER |           UUID           |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |         0 |
+-----+-----+-----+
+-----+-----+

cp sample-input/storage-class-csi.yaml.template sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

Kubernetes 오브젝트이므로 를 사용할 수 있습니다 `kubectl` Kubernetes에서 생성해야 합니다.

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

이제 Kubernetes 및 Astra Trident에 * basic-CSI * 스토리지 클래스가 표시됩니다. Astra Trident는 백엔드에서 폴을 검색했습니다.

```

kubect1 get sc basic-csi
NAME          PROVISIONER          AGE
basic-csi     csi.trident.netapp.io 15h

./tridentctl -n trident get storageclass basic-csi -o json
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}

```

3단계: 첫 번째 볼륨을 프로비저닝합니다

이제 첫 번째 볼륨을 동적으로 프로비저닝할 준비가 되었습니다. 이 작업은 Kubernetes를 생성하여 수행합니다 **"영구적 볼륨 클레임"** (PVC) 개체.

방금 만든 저장소 클래스를 사용하는 볼륨에 대해 PVC를 생성합니다.

을 참조하십시오 `sample-input/pvc-basic-csi.yaml` 예를 들어, 스토리지 클래스 이름이 생성한 이름과 일치하는지 확인합니다.

```
kubectl create -f sample-input/pvc-basic-csi.yaml
```

```
kubectl get pvc --watch
```

NAME	STATUS	VOLUME	CAPACITY
basic	Pending		
basic	1s		
basic	Pending	pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7	0
basic	5s		
basic	Bound	pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7	1Gi
RWO	basic	7s	

4단계: POD에 볼륨을 마운트합니다

이제 볼륨을 마운트하겠습니다. PV를 장착하는 nginx 포드를 시작합니다 /usr/share/nginx/html.

```
cat << EOF > task-pv-pod.yaml
```

```
kind: Pod
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: task-pv-pod
```

```
spec:
```

```
  volumes:
```

```
    - name: task-pv-storage
```

```
      persistentVolumeClaim:
```

```
        claimName: basic
```

```
  containers:
```

```
    - name: task-pv-container
```

```
      image: nginx
```

```
      ports:
```

```
        - containerPort: 80
```

```
          name: "http-server"
```

```
      volumeMounts:
```

```
        - mountPath: "/usr/share/nginx/html"
```

```
          name: task-pv-storage
```

```
EOF
```

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

```
# Wait for the pod to start
kubectl get pod --watch

# Verify that the volume is mounted on /usr/share/nginx/html
kubectl exec -it task-pv-pod -- df -h /usr/share/nginx/html

# Delete the pod
kubectl delete pod task-pv-pod
```

이때 POD(애플리케이션)는 더 이상 존재하지 않지만 볼륨은 여전히 존재합니다. 원하는 경우 다른 포드에서 사용할 수 있습니다.

볼륨을 삭제하려면 클레임을 삭제합니다.

```
kubectl delete pvc basic
```

이제 다음과 같은 추가 작업을 수행할 수 있습니다.

- "추가 백엔드를 구성합니다."
- "추가 스토리지 클래스를 생성합니다."

Astra Trident 관리

Astra Trident를 업그레이드합니다

Astra Trident를 업그레이드합니다

Astra Trident는 분기별 릴리스 케이던스를 따르며, 매년 4개의 주요 릴리즈를 제공합니다. 각 새로운 릴리스는 이전 릴리즈를 기반으로 하며 새로운 기능과 성능 향상, 버그 수정 및 개선 기능을 제공합니다. Astra Trident의 새로운 기능을 이용하려면 1년에 한 번 이상 업그레이드하시기 바랍니다.

버전을 선택합니다

Astra Trident 버전은 날짜를 기반으로 합니다 YY.MM 이름 지정 규칙. 여기서 "YY"는 연도의 마지막 두 자리이고 "MM"은 월입니다. DOT 릴리스는 a를 따릅니다 YY.MM.X 규칙. 여기서 "X"는 패치 수준입니다. 업그레이드할 버전에 따라 업그레이드할 버전을 선택합니다.

- 설치된 버전의 4 릴리스 창 내에 있는 모든 대상 릴리스에 대해 직접 업그레이드를 수행할 수 있습니다. 예를 들어 22.01에서 23.01로 직접 업그레이드할 수 있습니다(22.01.1과 같은 도트 릴리스 포함).
- 이전 버전이 설치되어 있는 경우, 특정 지침은 해당 릴리스의 설명서를 사용하여 다단계 업그레이드를 수행해야 합니다. 따라서 먼저 4개의 릴리스 창에 맞는 최신 릴리즈로 업그레이드해야 합니다. 예를 들어 18.07을 실행하고 20.07 릴리스로 업그레이드하려는 경우 다음과 같이 다단계 업그레이드 프로세스를 따르십시오.
 - a. 첫 번째 업그레이드는 18.07에서 19.07로 가능합니다.
 - b. 그런 다음 19.07에서 20.07로 업그레이드합니다.



- 버전 19.04 및 이전 버전의 모든 업그레이드를 수행하려면 IT에서 Astra Trident 메타데이터를 마이그레이션해야 합니다 etcd CRD 개체. 업그레이드 작동 방식을 이해하려면 릴리즈 설명서를 확인하십시오.
- 업그레이드할 때 제공하는 것이 중요합니다 parameter.fsType 인치 StorageClasses Astra Trident에서 사용 삭제하고 다시 만들 수 있습니다 StorageClasses 기존 볼륨을 그대로 사용합니다. 이것은 SAN 볼륨에 대해 security 컨텍스트를 적용하기 위한 ** 요구 사항입니다. sample input 디렉토리에는 <https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templ> 같은 예제가 포함되어 있습니다[storage-class-basic.yaml.templ] 및 링크: storage-class-bronze-default.yaml. 자세한 내용은 을 참조하십시오 "알려진 문제".

업그레이드 옵션을 선택합니다

Astra Trident를 업그레이드할 수 있는 두 가지 옵션이 있습니다. 일반적으로 초기 설치에 사용한 옵션과 동일한 옵션을 사용할 수 있지만 "설치 방법 간에 이동합니다".

- "Trident 연산자를 사용하여 업그레이드합니다"

*



CSI 볼륨 스냅샷은 이제 Kubernetes 1.20부터 시작되는 GA 기능입니다. Astra Trident를 업그레이드할 때 업그레이드를 수행하기 전에 이전의 모든 알파 스냅샷 CRS 및 CRD(볼륨 스냅샷 클래스, 볼륨 스냅샷 및 볼륨 스냅샷 콘텐츠)를 제거해야 합니다. 을 참조하십시오 ["블로그입니다"](#) 알파 스냅샷을 베타/GA 스펙으로 마이그레이션하는 단계에 대해 이해합니다.

조작자에 대한 변경

Astra Trident의 21.01 릴리스에는 다음과 같은 몇 가지 주요 아키텍처 변경 사항이 작업자에게 도입되었습니다.

- 이제 연산자가 * 클러스터 범위 * 가 됩니다. Trident 연산자(버전 20.04 - 20.10)의 이전 인스턴스는 * 네임스페이스 범위 * 였습니다. 클러스터 범위의 연산자는 다음과 같은 이유로 유용합니다.
 - 리소스 책임: 이제 운영자는 클러스터 수준에서 Astra Trident 설치와 관련된 리소스를 관리합니다. Astra Trident를 설치하는 과정에서 운영자는 를 사용하여 여러 리소스를 생성하고 유지 관리합니다 ownerReferences. 유지 관리 ownerReferences 클러스터 범위 리소스의 경우 OpenShift와 같은 특정 Kubernetes 배포판에서 오류가 발생할 수 있습니다. 이 문제는 클러스터 범위 운영자를 통해 완화됩니다. Trident 리소스의 자동 복구 및 패칭은 필수 요구사항입니다.
 - 제거 중 정리: Astra Trident를 완전히 제거하려면 모든 관련 리소스를 삭제해야 합니다. 네임스페이스 범위 연산자는 클러스터 범위 리소스(예: clusterRole, ClusterRoleBinding 및 PodSecurityPolicy)를 제거하는 데 문제가 있을 수 있으며 불완전한 정리 작업을 초래할 수 있습니다. 클러스터 범위 연산자로 인해 이 문제가 발생하지 않습니다. 사용자는 Astra Trident를 완전히 제거하고 필요한 경우 새로 설치할 수 있습니다.
- TridentProvisioner 이(가) 이제 로 대체됩니다 TridentOrchestrator Astra Trident를 설치 및 관리하는 데 사용되는 사용자 지정 리소스입니다. 또한 에 새 필드가 도입되었습니다 TridentOrchestrator 사양 사용자는 네임스페이스 Trident가 을 사용하여 설치/업그레이드되도록 지정할 수 있습니다 spec.namespace 필드에 입력합니다. 예를 들어 보겠습니다 ["여기"](#).

운영자와 함께 업그레이드하십시오

운영자를 사용하여 기존 Astra Trident 설치를 쉽게 업그레이드할 수 있습니다.

시작하기 전에

연산자를 사용하여 업그레이드하려면 다음 조건이 충족되어야 합니다.

- CSI 기반 Astra Trident가 설치되어 있어야 합니다. 의 19.07에서 모든 릴리스는 CSI 기반입니다. Trident 네임스페이스에서 Pod를 검사하여 확인할 수 있습니다.
 - 23.01 이전 버전의 POD 이름 지정은 A 뒤에 옵니다 trident-csi-* 컨벤션.
 - 23.01 이상에서 포드 이름 지정 시 사용: trident-controller-<generated id> 컨트롤러 포드의 경우 trident-node-<operating system>-<generated id> 노드 포드의 경우, trident-operator-<generated id> 작업자 포드의 경우
- CSI Trident를 제거한 후 설치 시 메타데이터가 지속되는 경우 연산자를 사용하여 업그레이드할 수 있습니다.
- 주어진 Kubernetes 클러스터의 모든 네임스페이스에 하나의 Astra Trident 설치만 존재해야 합니다.
- 를 실행하는 Kubernetes 클러스터를 사용해야 합니다 ["지원되는 Kubernetes 버전"](#).
- 알파 스냅샷 CRD가 있는 경우 를 사용하여 제거해야 합니다 tridentctl obliviate alpha-snapshot-crd. 그러면 알파 스냅샷 사양에 대한 CRD가 삭제됩니다. 삭제/마이그레이션해야 하는 기존 스냅샷의 경우 를 참조하십시오 ["블로그입니다"](#).



- OpenShift Container Platform에서 연산자를 사용하여 Trident를 업그레이드할 때는 Trident 21.01.1 이상으로 업그레이드해야 합니다. 21.01.0으로 릴리스된 Trident 연산자에는 21.01.1에서 해결된 알려진 문제가 포함되어 있습니다. 자세한 내용은 를 참조하십시오 ["GitHub에 대한 발행 세부 정보"](#).
- 를 사용하는 경우 연산자를 사용하여 Trident를 업그레이드하지 마십시오 `etcd` 기반 Trident 릴리즈(19.04 이상)

클러스터 범위 **Trident** 운영자 설치를 업그레이드합니다

클러스터 범위 Trident 운영자 설치를 업그레이드하려면 다음 단계를 수행하십시오. Astra Trident 버전 21.01 이상에서는 클러스터 범위 연산자를 사용합니다.

단계

1. Astra Trident 버전 확인:

```
./tridentctl -n trident version
```

2. 현재 Astra Trident 인스턴스를 설치하는 데 사용된 Trident 연산자를 삭제합니다. 예를 들어, 22.01에서 업그레이드하는 경우 다음 명령을 실행합니다.

```
kubectl delete -f 22.01/trident-installer/deploy/bundle.yaml -n trident
```

3. 를 사용하여 초기 설치를 사용자 지정한 경우 `TridentOrchestrator` 속성을 편집할 수 있습니다 `TridentOrchestrator` 설치 매개 변수를 수정하는 개체입니다. 여기에는 오프라인 모드에 대해 미러링된 Trident 및 CSI 이미지 레지스트리를 지정하는 변경 사항, 디버그 로그 활성화 또는 이미지 풀 비밀을 지정하는 변경 사항이 포함될 수 있습니다.
4. 사용자 환경과 Astra Trident 버전에 맞는 올바른 번들 YAML 파일을 사용하여 Astra Trident를 설치합니다. 예를 들어, Kubernetes 1.26용 Astra Trident 23.01을 설치하는 경우 다음 명령을 실행합니다.

```
kubectl create -f 23.01.1/trident-installer/deploy/bundle_post_1_25.yaml -n trident
```

Trident는 운영자를 설치하고 Kubernetes 버전용 관련 개체를 생성하는 데 사용할 수 있는 번들 파일을 제공합니다.



- Kubernetes 1.24 이하 를 실행하는 클러스터의 경우, 를 사용합니다 ["Bundle_PRE_1_25.YAML"](#).
- Kubernetes 1.25 이상을 실행하는 클러스터의 경우 를 사용합니다 ["Bundle_post_1_25.YAML"](#).

결과

Trident 운영자는 기존 Astra Trident 설치를 식별하고 운영자와 동일한 버전으로 업그레이드합니다.

네임스페이스 범위 연산자 설치를 업그레이드합니다

네임스페이스 범위 연산자(버전 20.07 ~ 20.10)를 사용하여 설치된 Astra Trident의 인스턴스에서 업그레이드하려면 다음 단계를 수행하십시오.

단계

1. 기존 Trident 설치의 상태를 확인합니다. 이렇게 하려면 의 * 상태 * 를 확인하십시오 TridentProvisioner. 상태는 입니다 Installed.

```
kubectl describe tprov trident -n trident | grep Message: -A 3
Message:  Trident installed
Status:   Installed
Version:  v20.10.1
```



상태가 표시되는 경우 `Updating` 계속하기 전에 이 문제를 해결하십시오. 가능한 상태 값 목록은 를 참조하십시오 ["여기"](#).

2. 를 생성합니다 TridentOrchestrator Trident 설치 프로그램과 함께 제공된 매니페스트를 사용하여 CRD를 만듭니다.

```
# Download the release required [23.01.1]
mkdir 23.01.1
cd 23.01.1
wget
https://github.com/NetApp/trident/releases/download/v23.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. 매니페스트를 사용하여 네임스페이스 범위 연산자를 삭제합니다. 이 단계를 완료하려면 에서 네임스페이스 범위 연산자를 배포하는 데 사용되는 번들 YAML 파일이 필요합니다

<https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/BUNDLE.YAML> 위치 vXX.XX 은(는) 버전 번호입니다 BUNDLE.YAML 번들 YAML 파일 이름입니다.



Trident 설치 매개 변수를 필요에 따라 변경해야 합니다(예: 의 값 변경) tridentImage, autosupportImage `개인 이미지 저장소 및 제공 `imagePullSecrets) 네임스페이스 범위 연산자를 삭제한 후 클러스터 범위 연산자를 설치하기 전에 먼저 . 업데이트할 수 있는 전체 매개 변수 목록은 을 참조하십시오 ["구성 옵션"](#).


```

#Ensure you are in the right directory
pwd
/root/20.10.1/trident-installer

#Delete the namespace-scoped operator
kubectl delete -f deploy/<BUNDLE.YAML> -n trident
serviceaccount "trident-operator" deleted
clusterrole.rbac.authorization.k8s.io "trident-operator" deleted
clusterrolebinding.rbac.authorization.k8s.io "trident-operator" deleted
deployment.apps "trident-operator" deleted
podsecuritypolicy.policy "tridentoperatorpods" deleted

#Confirm the Trident operator was removed
kubectl get all -n trident

```

NAME	READY	STATUS	RESTARTS	AGE
pod/trident-csi-68d979fb85-dsrmn	6/6	Running	12	99d
pod/trident-csi-8jfhf	2/2	Running	6	105d
pod/trident-csi-jtnjz	2/2	Running	6	105d
pod/trident-csi-lcxvh	2/2	Running	8	105d

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/trident-csi	ClusterIP	10.108.174.125	<none>	34571/TCP,9220/TCP	105d

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AGE
daemonset.apps/trident-csi	3	3	3	3	3
kubernetes.io/arch=amd64,kubernetes.io/os=linux			105d		

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/trident-csi	1/1	1	1	105d

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/trident-csi-68d979fb85	1	1	1	105d

이 단계에서는 를 선택합니다 trident-operator-xxxxxxxxxx-xxxxx POD가 삭제되었습니다.

- (선택 사항) 설치 매개 변수를 수정해야 하는 경우 를 업데이트합니다 TridentProvisioner 사양 컨테이너 이미지를 가져올 전용 이미지 레지스트리를 수정하거나, 디버그 로그를 사용하거나, 이미지 풀 비밀을 지정하는 등의 변경 사항이 있을 수 있습니다.

```
kubectl patch tprov <trident-provisioner-name> -n <trident-namespace>
--type=merge -p '{"spec":{"debug":true}}'
```

5. Trident 연산자를 설치합니다.



클러스터 범위 연산자를 설치하면 마이그레이션이 시작됩니다 `TridentProvisioner` 개체 대상 `TridentOrchestrator` 오브젝트, 삭제 `TridentProvisioner` 개체 및 `tridentprovisioner` CRD를 사용하여 Astra Trident를 사용 중인 클러스터 범위 운영자 버전으로 업그레이드합니다. 다음 예제에서 Trident는 23.01.1로 업그레이드됩니다.



Trident 연산자를 사용하여 Astra Trident를 업그레이드하면 `tridentprovisioner` A로 `tridentorchestrator` 같은 이름의 개체입니다. 이 작업은 작업자가 자동으로 처리합니다. 업그레이드에는 이전과 동일한 네임스페이스에 Astra Trident도 설치됩니다.

```

#Ensure you are in the correct directory
pwd
/root/23.01.1/trident-installer

#Install the cluster-scoped operator in the **same namespace**
kubectl create -f deploy/<BUNDLE.YAML>
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#All tridentProvisioners will be removed, including the CRD itself
kubectl get tprov -n trident
Error from server (NotFound): Unable to list "trident.netapp.io/v1,
Resource=tridentprovisioners": the server could not find the requested
resource (get tridentprovisioners.trident.netapp.io)

#tridentProvisioners are replaced by tridentOrchestrator
kubectl get torc
NAME          AGE
trident       13s

#Examine Trident pods in the namespace
kubectl get pods -n trident
NAME                                                    READY   STATUS    RESTARTS
AGE
trident-controller-79df798bdc-m79dc                    6/6     Running   0
1m41s
trident-node-linux-xrst8                               2/2     Running   0
1m41s
trident-operator-5574dbbc68-nthjv                      1/1     Running   0
1m52s

#Confirm Trident has been updated to the desired version
kubectl describe torc trident | grep Message -A 3
Message:          Trident installed
Namespace:       trident
Status:          Installed
Version:         v23.01.1

```



를 클릭합니다 trident-controller 및 POD 이름은 23.01에 도입된 명명 규칙을 반영합니다.

제어 기반 작업자 설치를 업그레이드합니다

제어 기반 운영자 설치를 업그레이드하려면 다음 단계를 수행하십시오.



Astra Trident가 설치된 Kubernetes 클러스터를 1.24에서 1.25 이상으로 업그레이드할 경우 Values.YAML을 업데이트해야 합니다 `excludePodSecurityPolicy` 를 선택합니다 `true` 또는 을 추가합니다 `--set excludePodSecurityPolicy=true` 를 누릅니다 `helm upgrade` 명령을 먼저 실행한 후 클러스터를 업그레이드하십시오.

단계

1. 최신 Astra Trident 릴리스를 다운로드하십시오.
2. 를 사용합니다 `helm upgrade` 명령 위치 `trident-operator-23.01.1.tgz` 업그레이드하려는 버전을 반영합니다.

```
helm upgrade <name> trident-operator-23.01.1.tgz
```

초기 설치 중에 기본값이 아닌 옵션을 설정한 경우(예: Trident 및 CSI 이미지에 대한 전용, 미러 레지스트리 지정) 를 사용합니다 `--set` 이러한 옵션이 업그레이드 명령에 포함되도록 하려면 값이 기본값으로 재설정됩니다.



예를 들어, 의 기본값을 변경합니다 `tridentDebug`에서 다음 명령을 실행합니다.`

```
helm upgrade <name> trident-operator-23.01.1-custom.tgz --set  
tridentDebug=true
```

3. 실행 `helm list` 차트와 앱 버전이 모두 업그레이드되었는지 확인합니다. 실행 `tridentctl logs` 디버그 메시지를 검토합니다.

결과

Trident 운영자는 기존 Astra Trident 설치를 식별하고 운영자와 동일한 버전으로 업그레이드합니다.

비운영자 설치에서 업그레이드

에서 Trident 운영자의 최신 릴리즈로 업그레이드할 수 있습니다 `tridentctl` 설치:

단계

1. 최신 Astra Trident 릴리스를 다운로드하십시오.

```
# Download the release required [23.01.1]
mkdir 23.01.1
cd 23.01.1
wget
https://github.com/NetApp/trident/releases/download/v22.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

2. 를 생성합니다 tridentorchestrator 매니페스트에서 CRD를 선택합니다.

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. 연산자를 전개합니다.

```
#Install the cluster-scoped operator in the same namespace
kubectl create -f deploy/<BUNDLE.YAML>
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc 6/6     Running   0           150d
trident-node-linux-xrst8             2/2     Running   0           150d
trident-operator-5574dbbc68-nthjv    1/1     Running   0           1m30s
```

4. 을 생성합니다 TridentOrchestrator Astra Trident 설치용 CR.

```

#Create a tridentOrchestrator to initiate a Trident install
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0           1m
trident-csi-xrst8                    2/2     Running   0           1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0           5m41s

#Confirm Trident was upgraded to the desired version
kubectl describe torc trident | grep Message -A 3
Message:                             Trident installed
Namespace:                           trident
Status:                               Installed
Version:                              v23.01.1

```

결과

기존 백엔드 및 PVC는 자동으로 사용할 수 있습니다.

tridentctl로 업그레이드하십시오

을 사용하여 기존 Astra Trident 설치를 쉽게 업그레이드할 수 있습니다 `tridentctl`.

업그레이드 전 고려 사항

Astra Trident의 최신 릴리즈로 업그레이드할 때 다음 사항을 고려하십시오.

- Trident 20.01부터 베타 릴리즈만 제공됩니다 **"볼륨 스냅샷"** 가 지원됩니다. Kubernetes 관리자는 알파 스냅샷 개체를 베타로 안전하게 백업하거나 변환하여 레거시 알파 스냅샷을 유지하도록 주의해야 합니다.
- 볼륨 스냅샷의 베타 릴리스에는 수정된 CRD 세트와 스냅샷 컨트롤러가 도입되며, 이 두 가지 모두 Astra Trident를 설치하기 전에 설정해야 합니다. **"블로그입니다"** 알파 볼륨 스냅샷을 베타 형식으로 마이그레이션하는 단계에 대해 설명합니다.
- Astra Trident를 제거하고 다시 설치하면 업그레이드 역할을 합니다. Trident를 제거할 때 Astra Trident 배포에 사용되는 영구 볼륨 클레임(PVC) 및 영구 볼륨(PV)은 삭제되지 않습니다. 이미 프로비저닝된 PVS는 Astra Trident가 오프라인 상태인 동안 계속 사용할 수 있으며, Astra Trident는 다시 온라인 상태가 되면 중간 기간 동안 생성된 모든 PVC에 대해 볼륨을 프로비저닝합니다.



Astra Trident를 업그레이드할 때 업그레이드 프로세스를 중단하지 마십시오. 설치 프로그램이 완료될 때까지 실행되는지 확인합니다.

업그레이드 후 다음 단계

새로운 Trident 릴리즈(예: 주문형 볼륨 스냅샷)에서 제공되는 다양한 기능을 사용하려면 `rl` 사용하여 볼륨을 업그레이드할 수 있습니다 `tridentctl upgrade` 명령.

레거시 볼륨이 있는 경우 Astra Trident의 새로운 기능 세트를 모두 사용하려면 NFS/iSCSI 유형에서 CSI 유형으로 업그레이드해야 합니다. Trident에서 프로비저닝한 레거시 PV는 기존 기능 세트를 지원합니다.

볼륨을 CSI 유형으로 업그레이드하기로 결정할 때 다음 사항을 고려하십시오.

- 모든 볼륨을 업그레이드할 필요는 없습니다. 이전에 생성된 볼륨은 계속 액세스할 수 있으며 정상적으로 작동합니다.
- 업그레이드할 때 배포/StatefulSet 의 일부로 PV를 마운트할 수 있습니다. `deployment/StatefulSet` 을 아래로 가져올 필요는 없습니다.
- 업그레이드 시 독립 실행형 POD에 PV를 * 첨부할 수 없습니다. 볼륨을 업그레이드하기 전에 포드를 종료해야 합니다.
- PVC에 바인딩된 볼륨만 업그레이드할 수 있습니다. PVC에 바인딩되지 않은 용적은 업그레이드 전에 제거 및 가져와야 합니다.

볼륨 업그레이드의 예

다음은 볼륨 업그레이드를 수행하는 방법을 보여 주는 예입니다.

1. 실행 `kubectl get pv` PVS를 나열합니다.

```
kubectl get pv
NAME                                CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                                STORAGECLASS   REASON   AGE
default-pvc-1-a8475                    1073741824   RWO           Delete
Bound    default/pvc-1                        standard              19h
default-pvc-2-a8486                    1073741824   RWO           Delete
Bound    default/pvc-2                        standard              19h
default-pvc-3-a849e                    1073741824   RWO           Delete
Bound    default/pvc-3                        standard              19h
default-pvc-4-a84de                    1073741824   RWO           Delete
Bound    default/pvc-4                        standard              19h
trident                                2Gi         RWO           Retain
Bound    trident/trident                      standard              19h
```

현재 Trident 20.07에서 `rl` 사용하여 생성한 PVS는 4개입니다 `netapp.io/trident` 공급자.

2. 실행 `kubectl describe pv` PV에 대한 세부 정보를 봅니다.

```

kubect1 describe pv default-pvc-2-a8486

Name:                default-pvc-2-a8486
Labels:              <none>
Annotations:         pv.kubernetes.io/provisioned-by: netapp.io/trident
                    volume.beta.kubernetes.io/storage-class: standard
Finalizers:          [kubernetes.io/pv-protection]
StorageClass:        standard
Status:              Bound
Claim:               default/pvc-2
Reclaim Policy:      Delete
Access Modes:        RWO
VolumeMode:          Filesystem
Capacity:            1073741824
Node Affinity:       <none>
Message:
Source:
  Type:               NFS (an NFS mount that lasts the lifetime of a pod)
  Server:             10.xx.xx.xx
  Path:               /trid_1907_alpha_default_pvc_2_a8486
  ReadOnly:           false

```

PV는 을 사용하여 만들었습니다 netapp.io/trident 프로비저닝했으며 NFS 유형입니다. Astra Trident에서 제공하는 모든 새로운 기능을 지원하려면 이 PV를 CSI 유형으로 업그레이드해야 합니다.

3. 를 실행합니다 tridentctl upgrade volume <name-of-trident-volume> 레거시 Astra Trident 볼륨을 CSI 사양으로 업그레이드하는 명령입니다.


```

./tridentctl get volumes -n trident
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|          NAME          |  SIZE  | STORAGE CLASS | PROTOCOL |
BACKEND UUID           | STATE  | MANAGED      |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true         |          |
| default-pvc-3-a849e | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true         |          |
| default-pvc-1-a8475 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true         |          |
| default-pvc-4-a84de | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true         |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

./tridentctl upgrade volume default-pvc-2-a8486 -n trident
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|          NAME          |  SIZE  | STORAGE CLASS | PROTOCOL |
BACKEND UUID           | STATE  | MANAGED      |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true         |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

4. A를 실행합니다 kubectl describe pv 볼륨이 CSI 볼륨인지 확인합니다.

```

kubect1 describe pv default-pvc-2-a8486
Name:                default-pvc-2-a8486
Labels:              <none>
Annotations:         pv.kubernetes.io/provisioned-by: csi.trident.netapp.io
                    volume.beta.kubernetes.io/storage-class: standard
Finalizers:          [kubernetes.io/pv-protection]
StorageClass:        standard
Status:              Bound
Claim:               default/pvc-2
Reclaim Policy:      Delete
Access Modes:        RWO
VolumeMode:          Filesystem
Capacity:            1073741824
Node Affinity:       <none>
Message:
Source:
  Type:              CSI (a Container Storage Interface (CSI) volume
source)
  Driver:            csi.trident.netapp.io
  VolumeHandle:      default-pvc-2-a8486
  ReadOnly:          false
  VolumeAttributes:  backendUUID=c5a6f6a4-b052-423b-80d4-
8fb491a14a22

internalName=trid_1907_alpha_default_pvc_2_a8486
                    name=default-pvc-2-a8486
                    protocol=file
Events:              <none>

```

이러한 방식으로 Astra Trident에서 생성한 NFS/iSCSI 유형의 볼륨을 볼륨별로 CSI 유형으로 업그레이드할 수 있습니다.

Astra Trident를 제거합니다

Astra Trident의 설치 방식에 따라 여러 옵션을 사용하여 제거할 수 있습니다.

Helm을 사용하여 제거합니다

Helm을 사용하여 Astra Trident를 설치한 경우 를 사용하여 제거할 수 있습니다 `helm uninstall`.

```
#List the Helm release corresponding to the Astra Trident install.
helm ls -n trident
NAME                NAMESPACE          REVISION          UPDATED
STATUS              CHART               APP VERSION
trident             trident             1                2021-04-20
00:26:42.417764794 +0000 UTC deployed      trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

Trident 연산자를 사용하여 제거합니다

운영자를 사용하여 Astra Trident를 설치한 경우 다음 중 하나를 수행하여 제거할 수 있습니다.

- 편집 **TridentOrchestrator** 제거 플래그를 설정하려면: 편집할 수 있습니다 TridentOrchestrator 그리고 설정합니다 `spec.uninstall=true`. 를 편집합니다 TridentOrchestrator 를 사용하여 를 설정합니다 `uninstall` 아래에 표시된 대로 플래그 지정:

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"uninstall":true}}'
```

를 누릅니다 `uninstall` 플래그가 로 설정되어 있습니다 `true` Trident 운영자가 Trident를 제거하지만 Trident 자체 자체를 제거하지는 않습니다. Trident를 다시 설치하려면 해당 Trident를 정리하고 새 AgentOrchestrator를 생성해야 합니다.

- 삭제 **TridentOrchestrator**: 를 제거하여 TridentOrchestrator Astra Trident를 배포하는 데 사용된 CR은 작업자에게 Trident를 제거하도록 지시합니다. 작업자가 의 제거를 처리합니다 TridentOrchestrator 그런 다음 Astra Trident 구축과 디멘시작을 제거하고 설치의 일부로 생성한 Trident 포드를 삭제합니다. Astra Trident(CRD 생성 포함)를 완전히 제거하고 슬레이트 클린을 효과적으로 닦을 수 있습니다 TridentOrchestrator 을 전달합니다 `wipeout` 옵션을 선택합니다. 다음 예를 참조하십시오.

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

그러면 Astra Trident가 완전히 설치 제거되며, 백엔드 및 관리하는 볼륨과 관련된 모든 메타데이터가 지워집니다. 이후 설치하는 새로 설치하는 것으로 처리됩니다.



전체 제거를 수행할 때에만 CRD를 지우는 것을 고려해야 합니다. 이 작업은 취소할 수 없습니다. 처음부터 새로 **Astra Trident** 설치를 생성하기 위해 사용하지 않는 한 **CRD**를 지우지 마십시오.

을 사용하여 를 제거합니다 `tridentctl`

를 실행합니다 `uninstall` 명령을 입력합니다 `tridentctl` 다음과 같이 CRD 및 관련 객체를 제외한 Astra Trident와 관련된 모든 리소스를 제거하므로 설치 프로그램을 다시 쉽게 실행하여 최신 버전으로 업데이트할 수 있습니다.

```
./tridentctl uninstall -n <namespace>
```

Astra Trident를 완전히 제거하려면 Astra Trident에서 생성한 CRD의 종료자를 제거하고 CRD를 삭제해야 합니다.

Astra Trident를 다운그레이드하십시오

Astra Trident의 이전 버전으로 다운그레이드하는 데 필요한 단계에 대해 알아보십시오.

다운그레이드 시점

다음과 같은 다양한 이유로 다운그레이드를 고려할 수 있습니다.

- 비상 계획
- 업그레이드로 인해 발견된 버그를 즉시 수정합니다
- 종속성 문제, 실패 및 불완전한 업그레이드

CRD를 사용하는 Astra Trident 릴리즈로 이전할 때는 다운그레이드를 고려해야 합니다. Astra Trident는 CRD를 사용하여 상태를 유지하기 때문에 생성된 모든 스토리지 요소(백엔드, 스토리지 클래스, PV 및 볼륨 스냅샷)는 에 기록된 데이터 대신 CRD 객체를 연결합니다 `trident PV`(이전 설치 버전의 Astra Trident에서 사용). 새로 생성된 PVS, 백엔드 및 스토리지 클래스는 모두 CRD 객체로 유지됩니다.

CRD를 사용하여 실행되는 Astra Trident 버전에 대해서만 다운그레이드를 시도하십시오(19.07 이상). 그러면 다운그레이드가 발생한 후 현재 Astra Trident 릴리스에 대해 수행된 작업이 표시됩니다.

다운그레이드를 하지 않는 경우

를 사용하는 Trident 릴리즈로 다운그레이드하면 안 됩니다 `etcd` 상태를 유지합니다(19.04 이하). 현재 Astra Trident 릴리즈를 통해 수행된 모든 작업은 다운그레이드 후 반영되지 않습니다. 새로 생성된 PVS는 이전 버전으로 되돌릴 때 사용할 수 없습니다. 이전 버전으로 돌아갈 때 Astra Trident에서 백엔드, PVS, 스토리지 클래스 및 볼륨 스냅샷(생성/업데이트/삭제)과 같은 객체에 대한 변경 사항을 볼 수 없습니다. 이전 버전으로 돌아가도 업그레이드되지 않은 경우 이전 릴리즈를 사용하여 이미 생성된 PVS에 대한 액세스가 중단되지 않습니다.

운영자를 통해 Astra Trident가 설치된 경우의 다운그레이드 프로세스

Trident 연산자를 사용하여 설치한 경우 다운그레이드 프로세스가 다르며 을 사용할 필요가 없습니다 `tridentctl`.

Trident 연산자를 사용하여 설치한 경우 Astra Trident를 다음 중 하나로 다운그레이드할 수 있습니다.

- 네임스페이스 범위 연산자를 사용하여 설치된 버전(20.07-20.10).
- 클러스터 범위 연산자(21.01 이상)를 사용하여 설치된 버전입니다.

클러스터 범위 연산자로 다운그레이드

Astra Trident를 클러스터 범위 운영자를 사용하는 릴리즈로 다운그레이드하려면 아래에 설명된 단계를 따르십시오.

단계

1. "**Astra Trident를 제거합니다**". 기존 설치를 완전히 제거하지 않는 한 **CRD**를 삭제하지 마십시오.
2. Trident 연산자는 사용 중인 Trident 버전에 연결된 연산자 매니페스트를 사용하여 삭제할 수 있습니다. 예를 들면, 다음과 같습니다. <https://github.com/NetApp/trident/tree/stable/vXX.XX> /*deploy/bundle.yaml* 위치 *vXX.XX* 은 버전 번호입니다(예 *v22.10*) 및 *bundle.yaml* 번들 YAML 파일 이름입니다.
3. 원하는 버전의 Astra Trident를 설치하여 다운그레이드를 계속합니다. 원하는 릴리스에 대한 설명서를 따릅니다.

네임스페이스 범위 연산자로 다운그레이드합니다

이 섹션에서는 네임스페이스 범위 연산자를 사용하여 설치되는 20.07 ~ 20.10 범위의 Astra Trident 릴리스로 다운그레이드하는 단계를 요약합니다.

단계

1. "**Astra Trident를 제거합니다**". 기존 설치를 완전히 제거하지 않는 한 **CRD**를 휘두리지 마십시오. 을 확인하십시오 `tridentorchestrator` 이(가) 삭제됩니다.

```
#Check to see if there are any tridentorchestrators present
kubectl get torc
NAME          AGE
trident      20h

#Looks like there is a tridentorchestrator that needs deleting
kubectl delete torc trident
tridentorchestrator.trident.netapp.io "trident" deleted
```

2. Trident 연산자는 사용 중인 Trident 버전에 연결된 연산자 매니페스트를 사용하여 삭제할 수 있습니다. 예를 들면, 다음과 같습니다. <https://github.com/NetApp/trident/tree/stable/vXX.XX> /*deploy/bundle.yaml* 위치 *vXX.XX* 은 버전 번호입니다(예 *v22.10*) 및 *bundle.yaml* 번들 YAML 파일 이름입니다.
3. 를 삭제합니다 `tridentorchestrator` CRD

```
#Check to see if ``tridentorchestrators.trident.netapp.io`` CRD is present and delete it.
```

```
kubectl get crd tridentorchestrators.trident.netapp.io
```

```
NAME                                CREATED AT
tridentorchestrators.trident.netapp.io 2021-01-21T21:11:37Z
```

```
kubectl delete crd tridentorchestrators.trident.netapp.io
```

```
customresourcedefinition.apiextensions.k8s.io
"tridentorchestrators.trident.netapp.io" deleted
```

Astra Trident가 제거되었습니다.

- 원하는 버전을 설치하여 다운그레이드를 계속합니다. 원하는 릴리스에 대한 설명서를 따릅니다.

H제어 를 사용하여 다운그레이드합니다

다운그레이드하려면 을 사용합니다 helm rollback 명령. 다음 예를 참조하십시오.

```
helm rollback trident [revision #]
```

을 사용하여 **Astra Trident**를 설치한 경우의 다운그레이드 프로세스 tridentctl

을 사용하여 Astra Trident를 설치한 경우 `tridentctl`다운그레이드 프로세스는 다음 단계를 포함합니다. 이 시퀀스는 Astra Trident 21.07에서 20.07로 이동하는 다운그레이드 프로세스를 안내합니다.



다운그레이드를 시작하기 전에 Kubernetes 클러스터의 스냅샷을 만들어야 합니다 etcd. 이를 통해 Astra Trident의 CRD의 현재 상태를 백업할 수 있습니다.

단계

- 를 사용하여 Trident가 설치되었는지 확인합니다 tridentctl. Astra Trident의 설치 방법을 잘 모르는 경우 다음 간단한 테스트를 실행하십시오.
 - Trident 네임스페이스에 있는 포드를 나열합니다.
 - 클러스터에서 실행 중인 Astra Trident의 버전을 확인합니다. 를 사용할 수 있습니다 tridentctl 또는 Trident Pod에 사용된 이미지를 살펴보십시오.
 - 가 표시되지 않는 경우 * a tridentOrchestrator, (또는) a tridentprovisioner, (또는) 이름이 인 포드 trident-operator-xxxxxxxx-xxxxx, Astra Trident * 가 와 함께 설치됩니다 tridentctl.
- 기존 인프라와 함께 Astra Trident를 제거합니다 tridentctl 바이너리. 이 경우 21.07 바이너리로 를 제거합니다.

```

tridentctl version -n trident
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 21.07.0       | 21.07.0       |
+-----+-----+

tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted Trident daemonset.
INFO Deleted Trident service.
INFO Deleted Trident secret.
INFO Deleted Trident cluster role binding.
INFO Deleted Trident cluster role.
INFO Deleted Trident service account.
INFO Deleted Trident pod security policy.
podSecurityPolicy=tridentpods
INFO The uninstaller did not delete Trident's namespace in case it is
going to be reused.
INFO Trident uninstallation succeeded.

```

3. 이 작업이 완료되면 원하는 버전의 Trident 바이너리(이 예: 20.07)를 얻고 이를 사용하여 Astra Trident를 설치합니다. 예 대한 사용자 지정 YAML을 생성할 수 있습니다 "[맞춤형 설치](#)" 필요한 경우

```

cd 20.07/trident-installer/
./tridentctl install -n trident-ns
INFO Created installer service account.
serviceaccount=trident-installer
INFO Created installer cluster role.                clusterrole=trident-
installer
INFO Created installer cluster role binding.
clusterrolebinding=trident-installer
INFO Created installer configmap.                  configmap=trident-
installer
...
...
INFO Deleted installer cluster role binding.
INFO Deleted installer cluster role.
INFO Deleted installer service account.

```

다운그레이드 프로세스가 완료되었습니다.

Astra Trident를 사용해 보십시오

작업자 노드를 준비합니다

Kubernetes 클러스터의 모든 작업자 노드는 Pod용으로 프로비저닝된 볼륨을 마운트할 수 있어야 합니다. 작업자 노드를 준비하려면 드라이버 선택에 따라 NFS 또는 iSCSI 도구를 설치해야 합니다.

올바른 도구 선택

드라이버를 함께 사용하는 경우 NFS 및 iSCSI 도구를 설치해야 합니다.

NFS 툴

다음은 사용 중인 경우 NFS 툴을 설치합니다. `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `azure-netapp-files`, `gcp-cvs`

iSCSI 툴

다음은 사용하는 경우 iSCSI 도구를 설치합니다. `ontap-san`, `ontap-san-economy`, `solidfire-san`



최신 버전의 RedHat CoreOS에는 기본적으로 NFS 및 iSCSI가 설치되어 있습니다.

노드 서비스 검색

Astra Trident는 노드가 iSCSI 또는 NFS 서비스를 실행할 수 있는지 자동으로 감지하려고 시도합니다.



노드 서비스 검색은 검색된 서비스를 식별하지만 서비스가 올바르게 구성되었다고 보장하지 않습니다. 반대로 검색된 서비스가 없으면 볼륨 마운트가 실패한다고 보장할 수 없습니다.

이벤트를 검토합니다

Astra Trident는 검색된 서비스를 식별하기 위해 노드에 대한 이벤트를 생성합니다. 이러한 이벤트를 검토하려면 다음을 실행합니다.

```
kubect1 get event -A --field-selector involvedObject.name=<Kubernetes node name>
```

검색된 서비스를 검토합니다

Astra Trident는 Trident 노드 CR의 각 노드에 대해 활성화된 서비스를 식별합니다. 검색된 서비스를 보려면 다음을 실행합니다.

```
tridentctl get node -o wide -n <Trident namespace>
```


NFS 볼륨

운영 체제의 명령을 사용하여 NFS 툴을 설치합니다. 부팅 중에 NFS 서비스가 시작되었는지 확인합니다.

RHEL 8+

```
sudo yum install -y nfs-utils
```

우분투

```
sudo apt-get install -y nfs-common
```



볼륨에 연결할 때 오류가 발생하지 않도록 NFS 툴을 설치한 후 작업자 노드를 재부팅합니다.

iSCSI 볼륨

Astra Trident는 iSCSI 세션을 자동으로 설정하고, LUN을 검색하고, 다중 경로 장치를 검색하고, 포맷하고, 포드에 마운트할 수 있습니다.

iSCSI 자동 복구 기능

ONTAP 시스템의 경우, Astra Trident가 5분마다 iSCSI 자동 복구를 실행하여 다음을 수행합니다.

1. * 원하는 iSCSI 세션 상태와 현재 iSCSI 세션 상태를 식별합니다.
2. * 원하는 상태를 현재 상태와 비교 * 하여 필요한 수리를 확인합니다. Astra Trident는 수리 우선 순위 및 수리 시기를 결정합니다.
3. * 현재 iSCSI 세션 상태를 원하는 iSCSI 세션 상태로 되돌리는 데 필요한 복구 수행 *



자동 복구 작업의 로그는 `trident-main` 각 드로포드에 있는 용기. 로그를 보려면 `을(를)` 설정해야 합니다 `debug Astra Trident`를 설치하는 동안 "참"으로 표시합니다.

Astra Trident iSCSI 자동 복구 기능을 사용하면 다음과 같은 문제를 방지할 수 있습니다.

- 네트워크 연결 문제가 발생한 후 발생할 수 있는 오래되거나 비정상적인 iSCSI 세션. 오래된 세션의 경우 Astra Trident는 로그아웃하기 7분 전에 대기하여 포털과의 연결을 다시 설정합니다.



예를 들어, 스토리지 컨트롤러에서 CHAP 암호를 회전시키고 네트워크에서 연결이 끊어지면 이전의 (*stale*) CHAP 암호가 지속될 수 있습니다. 자동 복구 기능은 이 문제를 인식하고 업데이트된 CHAP 암호를 적용하기 위해 세션을 자동으로 다시 설정할 수 있습니다.

- iSCSI 세션이 누락되었습니다
- LUN이 없습니다

iSCSI 도구를 설치합니다

운영 체제의 명령을 사용하여 iSCSI 도구를 설치합니다.

시작하기 전에

- Kubernetes 클러스터의 각 노드에는 고유한 IQN이 있어야 합니다. * 이것은 필수 전제 조건입니다 *.
- RHCOS 버전 4.5 이상 또는 기타 RHEL 호환 Linux 배포를 사용하는 경우 를 참조하십시오 `solidfire-san` 드라이버 및 Element OS 12.5 이전 버전에서는 CHAP 인증 알고리즘이 에서 MD5로 설정되어 있는지 확인합니다 `/etc/iscsi/iscsid.conf`. 보안 FIPS 호환 CHAP 알고리즘 SHA1, SHA-256 및 SHA3-256은 Element 12.7에서 사용할 수 있습니다.

```
sudo sed -i 's/^\(node.session.auth.chap_algs\) .*/\1 = MD5/'  
/etc/iscsi/iscsid.conf
```

- iSCSI PVS에서 RHEL/RedHat CoreOS를 실행하는 작업자 노드를 사용하는 경우 를 지정합니다 `discard` StorageClass의 `mountOption`을 사용하여 인라인 공간 재확보를 수행합니다. 을 참조하십시오 "[RedHat 설명서](#)".

RHEL 8+

1. 다음 시스템 패키지를 설치합니다.

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-  
mapper-multipath
```

2. iscsi-initiator-utils 버전이 6.2.0.874-2.el7 이상인지 확인합니다.

```
rpm -q iscsi-initiator-utils
```

3. 스캔을 수동으로 설정합니다.

```
sudo sed -i 's/^\(node.session.scan\) .*/\1 = manual/'  
/etc/iscsi/iscsid.conf
```

4. 다중 경로 설정:

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```



확인합니다 `etc/multipath.conf` 포함 `find_multipaths no` 에서 defaults.

5. 확인하십시오 iscsid 및 multipathd 실행 중:

```
sudo systemctl enable --now iscsid multipathd
```

6. 활성화 및 시작 iscsi:

```
sudo systemctl enable --now iscsi
```

우분투

1. 다음 시스템 패키지를 설치합니다.

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools  
scsitools
```

2. open-iscsi 버전이 2.0.874-5ubuntu2.10 이상(bionic) 또는 2.0.874-7.1ubuntu6.1 이상(focal)인지 확인합니다.

```
dpkg -l open-iscsi
```

3. 스캔을 수동으로 설정합니다.

```
sudo sed -i 's/^\(node.session.scan\).*\/\1 = manual/'  
/etc/iscsi/iscsid.conf
```

4. 다중 경로 설정:

```
sudo tee /etc/multipath.conf <<-'EOF'  
defaults {  
    user_friendly_names yes  
    find_multipaths no  
}  
EOF  
sudo systemctl enable --now multipath-tools.service  
sudo service multipath-tools restart
```



확인합니다 `etc/multipath.conf` 포함 `find_multipaths no` 에서 `defaults`.

5. 확인하십시오 `open-iscsi` 및 `multipath-tools` 활성화 및 실행:

```
sudo systemctl status multipath-tools  
sudo systemctl enable --now open-iscsi.service  
sudo systemctl status open-iscsi
```



Ubuntu 18.04의 경우 을 사용하여 대상 포트를 검색해야 합니다 `iscsiadm` 시작 전 `open-iscsi` iSCSI 데몬을 시작합니다. 또는 을 수정할 수 있습니다 `iscsi` 시작할 서비스 `iscsid` 자동으로.



컨테이너에 볼륨을 연결할 때 오류가 발생하지 않도록 iSCSI 도구를 설치한 후 작업자 노드를 재부팅합니다.

백엔드 구성

백엔드는 Astra Trident와 스토리지 시스템 간의 관계를 정의합니다. Astra Trident가 스토리지 시스템과 통신하는 방법과 Astra Trident가 스토리지 시스템에서 볼륨을 프로비저닝하는 방법을 알려줍니다.

Astra Trident는 스토리지 클래스에 정의된 요구 사항과 일치하는 백엔드에서 스토리지 풀을 자동으로 제공합니다.

스토리지 시스템에 대한 백엔드를 구성하는 방법에 대해 알아봅니다.

- ["Azure NetApp Files 백엔드를 구성합니다"](#)
- ["Google Cloud Platform 백엔드에 Cloud Volumes Service를 구성합니다"](#)
- ["NetApp HCI 또는 SolidFire 백엔드를 구성합니다"](#)
- ["ONTAP 또는 Cloud Volumes ONTAP NAS 드라이버를 사용하여 백엔드를 구성합니다"](#)
- ["ONTAP 또는 Cloud Volumes ONTAP SAN 드라이버를 사용하여 백엔드를 구성합니다"](#)
- ["NetApp ONTAP용 Amazon FSx와 함께 Astra Trident를 사용하십시오"](#)

Azure NetApp Files

Azure NetApp Files 백엔드를 구성합니다

Astra Trident의 백엔드로 Azure NetApp Files(ANF)를 구성할 수 있습니다. ANF 백엔드를 사용하여 NFS 및 SMB 볼륨을 연결할 수 있습니다.

- ["준비"](#)
- ["구성 옵션 및 예"](#)

고려 사항

- Azure NetApp Files 서비스는 100GB 미만의 볼륨을 지원하지 않습니다. Astra Trident는 더 작은 볼륨을 요청하는 경우 100GB 볼륨을 자동으로 생성합니다.
- Astra Trident는 Windows 노드에서 실행되는 Pod에만 마운트된 SMB 볼륨을 지원합니다.
- Astra Trident는 Windows ARM 아키텍처를 지원하지 않습니다.

Azure NetApp Files 백엔드를 구성할 준비를 합니다

Azure NetApp Files 백엔드를 구성하기 전에 다음 요구 사항이 충족되는지 확인해야 합니다.



Azure NetApp Files를 처음 사용하거나 새 위치에서 사용하는 경우 Azure NetApp Files를 설정하고 NFS 볼륨을 생성하려면 몇 가지 초기 구성이 필요합니다. 을 참조하십시오 ["Azure: Azure NetApp Files를 설정하고 NFS 볼륨을 생성합니다"](#).

NFS 및 SMB 볼륨의 사전 요구 사항

를 구성하고 사용합니다 ["Azure NetApp Files"](#) 백엔드, 다음이 필요합니다.

- 용량 풀입니다. 을 참조하십시오 ["Microsoft: Azure NetApp Files에 대한 용량 풀을 생성합니다"](#).
- Azure NetApp Files에 위임된 서브넷. 을 참조하십시오 ["Microsoft: Azure NetApp Files에 서브넷을 위임합니다"](#).
- subscriptionID Azure NetApp Files가 활성화된 Azure 구독에서
- tenantID, clientID, 및 clientSecret 에서 ["앱 등록"](#) Azure NetApp Files 서비스에 대한 충분한 권한이 있는 Azure Active Directory에서 앱 등록에서는 다음 중 하나를 사용해야 합니다.
 - 소유자 또는 참가자 역할입니다 ["Azure에서 사전 정의"](#).

- A "사용자 지정 참가자 역할" 구독 레벨입니다 (assignableScopes) 다음 사용 권한은 Astra Trident에 필요한 권한만 가집니다. 사용자 지정 역할을 만든 후 "Azure 포털을 사용하여 역할을 할당합니다".

```
{
  "id": "/subscriptions/<subscription-id>/providers/Microsoft.Authorization/roleDefinitions/<role-definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/delete",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/delete",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/write",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/delete",
          "Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/Get"
        ]
      }
    ]
  }
}
```

```

Metadata/action",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTargets/read",
    "Microsoft.Network/virtualNetworks/read",
    "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations/delete",
    "Microsoft.Features/features/read",
    "Microsoft.Features/operations/read",
    "Microsoft.Features/providers/features/read",

"Microsoft.Features/providers/features/register/action",

"Microsoft.Features/providers/features/unregister/action",

"Microsoft.Features/subscriptionFeatureRegistrations/read"
    ],
    "notActions": [],
    "dataActions": [],
    "notDataActions": []
}
]
}
}

```

- Azure를 선택합니다 location 하나 이상의 항목이 포함되어 있습니다 "위임된 서브넷". Trident 22.01부터 location 매개 변수는 백엔드 구성 파일의 최상위 수준에 있는 필수 필드입니다. 가상 풀에 지정된 위치 값은 무시됩니다.

SMB 볼륨에 대한 추가 요구사항

SMB 볼륨을 생성하려면 다음이 있어야 합니다.

- Active Directory가 구성되어 Azure NetApp Files에 연결되었습니다. 을 참조하십시오 "[Microsoft: Azure NetApp Files에 대한 Active Directory 연결을 만들고 관리합니다](#)".
- Linux 컨트롤러 노드 및 Windows Server 2019를 실행하는 Windows 작업자 노드가 있는 Kubernetes 클러스터 Astra Trident는 Windows 노드에서 실행되는 Pod에만 마운트된 SMB 볼륨을 지원합니다.
- Azure NetApp Files가 Active Directory에 인증할 수 있도록 Active Directory 자격 증명에 포함된 Astra Trident

암호가 하나 이상 있어야 합니다. 암호를 생성합니다 smbcreds:

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- Windows 서비스로 구성된 CSI 프록시. 를 구성합니다 `csi-proxy`를 참조하십시오 ["GitHub:CSI 프록시"](#) 또는 ["GitHub: Windows용 CSI 프록시"](#) Windows에서 실행되는 Kubernetes 노드의 경우:

Azure NetApp Files 백엔드 구성 옵션 및 예

ANF에 대한 NFS 및 SMB 백엔드 구성 옵션에 대해 알아보고 구성 예를 검토하십시오.

Astra Trident는 백엔드 구성(서브넷, 가상 네트워크, 서비스 수준 및 위치)을 사용하여 요청된 위치에서 사용할 수 있고 요청된 서비스 수준 및 서브넷과 일치하는 용량 풀에 ANF 볼륨을 생성합니다.



Astra Trident는 수동 QoS 용량 풀을 지원하지 않습니다.

백엔드 구성 옵션

ANF 백엔드는 이러한 구성 옵션을 제공합니다.

매개 변수	설명	기본값
version		항상 1
storageDriverName	스토리지 드라이버의 이름입니다	"Azure-NetApp-파일"
backendName	사용자 지정 이름 또는 스토리지 백엔드	드라이버 이름 + "_" + 임의 문자
subscriptionID	Azure 구독의 구독 ID입니다	
tenantID	앱 등록에서 테넌트 ID입니다	
clientID	앱 등록의 클라이언트 ID입니다	
clientSecret	앱 등록에서 클라이언트 암호	
serviceLevel	중 하나 Standard, Premium, 또는 Ultra	""(임의)
location	새 볼륨을 생성할 Azure 위치의 이름입니다	
resourceGroups	검색된 자원을 필터링하기 위한 자원 그룹 목록입니다	[]"(필터 없음)
netappAccounts	검색된 리소스를 필터링하기 위한 NetApp 계정의 목록입니다	[]"(필터 없음)
capacityPools	검색된 리소스를 필터링하기 위한 용량 풀 목록입니다	[]"(필터 없음, 임의)
virtualNetwork	위임된 서브넷이 있는 가상 네트워크의 이름입니다	""

매개 변수	설명	기본값
subnet	위임된 서브넷의 이름입니다 Microsoft.Netapp/volumes	""
networkFeatures	볼륨에 대한 VNET 기능 집합은 일 수 있습니다 Basic 또는 Standard. 일부 지역에서는 네트워크 기능을 사용할 수 없으며 구독에서 활성화해야 할 수도 있습니다. 지정 networkFeatures 이 기능을 사용하지 않으면 볼륨 프로비저닝이 실패합니다.	""
nfsMountOptions	NFS 마운트 옵션에 대한 세밀한 제어 SMB 볼륨에 대해 무시됩니다. NFS 버전 4.1을 사용하여 볼륨을 마운트하려면 을 포함합니다 nfsvers=4 심표로 구분된 마운트 옵션 목록에서 NFS v4.1을 선택합니다. 스토리지 클래스 정의에 설정된 마운트 옵션은 백엔드 구성에 설정된 마운트 옵션을 재정의합니다.	"nfsvers=3"
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다	""(기본적으로 적용되지 않음)
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: \{"api": false, "method": true, "discovery": true\}. 문제 해결 중이 아니며 자세한 로그 덤프가 필요한 경우가 아니면 이 방법을 사용하지 마십시오.	null입니다
nasType	NFS 또는 SMB 볼륨 생성을 구성합니다. 옵션은 입니다 nfs, smb 또는 null입니다. Null로 설정하면 기본적으로 NFS 볼륨이 설정됩니다.	nfs



네트워크 기능에 대한 자세한 내용은 을 참조하십시오 ["Azure NetApp Files 볼륨에 대한 네트워크 기능을 구성합니다"](#).

필요한 권한 및 리소스

PVC를 생성할 때 "용량 풀을 찾을 수 없음" 오류가 발생하는 경우 앱 등록에 필요한 사용 권한 및 리소스(서브넷, 가상 네트워크, 용량 풀)가 연결되지 않은 것일 수 있습니다. DEBUG가 활성화된 경우 Astra Trident는 백엔드가 생성될 때 검색된 Azure 리소스를 기록합니다. 적절한 역할이 사용되고 있는지 확인합니다.

의 값 resourceGroups, netappAccounts, capacityPools, virtualNetwork, 및 subnet 간단한 이름 또는 정규화된 이름을 사용하여 지정할 수 있습니다. 이름이 같은 여러 리소스와 이름이 일치할 수 있으므로 대부분의 경우 정규화된 이름을 사용하는 것이 좋습니다.

를 클릭합니다 resourceGroups, netappAccounts, 및 capacityPools 값은 검색된 리소스 집합을 이 스토리지 백엔드에서 사용할 수 있는 리소스로 제한하는 필터이며, 이 둘을 조합하여 지정할 수 있습니다. 정규화된

이름은 다음 형식을 따릅니다.

유형	형식
리소스 그룹	리소스 그룹>
NetApp 계정	리소스 그룹>/<NetApp 계정>
용량 풀	리소스 그룹>/<NetApp 계정>/<용량 풀>
가상 네트워크	리소스 그룹>/<가상 네트워크>
서브넷	리소스 그룹>/<가상 네트워크>/<서브넷>

볼륨 프로비저닝

구성 파일의 특수 섹션에서 다음 옵션을 지정하여 기본 볼륨 프로비저닝을 제어할 수 있습니다. 을 참조하십시오 [예제 설정](#) 를 참조하십시오.

매개 변수	설명	기본값
exportRule	새 볼륨에 대한 익스포트 규칙 exportRule CIDR 표기법을 사용하여 IPv4 주소 또는 IPv4 서브넷의 조합을 쉼표로 구분해야 합니다. SMB 볼륨에 대해 무시됩니다.	"0.0.0.0/0"
snapshotDir	스냅샷 디렉터리의 표시 여부를 제어합니다	"거짓"
size	새 볼륨의 기본 크기입니다	"100G"
unixPermissions	새 볼륨의 UNIX 사용 권한(8진수 4자리) SMB 볼륨에 대해 무시됩니다.	""(미리보기 기능, 가입 시 화이트리스트 필요)

예제 설정

예 1: 최소 구성

이는 절대적인 최소 백엔드 구성입니다. 이 구성을 통해 Astra Trident는 구성된 위치에서 ANF에 위임된 모든 NetApp 계정, 용량 풀 및 서버넷을 검색하고 해당 풀 및 서버넷 중 하나에 무작위로 새 볼륨을 배치합니다. 왜냐하면 `nasType` 생략됩니다 `nfs` 기본값은 NFS 볼륨에 대해 백엔드가 프로비저닝됩니다.

이 구성은 ANF를 사용하여 지금 막 시작하는 데 이상적이지만, 실제로 용량 할당을 수행하는 볼륨에 대한 추가적인 범위 지정을 제공하려는 경우에 적합합니다.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
```

예 2: 용량 풀 필터를 사용하는 특정 서비스 수준 구성

이 백엔드 구성은 Azure에 볼륨을 배치합니다 `eastus` 의 위치 `Ultra` 용량 풀. Astra Trident는 해당 위치의 ANF에 위임된 모든 서버넷을 자동으로 검색하여 그 중 하나에 무작위로 새 볼륨을 배치합니다.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
- application-group-1/account-1/ultra-1
- application-group-1/account-1/ultra-2
```

예 3: 고급 구성

이 백엔드 구성은 단일 서브넷에 대한 볼륨 배치 범위를 더욱 줄여주고 일부 볼륨 프로비저닝 기본값도 수정합니다.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
- application-group-1/account-1/ultra-1
- application-group-1/account-1/ultra-2
virtualNetwork: my-virtual-network
subnet: my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: 'true'
  size: 200Gi
  unixPermissions: '0777'
```

예 4: 가상 풀 구성

이 백엔드 구성은 단일 파일에 여러 스토리지 풀을 정의합니다. 다양한 서비스 수준을 지원하는 여러 용량 풀이 있고 이를 나타내는 Kubernetes의 스토리지 클래스를 생성하려는 경우에 유용합니다. 가상 풀 레이블을 사용하여 에 따라 풀을 구분했습니다 performance.

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
resourceGroups:
- application-group-1
networkFeatures: Basic
nfsMountOptions: vers=3,proto=tcp,timeo=600
labels:
  cloud: azure
storage:
- labels:
  performance: gold
  serviceLevel: Ultra
  capacityPools:
  - ultra-1
  - ultra-2
  networkFeatures: Standard
- labels:
  performance: silver
  serviceLevel: Premium
  capacityPools:
  - premium-1
- labels:
  performance: bronze
  serviceLevel: Standard
  capacityPools:
  - standard-1
  - standard-2
```

스토리지 클래스 정의

다음 사항을 참조하십시오 StorageClass 정의는 위의 스토리지 풀을 참조합니다.

을 사용한 정의 예 `parameter.selector` 필드에 입력합니다

사용 `parameter.selector` 각각에 대해 지정할 수 있습니다 `StorageClass` 볼륨을 호스팅하는 데 사용되는 가상 풀입니다. 볼륨은 선택한 풀에 정의된 측면을 갖습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=gold"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: silver
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: bronze
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=bronze"
allowVolumeExpansion: true
```

SMB 볼륨에 대한 정의의 예

사용 `nasType`, `node-stage-secret-name`, 및 `node-stage-secret-namespace`, **SMB** 볼륨을 지정하고 필요한 **Active Directory** 자격 증명을 제공할 수 있습니다.

예제 1: 기본 네임스페이스의 기본 구성

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

예제 2: 네임스페이스당 다른 암호 사용

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

예 3: 볼륨별로 다른 암호 사용

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```



nasType: `smb` SMB 볼륨을 지원하는 풀에 대한 필터입니다. nasType: `nfs` 또는 nasType: `null` NFS 풀에 대한 필터입니다.

백엔드를 생성합니다

백엔드 구성 파일을 생성한 후 다음 명령을 실행합니다.

```
tridentctl create backend -f <backend-file>
```

백엔드 생성에 실패하면 백엔드 구성에 문제가 있는 것입니다. 다음 명령을 실행하여 로그를 보고 원인을 확인할 수 있습니다.

```
tridentctl logs
```

구성 파일의 문제를 확인하고 수정한 후 create 명령을 다시 실행할 수 있습니다.

Google Cloud용 Cloud Volumes Service 백엔드를 구성합니다

제공된 샘플 구성을 사용하여 Astra Trident 설치를 위한 백엔드로 NetApp Cloud Volumes Service for Google Cloud를 구성하는 방법을 알아보십시오.

Cloud Volumes Service for Google Cloud를 위한 **Astra Trident** 지원에 대해 알아보십시오

Astra Trident는 두 개 중 하나로 Cloud Volumes Service 볼륨을 생성할 수 있습니다 "[서비스 유형](#)":

- * CVS - 성능 *: 기본 Astra Trident 서비스 유형입니다. 이처럼 성능에 최적화된 서비스 유형은 성능을 중요시하는 운영 워크로드에 가장 적합합니다. CVS - 성능 서비스 유형은 최소 100GiB 크기의 볼륨을 지원하는 하드웨어 옵션입니다. 다음 중 하나를 선택할 수 있습니다 "[3가지 서비스 레벨](#)":
 - standard
 - premium
 - extreme
- CVS *: CVS 서비스 유형은 높은 조널 가용성을 제공하며, 성능은 중간 수준으로 제한됩니다. CVS 서비스 유형은 스토리지 풀을 사용하여 1GiB의 작은 볼륨을 지원하는 소프트웨어 옵션입니다. 스토리지 풀에는 최대 50개의 볼륨이 포함될 수 있으며 이 볼륨에서 풀의 용량과 성능을 공유할 수 있습니다. 다음 중 하나를 선택할 수 있습니다 "[서비스 레벨 2개](#)":
 - standardsw
 - zoneredundantstandardsw

필요한 것

를 구성하고 사용합니다 "[Google Cloud용 Cloud Volumes Service](#)" 백엔드, 다음이 필요합니다.

- NetApp Cloud Volumes Service로 구성된 Google Cloud 계정
- Google Cloud 계정의 프로젝트 번호입니다

- 에 Google Cloud 서비스 계정이 있습니다 `netappcloudvolumes.admin` 역할
- Cloud Volumes Service 계정에 대한 API 키 파일입니다

백엔드 구성 옵션

각 백엔드는 단일 Google Cloud 지역에 볼륨을 프로비저닝합니다. 다른 영역에 볼륨을 생성하려면 추가 백엔드를 정의할 수 있습니다.

매개 변수	설명	기본값
<code>version</code>		항상 1
<code>storageDriverName</code>	스토리지 드라이버의 이름입니다	"GCP-CV"
<code>backendName</code>	사용자 지정 이름 또는 스토리지 백엔드	드라이버 이름 + "_" + API 키의 일부
<code>storageClass</code>	CVS 서비스 유형을 지정하는 데 사용되는 선택적 매개 변수입니다. 사용 <code>software</code> CVS 서비스 유형을 선택합니다. 그렇지 않으면 Astra Trident가 CVS - 성능 서비스 유형을 가정합니다 (<code>hardware</code>)를 클릭합니다.	
<code>storagePools</code>	CVS 서비스 유형만 볼륨 생성을 위한 스토리지 풀을 지정하는 데 사용되는 선택적 매개 변수입니다.	
<code>projectNumber</code>	Google Cloud 계정 프로젝트 번호입니다. 이 값은 Google Cloud 포털 홈 페이지에서 확인할 수 있습니다.	
<code>hostProjectNumber</code>	공유 VPC 네트워크를 사용하는 경우 필요합니다. 이 시나리오에서는 <code>projectNumber</code> 은(는) 서비스 프로젝트입니다 <code>hostProjectNumber</code> 는 호스트 프로젝트입니다.	
<code>apiRegion</code>	Astra Trident가 Cloud Volumes Service 볼륨을 생성하는 Google 클라우드 영역 지역 간 Kubernetes 클러스터를 생성할 때 에서 생성된 볼륨입니다 <code>apiRegion</code> 여러 Google Cloud 지역의 노드에 예약된 워크로드에 사용할 수 있습니다. 지역 간 트래픽에는 추가 비용이 발생합니다.	

매개 변수	설명	기본값
apiKey	를 사용하여 Google Cloud 서비스 계정에 대한 API 키입니다 netappcloudvolumes.admin 역할. 여기에는 Google Cloud 서비스 계정의 개인 키 파일(백엔드 구성 파일에 verbatim 복사)의 JSON 형식 콘텐츠가 포함됩니다.	
proxyURL	프록시 서버가 CVS 계정에 연결해야 하는 경우 프록시 URL입니다. 프록시 서버는 HTTP 프록시 또는 HTTPS 프록시일 수 있습니다. HTTPS 프록시의 경우 프록시 서버에서 자체 서명된 인증서를 사용할 수 있도록 인증서 유효성 검사를 건너뛸 수 있습니다. 인증이 활성화된 프록시 서버는 지원되지 않습니다.	
nfsMountOptions	NFS 마운트 옵션에 대한 세밀한 제어	"nfsvers=3"
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다.	""(기본적으로 적용되지 않음)
serviceLevel	새 볼륨에 대한 CVS - 성능 또는 CVS 서비스 수준 CV - 성능 값은 입니다 standard, premium, 또는 extreme. CV 값은 입니다 standardsw 또는 zoneredundantstandardsw.	CV - 성능 기본값은 "표준"입니다. CV 기본값은 "standardsw"입니다.
network	Cloud Volumes Service 볼륨에 사용되는 Google Cloud 네트워크	"기본값"
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: \{"api":false, "method":true}. 문제 해결 중이 아니며 자세한 로그 덤프가 필요한 경우가 아니면 이 방법을 사용하지 마십시오.	null입니다
allowedTopologies	지역 간 액세스를 설정하려면 에 대한 StorageClass 정의를 사용합니다 allowedTopologies 모든 지역을 포함해야 합니다. 예를 들면 다음과 같습니다. - key: topology.kubernetes.io/region values: - us-east1 - europe-west1	

볼륨 프로비저닝 옵션

에서 기본 볼륨 프로비저닝을 제어할 수 있습니다 `defaults` 구성 파일의 섹션입니다.

매개 변수	설명	기본값
<code>exportRule</code>	새 볼륨의 내보내기 규칙. CIDR 표기법을 사용하여 IPv4 주소 또는 IPv4 서브넷의 조합을 심표로 구분해야 합니다.	"0.0.0.0/0"
<code>snapshotDir</code>	에 액세스합니다 <code>.snapshot</code> 디렉토리	"거짓"
<code>snapshotReserve</code>	스냅숏용으로 예약된 볼륨의 백분율입니다	""(CVS 기본값 0 허용)
<code>size</code>	새 볼륨의 크기입니다. CVS - 최소 성능은 100GiB입니다. CV 최소값은 1GiB입니다.	CVS - 성능 서비스 유형의 기본값은 "100GiB"입니다. CVS 서비스 유형은 기본값을 설정하지 않지만 최소 1GiB가 필요합니다.

CVS - 성능 서비스 유형의 예

다음 예에서는 CVS - 성능 서비스 유형에 대한 샘플 구성을 제공합니다.

예 1: 최소 구성

기본 CVS - 성능 서비스 유형과 기본 "표준" 서비스 수준을 사용하는 최소 백엔드 구성입니다.

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7O1wWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
  project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
```

```
token_uri: https://oauth2.googleapis.com/token
auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```



```
token_uri: https://oauth2.googleapis.com/token
auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

예 3: 가상 풀 구성

이 샘플은 를 사용합니다 storage 가상 풀 및 를 구성합니다 StorageClasses 다시 언급한다는 것입니다. 을 참조하십시오 스토리지 클래스 정의 스토리지 클래스를 정의한 방법을 확인합니다.

이 경우 를 설정하는 모든 가상 풀에 대해 특정 기본값이 설정됩니다 snapshotReserve 5% 및 에서 exportRule 를 0.0.0.0/0으로 설정합니다. 가상 풀은 에 정의되어 있습니다 storage 섹션을 참조하십시오. 각 개별 가상 풀은 고유한 가상 풀을 정의합니다 serviceLevel` 그리고 일부 풀은 기본값을 덮어씁니다. 가상 풀 레이블을 사용하여 에 따라 풀을 구분했습니다 `performance 및 protection.

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrtrHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    -----END PRIVATE KEY-----
```



```
XsYg6gyxy4zq70lwWgLwGa==
-----END PRIVATE KEY-----
client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
client_id: '123456789012345678901'
auth_uri: https://accounts.google.com/o/oauth2/auth
token_uri: https://oauth2.googleapis.com/token
auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
  performance: extreme
  protection: extra
  serviceLevel: extreme
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
- labels:
  performance: extreme
  protection: standard
  serviceLevel: extreme
- labels:
  performance: premium
  protection: extra
  serviceLevel: premium
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
- labels:
  performance: premium
  protection: standard
  serviceLevel: premium
- labels:
  performance: standard
  serviceLevel: standard
```

다음 StorageClass 정의는 가상 풀 구성 예에 적용됩니다. 사용 `parameters.selector` 볼륨을 호스팅하는 데 사용되는 가상 풀을 각 StorageClass에 대해 지정할 수 있습니다. 볼륨은 선택한 풀에 정의된 측면을 갖습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=extreme; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-standard-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium
provisioner: netapp.io/trident
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-standard
provisioner: netapp.io/trident
parameters:
  selector: "performance=standard"
```

```
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extra-protection
provisioner: netapp.io/trident
parameters:
  selector: "protection=extra"
allowVolumeExpansion: true
```

- 첫 번째 StorageClass입니다 (cvs-extreme-extra-protection)가 첫 번째 가상 풀에 매핑됩니다. 이 풀은 스냅샷 예약 공간이 10%인 최고 성능을 제공하는 유일한 풀입니다.
- 마지막 StorageClass입니다 (cvs-extra-protection) 10%의 스냅샷 예약 공간을 제공하는 스토리지 풀을 호출합니다. Astra Trident는 선택된 가상 풀을 결정하고 스냅샷 예약 요구 사항이 충족되는지 확인합니다.

CVS 서비스 유형 예

다음 예에서는 CVS 서비스 유형에 대한 샘플 구성을 제공합니다.

예 1: 최소 구성

을 사용하는 최소 백엔드 구성입니다 storageClass CVS 서비스 유형과 기본값을 지정합니다 standardsw 서비스 레벨:

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz1zZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq70lwWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
```

```
client_id: '123456789012345678901'  
auth_uri: https://accounts.google.com/o/oauth2/auth  
token_uri: https://oauth2.googleapis.com/token  
auth_provider_x509_cert_url:  
https://www.googleapis.com/oauth2/v1/certs  
client_x509_cert_url:  
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-  
sa%40my-gcp-project.iam.gserviceaccount.com  
serviceLevel: standardsw
```

예 2: 스토리지 풀 구성

이 백엔드 구성은 를 사용합니다 storagePools 스토리지 풀을 구성하려면 다음을 수행합니다.

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    MIIEvAIBADANBgkqhkiG9w0BAQEFAASCbKYwggSiAgEAAoIBAQDaT+Oui9FBAw19
    L1AGEkrYU5xd9K5NlO5jMkIFND5wCD+Nv+jd1GvtFRLaLK5RvXyF5wzvztMODNS+
    qtScpQ+5cFpQkuGtv9U9+N6qtuVYYO3b504Kp5CtqVPJCgMJAK2j8pZTIqUiMum/
    5/Y9oTbZrjAHSMgJm2nHzFq2X0rqVmaHghI6ATm4DOuWx8XGWKTGIPlc0qPqJlqS
    LLaWOH4VIZQZCAyW5IU9CAmwqHgdG0uhFNfCgMmED6PBUvVLsLvcq86X+QSWR9k
    ETqElj/sGCenPF7tilDhGBFafd9hPnxg9PZY29ArEZwY9G/ZjZQX7WPgs0VvxiNR
    DxZRC3GXAgMBAECggEACn5c59bG/qnVEVI1CwMAalM5M2z09JFh1L11jKwntNPj
    Vilw2eTW2+UE7HbJru/S7KQgA5Dnn9kvCraEahPRuddUMrD0vG4kTl/IODV6uFuk
    Y0sZfbqd4jMUQ21smvGsqFzwloYWS5qz01W83ivXH/HW/iqkmY2eW+EPRS/hwSSu
    SscR+SojI7PB0BWSJhlV4yqYf3vcD/D95e12CVHfRCkL85DKumeZ+yHENpiXGZAE
    t8xSs4a500Pm6NHhevCw2a/UQ95/foXNUR450HtbjieJo5o+FF6EYZQGfU2ZHZO8
    37FBKuaJkdGW5xqaI9TL7aqkGkFMF4F2qvOZM+vy8QKBgQD4oVuOkJD1hkTHP86W
    esFlw1kpWyJR9ZA7LI0g/rVpslnX+XdDq0WQf4umdLNau5hYEH9LU6ZSGs1Xk3/B
    NHwR6OXFuqEKNiu83d0zSlHhTy7PZpOZdj5a/vVvQfPDMz7OvsqLRd7YCAbdzuQ0
    +Ahq0Ztwvg0HQ64hdW0ukpYRRwKBgQDgyHj98oqswoYuIa+pPlyS0pPwLmjwKyNm
    /HayzCp+Qjiiyy7Tzg8AUq1H1Ou83XbV428jvg7kDhO7PCKfQ+mMmfqHmTpb0Maq
    KpKnZg4ipsqPlyHNNEoRmcailXbwIhCLewMqMrggUiLOmCw4PscL5nK+4GKu2XE1
    jLqjWAZFMQKBgFhkQ9XXRAJ1kR3XpGHoGN890pZOkCVSrqu6aUef/5KY1Fct8ew
    F/+aIxM2iQsvmWQYOvVCnhuY/F2GfAQ7d0om3decuwIOCX/xy7PjHMkLXa2uaZs4
    WR17sLduj62RqXRLX0c0QkwBiNFyHbRcpdkZJQujbYMhBa+7j7SxT4BtAoGAWMWT
    UucocRXZm/pdvz9wteNH3YDwnJLMxm1KC06qMXbBoYrliY4sm3ywJWMC+iCd/H8A
    Gecxd/xVu5mA2L2N3KMq18Zhz8Th0G5DwKyDRJgOQ0Q46yuNXOoYEjlo4WjyK8Me
    +t1Q8iK98E0UmZnhTgfSpSNElzbz2AqnzQ3MN9uECgYAqdvvdVPnKGFvdtZ2DjyMoJ
    E89UIC41WjjJGmHsd8W65+3X0RwMzKMT6aZc5tK9J5dHvmWIEtNbM+lTImdBFFga
    NWOC6f3r2xbGXHhaWSl+nobpTuvlo56ZRJVvVk7lFMsiddzMuHH8pxfgNjemWA4P
    ThDHCEjv035NNV6Kyo00tA==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
  data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
```

```
auth_uri: https://accounts.google.com/o/oauth2/auth
token_uri: https://oauth2.googleapis.com/token
auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

다음 단계

백엔드 구성 파일을 생성한 후 다음 명령을 실행합니다.

```
tridentctl create backend -f <backend-file>
```

백엔드 생성에 실패하면 백엔드 구성에 문제가 있는 것입니다. 다음 명령을 실행하여 로그를 보고 원인을 확인할 수 있습니다.

```
tridentctl logs
```

구성 파일의 문제를 확인하고 수정한 후 create 명령을 다시 실행할 수 있습니다.

NetApp HCI 또는 SolidFire 백엔드를 구성합니다

Astra Trident 설치와 함께 Element 백엔드를 생성하고 사용하는 방법에 대해 알아보십시오.

필요한 것

- Element 소프트웨어를 실행하는 지원되는 스토리지 시스템
- 볼륨을 관리할 수 있는 NetApp HCI/SolidFire 클러스터 관리자 또는 테넌트 사용자에게 대한 자격 증명
- 모든 Kubernetes 작업자 노드에 적절한 iSCSI 툴이 설치되어 있어야 합니다. 을 참조하십시오 ["작업자 노드 준비 정보"](#).

알아야 할 내용

를 클릭합니다 solidfire-san 스토리지 드라이버는 파일 및 블록의 볼륨 모드를 모두 지원합니다. 의 경우 Filesystem 볼륨 코드, Astra Trident가 볼륨을 생성하고 파일 시스템을 생성합니다. 파일 시스템 유형은 StorageClass에 의해 지정됩니다.

드라이버	프로토콜	볼륨 모드	액세스 모드가 지원됩니다	지원되는 파일 시스템
solidfire-san	iSCSI	블록	RWO, ROX, rwx	파일 시스템이 없습니다. 원시 블록 장치.
solidfire-san	iSCSI	블록	RWO, ROX, rwx	파일 시스템이 없습니다. 원시 블록 장치.
solidfire-san	iSCSI	파일 시스템	RWO, ROX	xfss, ext3, ext4
solidfire-san	iSCSI	파일 시스템	RWO, ROX	xfss, ext3, ext4



Astra Trident는 향상된 CSI Provisioner로 작동할 때 CHAP를 사용합니다. CHAP(CSI의 기본값)를 사용하는 경우 추가 준비가 필요하지 않습니다. 를 명시적으로 설정하는 것이 좋습니다 UseCHAP 비 CSI Trident와 CHAP를 사용하는 옵션입니다. 그렇지 않으면 를 참조하십시오 ["여기"](#).



볼륨 액세스 그룹은 Astra Trident의 기존 비 CSI 프레임워크에서만 지원됩니다. CSI 모드에서 작동하도록 구성된 경우 Astra Trident는 CHAP를 사용합니다.

둘 다 해당되지 않는 경우 AccessGroups 또는 UseCHAP 설정된 경우 다음 규칙 중 하나가 적용됩니다.

- 기본값 trident 액세스 그룹이 감지되고 액세스 그룹이 사용됩니다.
- 액세스 그룹이 감지되지 않고 Kubernetes 버전이 1.7 이상인 경우 CHAP가 사용됩니다.

백엔드 구성 옵션

백엔드 구성 옵션은 다음 표를 참조하십시오.

매개 변수	설명	기본값
version		항상 1
storageDriverName	스토리지 드라이버의 이름입니다	항상 "solidfire-san"
backendName	사용자 지정 이름 또는 스토리지 백엔드	"SolidFire_" + 스토리지(iSCSI) IP 주소입니다
Endpoint	테넌트 자격 증명이 있는 SolidFire 클러스터의 MVIP입니다	
SVIP	스토리지(iSCSI) IP 주소 및 포트	
labels	볼륨에 적용할 임의의 JSON 형식 레이블 세트입니다.	""
TenantName	사용할 테넌트 이름(찾을 수 없는 경우 생성됨)	

매개 변수	설명	기본값
InitiatorIFace	iSCSI 트래픽을 특정 호스트 인터페이스로 제한합니다	“기본값”
UseCHAP	CHAP를 사용하여 iSCSI를 인증합니다	참
AccessGroups	사용할 액세스 그룹 ID 목록입니다	"트리덴트"라는 액세스 그룹의 ID를 찾습니다.
Types	QoS 사양	
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다	“(기본적으로 적용되지 않음)
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: {"api":false, "method":true}	null입니다



사용하지 마십시오 debugTraceFlags 문제 해결 및 자세한 로그 덤프가 필요한 경우를 제외하고

예 1: 에 대한 백엔드 구성 solidfire-san 세 가지 볼륨 유형을 가진 드라이버

이 예에서는 CHAP 인증을 사용하는 백엔드 파일을 보여 주고 특정 QoS 보장을 포함하는 세 가지 볼륨 유형을 모델링합니다. 그런 다음 에서 각 스토리지 클래스를 사용할 스토리지 클래스를 정의할 가능성이 높습니다 IOPS 스토리지 클래스 매개 변수입니다.

```

---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: "<svip>:3260"
TenantName: "<tenant>"
labels:
  k8scluster: dev1
  backend: dev1-element-cluster
UseCHAP: true
Types:
- Type: Bronze
  Qos:
    minIOPS: 1000
    maxIOPS: 2000
    burstIOPS: 4000
- Type: Silver
  Qos:
    minIOPS: 4000
    maxIOPS: 6000
    burstIOPS: 8000
- Type: Gold
  Qos:
    minIOPS: 6000
    maxIOPS: 8000
    burstIOPS: 10000

```

예 2: 에 대한 백엔드 및 스토리지 클래스 구성 `solidfire-san` 가상 풀이 있는 드라이버

이 예에서는 가상 풀과 이를 다시 참조하는 `StorageClasses`와 함께 구성된 백엔드 정의 파일을 보여 줍니다.

Astra Trident는 스토리지 풀에 있는 레이블을 프로비저닝할 때 백엔드 스토리지 LUN에 복사합니다. 편의를 위해 스토리지 관리자는 가상 풀 및 그룹 볼륨별로 레이블을 레이블별로 정의할 수 있습니다.

아래 표시된 샘플 백엔드 정의 파일에서 특정 기본값은 를 설정하는 모든 스토리지 풀에 대해 설정됩니다 `type` 실버. 가상 풀은 에 정의되어 있습니다 `storage` 섹션을 참조하십시오. 이 예에서는 일부 스토리지 풀이 자체 유형을 설정하고 일부 풀은 위에 설정된 기본값을 덮어씁니다.

```

---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: "<svip>:3260"
TenantName: "<tenant>"
UseCHAP: true

```

```

Types:
- Type: Bronze
  Qos:
    minIOPS: 1000
    maxIOPS: 2000
    burstIOPS: 4000
- Type: Silver
  Qos:
    minIOPS: 4000
    maxIOPS: 6000
    burstIOPS: 8000
- Type: Gold
  Qos:
    minIOPS: 6000
    maxIOPS: 8000
    burstIOPS: 10000
type: Silver
labels:
  store: solidfire
  k8scluster: dev-1-cluster
region: us-east-1
storage:
- labels:
  performance: gold
  cost: '4'
  zone: us-east-1a
  type: Gold
- labels:
  performance: silver
  cost: '3'
  zone: us-east-1b
  type: Silver
- labels:
  performance: bronze
  cost: '2'
  zone: us-east-1c
  type: Bronze
- labels:
  performance: silver
  cost: '1'
  zone: us-east-1d

```

다음 StorageClass 정의는 위의 가상 풀을 참조합니다. 를 사용합니다 parameters.selector 필드에서 각 StorageClass는 볼륨을 호스팅하는 데 사용할 수 있는 가상 풀을 호출합니다. 선택한 가상 풀에 볼륨이 정의되어 있습니다.

첫 번째 StorageClass입니다 (solidfire-gold-four)는 첫 번째 가상 풀에 매핑합니다. 이 수영장은 골드 성능을 제공하는 유일한 수영장입니다 Volume Type QoS 않습니다. 마지막 StorageClass입니다 (solidfire-silver)은 뛰어난 성능을 제공하는 스토리지 풀을 호출합니다. Astra Trident가 선택한 가상 풀을 결정하고 스토리지 요구 사항을 충족시킵니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold-four
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=gold; cost=4"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-three
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver; cost=3"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=bronze; cost=2"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver; cost=1"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=silver"
  fsType: "ext4"
```

자세한 내용을 확인하십시오

- "볼륨 액세스 그룹"

ONTAP SAN 드라이버를 사용하여 백엔드를 구성합니다

ONTAP 및 Cloud Volumes ONTAP SAN 드라이버를 사용하여 ONTAP 백엔드를 구성하는 방법에 대해 알아보십시오.

- "준비"
- "구성 및 예"

Astra Control은 로 생성한 볼륨을 위해 원활한 보호, 재해 복구, 이동성(Kubernetes 클러스터 간에 볼륨 이동)을 제공합니다 `ontap-nas`, `ontap-nas-flexgroup`, 및 `ontap-san` 드라이버. 을 참조하십시오 "Astra Control 복제 사전 요구 사항" 를 참조하십시오.

- 를 사용해야 합니다 `ontap-nas` 데이터 보호, 재해 복구 및 이동성을 필요로 하는 운영 워크로드에 적합합니다.
- 사용 `ontap-san-economy` 예상되는 볼륨 사용량이 ONTAP가 지원하는 것보다 훨씬 많을 것으로 예상됩니다.
- 사용 `ontap-nas-economy` 예상되는 볼륨 사용량이 ONTAP에서 지원하는 것보다 훨씬 많을 것으로 예상되는 경우에만 및 입니다 `ontap-san-economy` 드라이버를 사용할 수 없습니다.
- 사용하지 마십시오 `ontap-nas-economy` 데이터 보호, 재해 복구 또는 이동성이 필요할 것으로 예상되는 경우



사용자 권한

Astra Trident는 일반적으로 를 사용하여 ONTAP 또는 SVM 관리자로 실행될 것으로 예상합니다 `admin` 클러스터 사용자 또는 입니다 `vsadmin` SVM 사용자 또는 동일한 역할을 가진 다른 이름의 사용자 NetApp ONTAP 구축을 위한 Amazon FSx의 경우, Astra Trident는 클러스터를 사용하여 ONTAP 또는 SVM 관리자로 실행될 것으로 예상합니다 `fsxadmin` 사용자 또는 `a vsadmin` SVM 사용자 또는 동일한 역할을 가진 다른 이름의 사용자 를 클릭합니다 `fsxadmin` 사용자는 클러스터 관리자를 제한적으로 대체합니다.



를 사용하는 경우 `limitAggregateUsage` 매개 변수, 클러스터 관리자 권한이 필요합니다. Astra Trident와 함께 NetApp ONTAP에 Amazon FSx를 사용하는 경우, 를 참조하십시오 `limitAggregateUsage` 매개 변수는 에서 작동하지 않습니다 `vsadmin` 및 `fsxadmin` 사용자 계정. 이 매개 변수를 지정하면 구성 작업이 실패합니다.

ONTAP 내에서 Trident 드라이버가 사용할 수 있는 보다 제한적인 역할을 만들 수 있지만 권장하지 않습니다. Trident의 대부분의 새로운 릴리즈에서는 추가 API를 호출하므로 업그레이드가 어렵고 오류가 발생하기 쉽습니다.

ONTAP SAN 드라이버를 사용하여 백엔드를 구성할 준비를 합니다

ONTAP SAN 드라이버를 사용하여 ONTAP 백엔드를 구성하는 방법에 대해 알아보십시오. 모든 ONTAP 백엔드의 경우, Astra Trident는 SVM에 하나 이상의 Aggregate가 할당되어 있어야 합니다.

또한 둘 이상의 드라이버를 실행하고 둘 중 하나를 가리키는 스토리지 클래스를 생성할 수도 있습니다. 예를 들어, 을 구성할 수 있습니다 `san-dev` 를 사용하는 클래스입니다 `ontap-san` 드라이버 및 `A san-default` 를 사용하는 클래스입니다 `ontap-san-economy` 1개.

모든 Kubernetes 작업자 노드에는 적절한 iSCSI 툴이 설치되어 있어야 합니다. 을 참조하십시오 ["여기"](#) 를 참조하십시오.

인증

Astra Trident는 ONTAP 백엔드를 인증하는 두 가지 모드를 제공합니다.

- 자격 증명 기반: 필요한 권한이 있는 ONTAP 사용자의 사용자 이름 및 암호입니다. 과 같이 미리 정의된 보안 로그인 역할을 사용하는 것이 좋습니다 `admin` 또는 `vsadmin` ONTAP 버전과의 호환성을 최대한 보장하기 위해
- 인증서 기반: Astra Trident는 백엔드에 설치된 인증서를 사용하여 ONTAP 클러스터와 통신할 수도 있습니다. 이 경우 백엔드 정의에는 클라이언트 인증서, 키 및 사용할 경우 신뢰할 수 있는 CA 인증서의 Base64로 인코딩된 값이 있어야 합니다(권장).

자격 증명 기반 방법과 인증서 기반 방법 간에 이동하기 위해 기존 백엔드를 업데이트할 수 있습니다. 그러나 한 번에 하나의 인증 방법만 지원됩니다. 다른 인증 방법으로 전환하려면 백엔드 구성에서 기존 방법을 제거해야 합니다.



자격 증명과 인증서 * 를 모두 제공하려고 하면 구성 파일에 둘 이상의 인증 방법이 제공된다는 오류가 발생하여 백엔드 생성이 실패합니다.

자격 증명 기반 인증을 사용합니다

Astra Trident는 SVM 범위/클러스터 범위 관리자에게 ONTAP 백엔드와 통신하기 위한 자격 증명을 요구합니다. 과 같이 미리 정의된 표준 역할을 사용하는 것이 좋습니다 `admin` 또는 `vsadmin`. 이를 통해 향후 Astra Trident 릴리스에서 사용할 기능 API를 노출할 수 있는 향후 ONTAP 릴리스와 향후 호환성이 보장됩니다. 사용자 지정 보안 로그인 역할은 Astra Trident와 함께 생성 및 사용할 수 있지만 권장되지 않습니다.

백엔드 정의의 예는 다음과 같습니다.

YAML

```
버전: 1 backendName: ExampleBackend storageDriverName: ONTAP-SAN 관리 LIF: 10.0.0.1
svm:svm_nfs 사용자 이름: vsadmin 암호: password
```

JSON을 참조하십시오

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password"
}
```

백엔드 정의는 자격 증명이 일반 텍스트로 저장되는 유일한 위치라는 점에 유의하십시오. 백엔드가 생성된 후 사용자 이름/암호는 Base64로 인코딩되어 Kubernetes 암호로 저장됩니다. 백엔드의 생성 또는 업데이트는 자격 증명에 대한

지식이 필요한 유일한 단계입니다. 따라서 Kubernetes/스토리지 관리자가 수행할 수 있는 관리 전용 작업입니다.

인증서 기반 인증을 사용합니다

신규 및 기존 백엔드는 인증서를 사용하여 ONTAP 백엔드와 통신할 수 있습니다. 백엔드 정의에는 세 가지 매개 변수가 필요합니다.

- `clientCertificate`: Base64로 인코딩된 클라이언트 인증서 값입니다.
- `clientPrivateKey`: Base64 - 연결된 개인 키의 인코딩된 값입니다.
- `TrustedCACertificate`: 신뢰할 수 있는 CA 인증서의 Base64 인코딩 값입니다. 신뢰할 수 있는 CA를 사용하는 경우 이 매개 변수를 제공해야 합니다. 신뢰할 수 있는 CA가 사용되지 않으면 이 작업을 무시할 수 있습니다.

일반적인 워크플로에는 다음 단계가 포함됩니다.

단계

1. 클라이언트 인증서 및 키를 생성합니다. 생성 시 CN(일반 이름)을 ONTAP 사용자로 설정하여 인증하십시오.

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=admin"
```

2. 신뢰할 수 있는 CA 인증서를 ONTAP 클러스터에 추가합니다. 이는 스토리지 관리자가 이미 처리한 것일 수 있습니다. 트러스트된 CA가 사용되지 않으면 무시합니다.

```
security certificate install -type server -cert-name <trusted-ca-cert-name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca <cert-authority>
```

3. ONTAP 클러스터에 클라이언트 인증서 및 키(1단계)를 설치합니다.

```
security certificate install -type client-ca -cert-name <certificate-name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

4. ONTAP 보안 로그인 역할이 지원되는지 확인합니다 cert 인증 방법.

```
security login create -user-or-group-name admin -application ontapi -authentication-method cert
security login create -user-or-group-name admin -application http -authentication-method cert
```

5. 생성된 인증서를 사용하여 인증을 테스트합니다. ONTAP 관리 LIF> 및 <SVM 이름>을 관리 LIF IP 및 SVM

이름으로 바꿉니다.

```
curl -X POST -Lk https://<ONTAP-Management-LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key --cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp xmlns="http://www.netapp.com/filer/admin" version="1.21" vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. Base64로 인증서, 키 및 신뢰할 수 있는 CA 인증서를 인코딩합니다.

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 이전 단계에서 얻은 값을 사용하여 백엔드를 생성합니다.

```
cat cert-backend.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "SanBackend",
  "managementLIF": "1.2.3.4",
  "svm": "vserver_test",
  "clientCertificate": "Faaaakkkkeeee...Vaaalllluuueeeee",
  "clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
  "trustedCACertificate": "QNFinfO...SiqOyN",
  "storagePrefix": "myPrefix_"
}

tridentctl create backend -f cert-backend.json -n trident
+-----+-----+-----+
+-----+-----+
|   NAME   | STORAGE DRIVER |           UUID           |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |         0 |
+-----+-----+-----+
+-----+-----+

```

인증 방법을 업데이트하거나 자격 증명을 회전합니다

다른 인증 방법을 사용하거나 자격 증명을 회전하도록 기존 백엔드를 업데이트할 수 있습니다. 이렇게 하면 사용자 이름/암호를 사용하는 백엔드를 인증서를 사용하도록 업데이트할 수 있고 인증서를 사용하는 백엔드는 사용자 이름/암호 기반으로 업데이트할 수 있습니다. 이렇게 하려면 기존 인증 방법을 제거하고 새 인증 방법을 추가해야 합니다. 그런 다음 실행할 필수 매개 변수가 포함된 업데이트된 backend.json 파일을 사용합니다 tridentctl backend update.

```
cat cert-backend-updated.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "SanBackend",
  "managementLIF": "1.2.3.4",
  "svm": "vserver_test",
  "username": "vsadmin",
  "password": "password",
  "storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident
+-----+-----+-----+-----+
+-----+-----+
|   NAME   | STORAGE DRIVER |                               UUID                               |
STATE   | VOLUMES   |
+-----+-----+-----+-----+
+-----+-----+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |          9 |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```



암호를 회전할 때 스토리지 관리자는 먼저 ONTAP에서 사용자의 암호를 업데이트해야 합니다. 그 다음에는 백엔드 업데이트가 있습니다. 인증서를 회전할 때 여러 인증서를 사용자에게 추가할 수 있습니다. 그런 다음 백엔드가 업데이트되어 새 인증서를 사용합니다. 그러면 ONTAP 클러스터에서 이전 인증서를 삭제할 수 있습니다.

백엔드를 업데이트해도 이미 생성된 볼륨에 대한 액세스가 중단되거나 이후에 생성된 볼륨 연결에 영향을 미치지 않습니다. 백엔드 업데이트가 성공적이면 Astra Trident가 ONTAP 백엔드와 통신하고 향후 볼륨 작업을 처리할 수 있음을 나타냅니다.

igroup을 지정합니다

Astra Trident에서 igroup을 사용하여 프로비저닝하는 볼륨(LUN)에 대한 액세스를 제어합니다. 관리자는 백엔드에 대한 igroup을 지정할 때 다음 두 가지 옵션을 사용할 수 있습니다.

- Astra Trident는 백엔드에 따라 igroup을 자동으로 생성하고 관리할 수 있습니다. If(경우 igroupName 백엔드 정의에 포함되지 않은 Astra Trident는 이름을 가진 igroup을 생성합니다 trident-<backend-UUID> SVM에서. 그러면 각 백엔드에 전용 igroup이 있고 Kubernetes 노드 IQN의 자동 추가/삭제를 처리합니다.
- 또는 미리 생성된 igroup을 백엔드 정의로 제공할 수도 있습니다. 이 작업은 를 사용하여 수행할 수 있습니다 igroupName 구성 매개 변수입니다. Astra Trident가 기존 igroup에 Kubernetes 노드 IQN을 추가/삭제합니다.

을(를) 가진 백엔드의 경우 igroupName 정의, 입니다 igroupName 을 사용하여 삭제할 수 있습니다 tridentctl backend update Astra Trident에서 igroup 자동 처리 이 경우 워크로드에 이미 연결된 볼륨에 대한 액세스가 중단되지 않습니다. 생성된 igroup Astra Trident를 사용하여 향후 연결을 처리할 것입니다.



Astra Trident의 각 고유 인스턴스에 대해 igroup을 할당하는 것은 Kubernetes 관리자 및 스토리지 관리자에게 유용한 모범 사례입니다. CSI Trident는 클러스터 노드 IQN을 igroup에 추가 및 제거하여 관리를 크게 단순화합니다. 전용 igroup을 사용하여 Kubernetes 환경(및 Astra Trident 설치)에서 동일한 SVM을 사용할 경우 한 Kubernetes 클러스터의 변경 사항이 다른 Kubernetes 클러스터와 관련된 igroup에 영향을 미치지 않도록 합니다. 또한 Kubernetes 클러스터의 각 노드에 고유한 IQN이 있는지 확인하는 것도 중요합니다. 위에 언급한 바와 같이, Astra Trident는 IQN의 추가 및 제거를 자동으로 처리합니다. 호스트 간에 IQN을 재사용하면 호스트가 서로 잘못 인식되어 LUN에 대한 액세스가 거부되는 바람직하지 않은 시나리오가 발생할 수 있습니다.

Astra Trident가 CSI Provisioner로 작동하도록 구성된 경우 Kubernetes 노드 IQN이 igroup에 자동으로 추가/제거됩니다. Kubernetes 클러스터에 노드를 추가할 경우, trident-csi DemonSet가 포드를 전개합니다 (trident-csi-xxxxxx 23.01 또는 이전 버전 trident-node<operating system>-xxxx 23.01 이상에서)를 새로 추가한 노드에 추가하고 볼륨을 연결할 수 있는 새 노드를 등록합니다. 노드 IQN도 백엔드의 igroup에 추가됩니다. 이와 유사한 일련의 단계에서는 Kubernetes에서 노드에 코드로닝, 트레이닝 및 삭제가 발생하는 경우 IQN 제거를 처리합니다.

Astra Trident가 CSI Provisioner로 실행되지 않을 경우, Kubernetes 클러스터의 모든 작업자 노드에서 iSCSI IQN을 포함하도록 igroup을 수동으로 업데이트해야 합니다. Kubernetes 클러스터에 참여하는 노드의 IQN을 igroup에 추가해야 합니다. 마찬가지로, Kubernetes 클러스터에서 제거된 노드의 IQN을 igroup에서 제거해야 합니다.

양방향 **CHAP**를 사용하여 연결을 인증합니다

Astra Trident는 의 양방향 CHAP를 사용하여 iSCSI 세션을 인증할 수 있습니다 ontap-san 및 ontap-san-economy 드라이버. 이를 위해서는 가 활성화되어야 합니다 useCHAP 백엔드 정의에서 선택할 수 있습니다. 를 로 설정한 경우 true, Astra Trident는 SVM의 기본 이니시에이터 보안을 양방향 CHAP로 구성하고 백엔드 파일의 사용자 이름과 암호를 설정합니다. 양방향 CHAP를 사용하여 연결을 인증하는 것이 좋습니다. 다음 샘플 구성을 참조하십시오.

```

---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
igroupName: trident
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz

```



를 클릭합니다 useCHAP 매개 변수는 한 번만 구성할 수 있는 부울 옵션입니다. 기본적으로 false로 설정되어 있습니다. true 로 설정한 후에는 false 로 설정할 수 없습니다.

또한 useCHAP=true, chapInitiatorSecret, chapTargetInitiatorSecret, chapTargetUsername, 및 chapUsername 필드는 백엔드 정의에 포함되어야 합니다. 을 실행하여 백엔드를 생성한 후 암호를 변경할 수 있습니다 tridentctl update.

작동 방식

설정을 통해 useCHAP 스토리지 관리자는 스토리지 백엔드에서 CHAP를 구성하도록 Astra Trident에 지시합니다. 여기에는 다음이 포함됩니다.

- SVM에서 CHAP 설정:
 - SVM의 기본 이니시에이터 보안 유형이 없음(기본값 설정) * 이고 * 볼륨에 이미 기존 LUN이 없으면 Astra Trident가 기본 보안 유형을 로 설정합니다 CHAP CHAP 이니시에이터와 타겟 사용자 이름 및 암호 구성 을 진행합니다.
 - SVM에 LUN이 포함된 경우 Astra Trident는 SVM에서 CHAP를 활성화하지 않습니다. 따라서 SVM에 이미 있는 LUN에 대한 액세스가 제한되지 않습니다.
- CHAP 이니시에이터 및 타겟 사용자 이름과 암호를 구성합니다. 이러한 옵션은 백엔드 구성에 지정해야 합니다(위 참조).
- 에 이니시에이터 추가 관리 igroupName 백엔드에서 제공됩니다. 지정되지 않은 경우 이 기본값은 입니다 trident.

백엔드가 생성된 후 Astra Trident가 해당 을 생성합니다 tridentbackend CHAP 암호 및 사용자 이름을 Kubernetes 비밀로 CRD 및 저장합니다. 이 백엔드에서 Astra Trident에 의해 생성된 모든 PVS는 CHAP를 통해 마운트되고 연결됩니다.

자격 증명을 회전하고 백엔드를 업데이트합니다

에서 CHAP 매개 변수를 업데이트하여 CHAP 자격 증명을 업데이트할 수 있습니다 backend.json 파일. CHAP 암호를 업데이트하고 를 사용해야 합니다 tridentctl update 명령을 사용하여 이러한 변경 사항을 반영합니다.



백엔드의 CHAP 암호를 업데이트할 때 를 사용해야 합니다 tridentctl 백엔드를 업데이트합니다. Astra Trident에서 변경 사항을 선택할 수 없으므로 CLI/ONTAP UI를 통해 스토리지 클러스터의 자격 증명을 업데이트하지 마십시오.

```
cat backend-san.json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "ontap_san_chap",
  "managementLIF": "192.168.0.135",
  "svm": "ontap_iscsi_svm",
  "useCHAP": true,
  "username": "vsadmin",
  "password": "password",
  "igroupName": "trident",
  "chapInitiatorSecret": "cl9qxUpDaTeD",
  "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
  "chapTargetUsername": "iJF4heBRT0TCwxyz",
  "chapUsername": "uh2aNCLSD6cNwxyz",
}

./tridentctl update backend ontap_san_chap -f backend-san.json -n trident
+-----+-----+-----+-----+
+-----+-----+
|  NAME          | STORAGE DRIVER |          UUID          |
STATE | VOLUMES |
+-----+-----+-----+-----+
+-----+-----+
| ontap_san_chap | ontap-san      | aa458f3b-ad2d-4378-8a33-1a472ffbeb5c |
online |        7 |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

기존 연결은 영향을 받지 않습니다. SVM에서 Astra Trident가 자격 증명을 업데이트하면 활성 상태로 유지됩니다. 새 연결은 업데이트된 자격 증명을 사용하며 기존 연결은 계속 활성 상태로 유지됩니다. 기존 PVS를 연결 해제하고 다시 연결하면 업데이트된 자격 증명을 사용하게 됩니다.

ONTAP SAN 구성 옵션 및 예

Astra Trident 설치를 통해 ONTAP SAN 드라이버를 생성하고 사용하는 방법에 대해 알아보십시오. 이 섹션에서는 백엔드 구성 예제 및 백엔드를 StorageClasses에 매핑하는 방법에 대한 세부 정보를 제공합니다.

백엔드 구성 옵션

백엔드 구성 옵션은 다음 표를 참조하십시오.

매개 변수	설명	기본값
version		항상 1
storageDriverName	스토리지 드라이버의 이름입니다	"ONTAP-NAS", "ONTAP-NAS-이코노미", "ONTAP-NAS-Flexgroup", "ONTAP-SAN", "ONTAP-SAN-이코노미"
backendName	사용자 지정 이름 또는 스토리지 백엔드	드라이버 이름 + "_" + dataLIF
managementLIF	클러스터 또는 SVM 관리 LIF의 IP 주소 MetroCluster의 원활한 전환을 위해 SVM 관리 LIF를 지정해야 합니다. FQDN(정규화된 도메인 이름)을 지정할 수 있습니다. 을 사용하여 Astra Trident가 설치된 경우 IPv6 주소를 사용하도록 설정할 수 있습니다 --use-ipv6 깃발. IPv6 주소는 대괄호로 묶어야 합니다(예: [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]).	"10.0.0.1", "[2001:1234:ABCD::fee]"
dataLIF	프로토콜 LIF의 IP 주소입니다. * iSCSI에 대해서는 지정하지 마십시오. * Astra Trident가 사용합니다 "ONTAP 선택적 LUN 맵" 다중 경로 세션을 설정하는 데 필요한 iSCSI LIF를 검색하려면 경고 발생 시 dataLIF 명시적으로 정의됩니다.	SVM에서 파생됩니다
useCHAP	CHAP를 사용하여 ONTAP SAN 드라이버에 대한 iSCSI 인증 [Boolean]. 를 로 설정합니다 true Astra Trident에서 백엔드에 제공된 SVM에 대한 기본 인증으로 양방향 CHAP를 구성하고 사용합니다. 을 참조하십시오 "ONTAP SAN 드라이버를 사용하여 백엔드를 구성할 준비를 합니다" 를 참조하십시오.	거짓
chapInitiatorSecret	CHAP 이니시에이터 암호입니다. 필요한 경우 useCHAP=true	""
labels	볼륨에 적용할 임의의 JSON 형식 레이블 세트입니다	""
chapTargetInitiatorSecret	CHAP 타겟 이니시에이터 암호입니다. 필요한 경우 useCHAP=true	""
chapUsername	인바운드 사용자 이름입니다. 필요한 경우 useCHAP=true	""
chapTargetUsername	대상 사용자 이름입니다. 필요한 경우 useCHAP=true	""

매개 변수	설명	기본값
clientCertificate	Base64 - 클라이언트 인증서의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
clientPrivateKey	Base64 - 클라이언트 개인 키의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
trustedCACertificate	Base64 - 신뢰할 수 있는 CA 인증서의 인코딩된 값입니다. 선택 사항. 인증서 기반 인증에 사용됩니다.	""
username	ONTAP 클러스터와 통신하는 데 필요한 사용자 이름입니다. 자격 증명 기반 인증에 사용됩니다.	""
password	ONTAP 클러스터와 통신하는 데 필요한 암호입니다. 자격 증명 기반 인증에 사용됩니다.	""
svm	사용할 스토리지 가상 머신입니다	SVM에서 파생된 경우 managementLIF 이(가) 지정되었습니다
igroupName	사용할 SAN 볼륨에 대한 igroup의 이름입니다. 을 참조하십시오 를 참조하십시오.	"삼중 - <backend-UUID>"
storagePrefix	SVM에서 새 볼륨을 프로비저닝할 때 사용되는 접두사 나중에서 수정할 수 없습니다. 이 매개 변수를 업데이트하려면 새 백엔드를 생성해야 합니다.	"삼중류"
limitAggregateUsage	사용량이 이 비율을 초과하면 프로비저닝이 실패합니다. NetApp ONTAP 백엔드에 Amazon FSx를 사용하는 경우 를 지정하지 마십시오 limitAggregateUsage. 제공 fsxadmin 및 vsadmin 애그리게이트 사용을 검색하고 Astra Trident를 사용하여 제한하는 데 필요한 권한이 포함되어 있지 않습니다.	""(기본적으로 적용되지 않음)
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다. 또한 qtree 및 LUN에 대해 관리하는 볼륨의 최대 크기도 제한합니다.	""(기본적으로 적용되지 않음)
lunsPerFlexvol	FlexVol당 최대 LUN 수는 범위[50, 200]에 있어야 합니다.	"100"

매개 변수	설명	기본값
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: {"api":false, "method":true} 문제 해결 및 자세한 로그 덤프가 필요한 경우가 아니면 사용하지 마십시오.	null입니다
useREST	ONTAP REST API를 사용하는 부울 매개 변수입니다. * 기술 미리 보기 * useREST 프로덕션 작업 부하가 아닌 테스트 환경에 권장되는 기술 미리 보기로 제공됩니다. 를 로 설정한 경우 true, Astra Trident는 ONTAP REST API를 사용하여 백엔드와 통신합니다. 이 기능을 사용하려면 ONTAP 9.11.1 이상이 필요합니다. 또한 사용되는 ONTAP 로그인 역할에 대한 액세스 권한이 있어야 합니다 ontap 응용 프로그램. 이는 사전 정의된 에 의해 충족됩니다 vsadmin 및 cluster-admin 역할. useREST 는 MetroCluster에서 지원되지 않습니다.	거짓

에 대한 세부 정보 igroupName

igroupName ONTAP 클러스터에서 이미 생성된 igroup으로 설정할 수 있습니다. 지정되지 않은 경우 Astra Trident가 이름이 인 igroup을 자동으로 생성합니다 trident-<backend-UUID>.

미리 정의된 횡수 이름을 제공하는 경우 환경 간에 SVM을 공유하려면 Kubernetes 클러스터 당 하나의 igroup을 사용하는 것이 좋습니다. 이는 Astra Trident가 IQN 추가 및 삭제를 자동으로 유지 관리하는 데 필요합니다.

- igroupName Astra Trident 외부의 SVM에서 생성 및 관리되는 새로운 igroup을 가리키도록 업데이트할 수 있습니다.
- igroupName 생략할 수 있습니다. 이 경우 Astra Trident가 이름을 가진 igroup을 생성하고 관리합니다 trident-<backend-UUID> 자동으로.

두 경우 모두 볼륨 첨부 파일에 계속 액세스할 수 있습니다. 향후 볼륨 첨부 파일은 업데이트된 igroup을 사용합니다. 이 업데이트는 백엔드에 있는 볼륨에 대한 액세스를 방해하지 않습니다.

볼륨 프로비저닝을 위한 백엔드 구성 옵션

에서 이러한 옵션을 사용하여 기본 프로비저닝을 제어할 수 있습니다 defaults 섹션을 참조하십시오. 예를 들어, 아래 구성 예제를 참조하십시오.

매개 변수	설명	기본값
spaceAllocation	LUN에 대한 공간 할당	"참"
spaceReserve	공간 예약 모드, "없음"(씬) 또는 "볼륨"(일반)	"없음"
snapshotPolicy	사용할 스냅샷 정책입니다	"없음"

매개 변수	설명	기본값
qosPolicy	생성된 볼륨에 할당할 QoS 정책 그룹입니다. 스토리지 풀/백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다. Astra Trident와 함께 QoS 정책 그룹을 사용하려면 ONTAP 9.8 이상이 필요합니다. 비공유 QoS 정책 그룹을 사용하고 정책 그룹이 각 구성요소별로 적용되도록 하는 것이 좋습니다. 공유 QoS 정책 그룹은 모든 워크로드의 총 처리량에 대해 상한을 적용합니다.	""
adaptiveQosPolicy	생성된 볼륨에 할당할 적응형 QoS 정책 그룹입니다. 스토리지 풀/백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다	""
snapshotReserve	스냅샷 "0"에 예약된 볼륨의 백분율	If(경우 snapshotPolicy "없음"이고, 그렇지 않으면 ""입니다.
splitOnClone	생성 시 상위 클론에서 클론을 분할합니다	"거짓"
encryption	새 볼륨에 NVE(NetApp Volume Encryption)를 활성화합니다. 기본값은 입니다 false. 이 옵션을 사용하려면 NVE 라이선스가 클러스터에서 활성화되어 있어야 합니다. 백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다. 자세한 내용은 다음을 참조하십시오. "Astra Trident가 NVE 및 NAE와 연동되는 방식" .	"거짓"
luksEncryption	LUKS 암호화를 사용합니다. 을 참조하십시오 "LUKS(Linux Unified Key Setup) 사용" .	""
securityStyle	새로운 볼륨에 대한 보안 스타일	unix
tieringPolicy	"없음"을 사용하는 계층화 정책	ONTAP 9.5 이전 SVM-DR 구성의 경우 "스냅샷 전용

볼륨 프로비저닝의 예

다음은 기본값이 정의된 예입니다.

```

---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: password
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
igroupName: custom
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: standard
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'

```



를 사용하여 생성된 모든 볼륨에 대해 `ontap-san` 드라이버, Astra Trident는 FlexVol에 10%의 용량을 추가하여 LUN 메타데이터를 수용합니다. LUN은 사용자가 PVC에서 요청하는 정확한 크기로 프로비저닝됩니다. Astra Trident가 FlexVol에 10%를 더합니다(ONTAP에서 사용 가능한 크기로 표시). 이제 사용자가 요청한 가용 용량을 얻을 수 있습니다. 또한 이 변경으로 인해 사용 가능한 공간이 완전히 활용되지 않는 한 LUN이 읽기 전용이 되는 것을 방지할 수 있습니다. ONTAP-SAN-경제에는 적용되지 않습니다.

을 정의하는 백엔드의 경우 `snapshotReserve`, Astra Trident는 다음과 같이 볼륨의 크기를 계산합니다.

$$\text{Total volume size} = [(\text{PVC requested size}) / (1 - (\text{snapshotReserve percentage}) / 100)] * 1.1$$

1.1은 LUN 메타데이터를 수용하도록 FlexVol에 추가된 10%의 Astra Trident입니다. 용 `snapshotReserve = 5%`, PVC 요청 = 5GiB인 경우 총 볼륨 크기는 5.79GiB이고 사용 가능한 크기는 5.5GiB입니다. 를 클릭합니다 `volume show` 명령은 이 예제와 유사한 결과를 표시해야 합니다.

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e42ec6fe_3baa_4af6_996d_134adbbb8e6d	online	RW	5.79GB	5.50GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

3 entries were displayed.

현재 기존 볼륨에 대해 새 계산을 사용하는 유일한 방법은 크기 조정입니다.

최소 구성의 예

다음 예에서는 대부분의 매개 변수를 기본값으로 두는 기본 구성을 보여 줍니다. 이는 백엔드를 정의하는 가장 쉬운 방법입니다.



Astra Trident가 있는 NetApp ONTAP에서 Amazon FSx를 사용하는 경우 IP 주소 대신 LIF에 대한 DNS 이름을 지정하는 것이 좋습니다.

ontap-san 인증서 기반 인증을 사용하는 드라이버

이는 최소 백엔드 구성의 예입니다. `clientCertificate`, `clientPrivateKey`, 및 `trustedCACertificate` (신뢰할 수 있는 CA를 사용하는 경우 선택 사항)는 에 채워집니다 `backend.json` 그리고 각각 클라이언트 인증서, 개인 키 및 신뢰할 수 있는 CA 인증서의 base64로 인코딩된 값을 사용합니다.

```
---
version: 1
storageDriverName: ontap-san
backendName: DefaultSANBackend
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
igroupName: trident
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

ontap-san 양방향 CHAP가 있는 드라이버

이는 최소 백엔드 구성의 예입니다. 이 기본 구성은 를 생성합니다 `ontap-san` 을(를) 이용한 백엔드 `useCHAP` 를 로 설정합니다 `true`.

```

---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
igroupName: trident
username: vsadmin
password: password

```

ontap-san-economy 드라이버

```

---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
igroupName: trident
username: vsadmin
password: password

```

가상 풀의 백엔드 예

아래 표시된 샘플 백엔드 정의 파일에서와 같은 모든 스토리지 풀에 대한 특정 기본값이 설정됩니다. spaceReserve 없음, spaceAllocation 거짓일 경우, 및 encryption 거짓일 때. 가상 풀은 스토리지 섹션에 정의됩니다.

Astra Trident는 "Comments" 필드에 제공 레이블을 설정합니다. FlexVol에 주석이 설정됩니다. Astra Trident는 프로비저닝할 때 가상 풀에 있는 모든 레이블을 스토리지 볼륨에 복사합니다. 편의를 위해 스토리지 관리자는 가상 풀 및 그룹 볼륨별로 레이블을 레이블별로 정의할 수 있습니다.

이 예에서는 일부 스토리지 풀이 자체적으로 설정됩니다. spaceReserve, spaceAllocation, 및 encryption 값 및 일부 풀이 위에 설정된 기본값을 덮어씁니다.

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
igroupName: trident
username: vsadmin
password: password
defaults:
  spaceAllocation: 'false'
  encryption: 'false'
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
- labels:
  protection: gold
  creditpoints: '40000'
  zone: us_east_1a
  defaults:
    spaceAllocation: 'true'
    encryption: 'true'
    adaptiveQosPolicy: adaptive-extreme
- labels:
  protection: silver
  creditpoints: '20000'
  zone: us_east_1b
  defaults:
    spaceAllocation: 'false'
    encryption: 'true'
    qosPolicy: premium
- labels:
  protection: bronze
  creditpoints: '5000'
  zone: us_east_1c
  defaults:
    spaceAllocation: 'true'
    encryption: 'false'
```

iSCSI의 예는 다음과 같습니다 ontap-san-economy 드라이버:

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSD6cNwxyz
igroupName: trident
username: vsadmin
password: password
defaults:
  spaceAllocation: 'false'
  encryption: 'false'
labels:
  store: san_economy_store
region: us_east_1
storage:
- labels:
  app: oracledb
  cost: '30'
  zone: us_east_1a
  defaults:
    spaceAllocation: 'true'
    encryption: 'true'
- labels:
  app: postgresdb
  cost: '20'
  zone: us_east_1b
  defaults:
    spaceAllocation: 'false'
    encryption: 'true'
- labels:
  app: mysqldb
  cost: '10'
  zone: us_east_1c
  defaults:
    spaceAllocation: 'true'
    encryption: 'false'
```

백엔드를 **StorageClasses**에 매핑합니다

다음 StorageClass 정의는 위의 가상 풀을 참조합니다. 를 사용합니다 `parameters.selector` 필드에서 각 StorageClass는 볼륨을 호스팅하는 데 사용할 수 있는 가상 풀을 호출합니다. 선택한 가상 풀에 볼륨이 정의되어 있습니다.

- 첫 번째 StorageClass입니다 (`protection-gold`)는 의 첫 번째, 두 번째 가상 풀에 매핑됩니다 `ontap-nas-flexgroup` 백엔드 및 의 첫 번째 가상 풀 `ontap-san` 백엔드. 골드 레벨 보호 기능을 제공하는 유일한 풀입니다.
- 두 번째 StorageClass입니다 (`protection-not-gold`)는 에서 세 번째, 네 번째 가상 풀에 매핑됩니다 `ontap-nas-flexgroup` 백엔드 및 에서 세 번째 가상 풀입니다 `ontap-san` 백엔드. 금 이외의 보호 수준을 제공하는 유일한 풀입니다.
- 세 번째 StorageClass입니다 (`app-mysqldb`)는 에서 네 번째 가상 풀에 매핑됩니다 `ontap-nas` 에 있는 백엔드 및 세 번째 가상 풀입니다 `ontap-san-economy` 백엔드. `mysqldb` 유형 앱에 대한 스토리지 풀 구성을 제공하는 유일한 풀입니다.
- 네 번째 StorageClass입니다 (`protection-silver-creditpoints-20k`)는 의 세 번째 가상 풀에 매핑됩니다 `ontap-nas-flexgroup` 의 백엔드 및 두 번째 가상 풀입니다 `ontap-san` 백엔드. 20000 크레딧 포인트에서 골드 레벨 보호 기능을 제공하는 유일한 풀입니다.
- 다섯 번째 StorageClass입니다 (`creditpoints-5k`)는 의 두 번째 가상 풀에 매핑됩니다 `ontap-nas-economy` 에 있는 백엔드 및 세 번째 가상 풀입니다 `ontap-san` 백엔드. 5000 크레딧 포인트에 있는 유일한 풀 서비스입니다.

Astra Trident가 선택한 가상 풀을 결정하고 스토리지 요구 사항을 충족시킵니다.


```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"

```

ONTAP NAS 백엔드를 구성합니다

ONTAP 및 Cloud Volumes ONTAP NAS 드라이버를 사용하여 ONTAP 백엔드를 구성하는 방법에 대해 알아보십시오.

- "준비"
- "구성 및 예"

Astra Control은 로 생성한 볼륨을 위해 원활한 보호, 재해 복구, 이동성(Kubernetes 클러스터 간에 볼륨 이동)을 제공합니다. `ontap-nas`, `ontap-nas-flexgroup`, 및 `ontap-san` 드라이버. 을 참조하십시오 ["Astra Control 복제 사전 요구 사항"](#) 를 참조하십시오.



- 를 사용해야 합니다 `ontap-nas` 데이터 보호, 재해 복구 및 이동성을 필요로 하는 운영 워크로드에 적합합니다.
- 사용 `ontap-san-economy` 예상되는 볼륨 사용량이 ONTAP가 지원하는 것보다 훨씬 많을 것으로 예상됩니다.
- 사용 `ontap-nas-economy` 예상되는 볼륨 사용량이 ONTAP에서 지원하는 것보다 훨씬 많을 것으로 예상되는 경우에만 및 입니다 `ontap-san-economy` 드라이버를 사용할 수 없습니다.
- 사용하지 마십시오 `ontap-nas-economy` 데이터 보호, 재해 복구 또는 이동성이 필요할 것으로 예상되는 경우

사용자 권한

Astra Trident는 일반적으로 를 사용하여 ONTAP 또는 SVM 관리자로 실행될 것으로 예상합니다 `admin` 클러스터 사용자 또는 입니다 `vsadmin` SVM 사용자 또는 동일한 역할을 가진 다른 이름의 사용자 NetApp ONTAP 구축을 위한 Amazon FSx의 경우, Astra Trident는 클러스터를 사용하여 ONTAP 또는 SVM 관리자로 실행될 것으로 예상합니다 `fsxadmin` 사용자 또는 `a vsadmin` SVM 사용자 또는 동일한 역할을 가진 다른 이름의 사용자 를 클릭합니다 `fsxadmin` 사용자는 클러스터 관리자를 제한적으로 대체합니다.



를 사용하는 경우 `limitAggregateUsage` 매개 변수, 클러스터 관리자 권한이 필요합니다. Astra Trident와 함께 NetApp ONTAP에 Amazon FSx를 사용하는 경우, 를 참조하십시오 `limitAggregateUsage` 매개 변수는 에서 작동하지 않습니다 `vsadmin` 및 `fsxadmin` 사용자 계정. 이 매개 변수를 지정하면 구성 작업이 실패합니다.

ONTAP 내에서 Trident 드라이버가 사용할 수 있는 보다 제한적인 역할을 만들 수 있지만 권장하지 않습니다. Trident의 대부분의 새로운 릴리즈에서는 추가 API를 호출하므로 업그레이드가 어렵고 오류가 발생하기 쉽습니다.

ONTAP NAS 드라이버를 사용하여 백엔드를 구성할 준비를 합니다

ONTAP NAS 드라이버를 사용하여 ONTAP 백엔드를 구성하는 방법에 대해 알아보십시오. 모든 ONTAP 백엔드의 경우, Astra Trident는 SVM에 하나 이상의 Aggregate가 할당되어 있어야 합니다.

모든 ONTAP 백엔드의 경우, Astra Trident는 SVM에 하나 이상의 Aggregate가 할당되어 있어야 합니다.

또한 둘 이상의 드라이버를 실행하고 둘 중 하나를 가리키는 스토리지 클래스를 생성할 수도 있습니다. 예를 들어, 을 사용하는 Gold 클래스를 구성할 수 있습니다 `ontap-nas` 드라이버 및 를 사용하는 Bronze 클래스 `ontap-nas-economy` 1개.

모든 Kubernetes 작업자 노드에 적절한 NFS 톨이 설치되어 있어야 합니다. 을 참조하십시오 ["여기"](#) 를 참조하십시오.

인증

Astra Trident는 ONTAP 백엔드를 인증하는 두 가지 모드를 제공합니다.

- 자격 증명 기반: 필요한 권한이 있는 ONTAP 사용자의 사용자 이름 및 암호입니다. 과 같이 미리 정의된 보안 로그인 역할을 사용하는 것이 좋습니다 `admin` 또는 `vsadmin` ONTAP 버전과의 호환성을 최대한 보장하기 위해
- 인증서 기반: Astra Trident는 백엔드에 설치된 인증서를 사용하여 ONTAP 클러스터와 통신할 수도 있습니다. 이 경우 백엔드 정의에는 클라이언트 인증서, 키 및 사용할 경우 신뢰할 수 있는 CA 인증서의 Base64로 인코딩된 값이 있어야 합니다(권장).

자격 증명 기반 방법과 인증서 기반 방법 간에 이동하기 위해 기존 백엔드를 업데이트할 수 있습니다. 그러나 한 번에 하나의 인증 방법만 지원됩니다. 다른 인증 방법으로 전환하려면 백엔드 구성에서 기존 방법을 제거해야 합니다.



자격 증명과 인증서 * 를 모두 제공하려고 하면 구성 파일에 둘 이상의 인증 방법이 제공된다는 오류가 발생하여 백엔드 생성이 실패합니다.

자격 증명 기반 인증을 사용합니다

Astra Trident는 SVM 범위/클러스터 범위 관리자에게 ONTAP 백엔드와 통신하기 위한 자격 증명을 요구합니다. 과 같이 미리 정의된 표준 역할을 사용하는 것이 좋습니다 `admin` 또는 `vsadmin`. 이를 통해 향후 Astra Trident 릴리스에서 사용할 기능 API를 노출할 수 있는 향후 ONTAP 릴리스와 향후 호환성이 보장됩니다. 사용자 지정 보안 로그인 역할은 Astra Trident와 함께 생성 및 사용할 수 있지만 권장되지 않습니다.

백엔드 정의의 예는 다음과 같습니다.

YAML

```
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

JSON을 참조하십시오

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password"
}
```

백엔드 정의는 자격 증명에 일반 텍스트로 저장되는 유일한 위치라는 점에 유의하십시오. 백엔드가 생성된 후 사용자 이름/암호는 Base64로 인코딩되어 Kubernetes 암호로 저장됩니다. 백엔드의 생성/업데이트는 자격 증명에 대한 지식이 필요한 유일한 단계입니다. 따라서 Kubernetes/스토리지 관리자가 수행할 수 있는 관리 전용 작업입니다.

인증서 기반 인증을 사용합니다

신규 및 기존 백엔드는 인증서를 사용하여 ONTAP 백엔드와 통신할 수 있습니다. 백엔드 정의에는 세 가지 매개 변수가 필요합니다.

- `clientCertificate`: Base64로 인코딩된 클라이언트 인증서 값입니다.
- `clientPrivateKey`: Base64 - 연결된 개인 키의 인코딩된 값입니다.
- `TrustedCACertificate`: 신뢰할 수 있는 CA 인증서의 Base64 인코딩 값입니다. 신뢰할 수 있는 CA를 사용하는 경우 이 매개 변수를 제공해야 합니다. 신뢰할 수 있는 CA가 사용되지 않으면 이 작업을 무시할 수 있습니다.

일반적인 워크플로에는 다음 단계가 포함됩니다.

단계

1. 클라이언트 인증서 및 키를 생성합니다. 생성 시 CN(일반 이름)을 ONTAP 사용자로 설정하여 인증하십시오.

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=vsadmin"
```

2. 신뢰할 수 있는 CA 인증서를 ONTAP 클러스터에 추가합니다. 이는 스토리지 관리자가 이미 처리한 것일 수 있습니다. 트러스트된 CA가 사용되지 않으면 무시합니다.

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. ONTAP 클러스터에 클라이언트 인증서 및 키(1단계)를 설치합니다.

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

4. ONTAP 보안 로그인 역할이 지원되는지 확인합니다 cert 인증 방법.

```
security login create -user-or-group-name vsadmin -application ontapi
-authentication-method cert -vserver <vserver-name>
security login create -user-or-group-name vsadmin -application http
-authentication-method cert -vserver <vserver-name>
```

5. 생성된 인증서를 사용하여 인증을 테스트합니다. ONTAP 관리 LIF> 및 <SVM 이름>을 관리 LIF IP 및 SVM 이름으로 바꿉니다. LIF의 서비스 정책이 으로 설정되어 있는지 확인해야 합니다 default-data-management.

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. Base64로 인증서, 키 및 신뢰할 수 있는 CA 인증서를 인코딩합니다.

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 이전 단계에서 얻은 값을 사용하여 백엔드를 생성합니다.

```
cat cert-backend-updated.json
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "NasBackend",
  "managementLIF": "1.2.3.4",
  "dataLIF": "1.2.3.8",
  "svm": "vserver_test",
  "clientCertificate": "Faaaakkkkeeee...Vaaalllluuueeee",
  "clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
  "storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+-----+-----+-----+
+-----+-----+
|   NAME   | STORAGE DRIVER |           UUID           |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |      9 |
+-----+-----+-----+
+-----+-----+
```

인증 방법을 업데이트하거나 자격 증명을 회전합니다

다른 인증 방법을 사용하거나 자격 증명을 회전하도록 기존 백엔드를 업데이트할 수 있습니다. 이렇게 하면 사용자 이름/암호를 사용하는 백엔드를 인증서를 사용하도록 업데이트할 수 있고 인증서를 사용하는 백엔드는 사용자 이름/암호 기반으로 업데이트할 수 있습니다. 이렇게 하려면 기존 인증 방법을 제거하고 새 인증 방법을 추가해야 합니다. 그런 다음 실행할 필수 매개 변수가 포함된 업데이트된 backend.json 파일을 사용합니다 tridentctl update backend.

```

cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+-----+-----+-----+
+-----+-----+
|   NAME   | STORAGE DRIVER |           UUID           |
STATE | VOLUMES |
+-----+-----+-----+
+-----+-----+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |      9 |
+-----+-----+-----+
+-----+-----+

```



암호를 회전할 때 스토리지 관리자는 먼저 ONTAP에서 사용자의 암호를 업데이트해야 합니다. 그 다음에는 백엔드 업데이트가 있습니다. 인증서를 회전할 때 여러 인증서를 사용자에게 추가할 수 있습니다. 그런 다음 백엔드가 업데이트되어 새 인증서를 사용합니다. 그러면 ONTAP 클러스터에서 이전 인증서를 삭제할 수 있습니다.

백엔드를 업데이트해도 이미 생성된 볼륨에 대한 액세스가 중단되거나 이후에 생성된 볼륨 연결에 영향을 미치지 않습니다. 백엔드 업데이트가 성공적이면 Astra Trident가 ONTAP 백엔드와 통신하고 향후 볼륨 작업을 처리할 수 있음을 나타냅니다.

NFS 익스포트 정책을 관리합니다

Astra Trident는 NFS 익스포트 정책을 사용하여 프로비저닝하는 볼륨에 대한 액세스를 제어합니다.

Astra Trident는 익스포트 정책을 사용할 때 다음 두 가지 옵션을 제공합니다.

- Astra Trident는 익스포트 정책 자체를 동적으로 관리할 수 있습니다. 이 운영 모드에서 스토리지 관리자는 허용할 수 있는 IP 주소를 나타내는 CIDR 블록 목록을 지정합니다. Astra Trident는 이러한 범위에 속하는 노드 IP를 익스포트 정책에 자동으로 추가합니다. 또는 CIDR을 지정하지 않으면 노드에서 발견된 글로벌 범위의 유니캐스트 IP가 내보내기 정책에 추가됩니다.

- 스토리지 관리자는 익스포트 정책을 생성하고 규칙을 수동으로 추가할 수 있습니다. Astra Trident는 구성에 다른 익스포트 정책 이름을 지정하지 않는 한 기본 익스포트 정책을 사용합니다.

익스포트 정책을 동적으로 관리

CSI Trident의 20.04 릴리스는 ONTAP 백엔드에 대한 익스포트 정책을 동적으로 관리할 수 있는 기능을 제공합니다. 따라서 스토리지 관리자는 명시적 규칙을 수동으로 정의하는 대신 작업자 노드 IP에 허용되는 주소 공간을 지정할 수 있습니다. 익스포트 정책 관리를 크게 간소화하므로, 익스포트 정책을 수정하면 더 이상 스토리지 클러스터에 대한 수동 작업이 필요하지 않습니다. 또한 스토리지 클러스터에 대한 액세스를 지정된 범위의 IP가 있는 작업자 노드에만 제한함으로써 세분화된 자동 관리를 지원합니다.



CSI Trident에만 내보내기 정책의 동적 관리를 사용할 수 있습니다. 작업자 노드가 NATED가 아닌지 확인하는 것이 중요합니다.

예

두 가지 구성 옵션을 사용해야 합니다. 다음은 백엔드 정의의 예입니다.

```
---
version: 1
storageDriverName: ontap-nas
backendName: ontap_nas_auto_export
managementLIF: 192.168.0.135
svm: svm1
username: vsadmin
password: password
autoExportCIDRs:
- 192.168.0.0/24
autoExportPolicy: true
```



이 기능을 사용할 때는 SVM의 루트 교차점에 노드 CIDR 블록(예: 기본 익스포트 정책)을 허용하는 익스포트 규칙과 함께 이전에 생성된 익스포트 정책이 있는지 확인해야 합니다. Astra Trident를 위한 SVM 전용 NetApp의 권장 모범 사례를 항상 따르십시오.

다음은 위의 예를 사용하여 이 기능이 작동하는 방식에 대한 설명입니다.

- `autoExportPolicy` 가 로 설정되어 있습니다 `true`. 이는 Astra Trident가 에 대한 익스포트 정책을 생성한다는 것을 나타냅니다 `svm1` 를 사용하여 규칙 추가 및 삭제를 처리합니다 `autoExportCIDRs` 주소 블록. 예를 들어 UUID 403b5326-8482-40db-96d0-d83fb3f4daec 및 가 있는 백엔드를 사용할 수 있습니다 `autoExportPolicy` 를 로 설정합니다 `true` 이라는 익스포트 정책을 생성합니다 `trident-403b5326-8482-40db-96d0-d83fb3f4daec` SVM에서.
- `autoExportCIDRs` 주소 블록 목록이 포함되어 있습니다. 이 필드는 선택 사항이며 기본적으로 `["0.0.0.0/0", ":/0"]`입니다. 정의되지 않은 경우 Astra Trident는 작업자 노드에 있는 모든 전역 범위의 유니캐스트 주소를 추가합니다.

이 예에서 는 입니다 `192.168.0.0/24` 주소 공간이 제공됩니다. 이 주소 범위에 속하는 Kubernetes 노드 IP가 Astra Trident가 생성하는 익스포트 정책에 추가됨을 나타냅니다. Astra Trident가 실행되는 노드를 등록하면 노드의 IP 주소를 검색하여 에 제공된 주소 블록과 대조하여 확인합니다 `autoExportCIDRs`. IP를 필터링한 후 Astra Trident는

검색된 클라이언트 IP에 대한 익스포트 정책 규칙을 생성하며, 식별하는 각 노드에 대해 하나의 규칙을 사용합니다.

업데이트할 수 있습니다 autoExportPolicy 및 autoExportCIDRs 백엔드는 만든 후에 사용합니다. 기존 CIDR을 자동으로 관리하거나 삭제하는 백엔드에 새 CIDR을 추가할 수 있습니다. CIDR을 삭제할 때는 기존 연결이 끊어지지 않도록 주의해야 합니다. 를 비활성화하도록 선택할 수도 있습니다 autoExportPolicy 백엔드의 경우 수동으로 생성된 내보내기 정책으로 돌아갑니다. 이렇게 하려면 을 설정해야 합니다 exportPolicy 백엔드 구성의 매개 변수입니다.

Astra Trident가 백엔드를 생성하거나 업데이트한 후 을 사용하여 백엔드를 확인할 수 있습니다 tridentctl 또는 해당 tridentbackend CRD:

```
./tridentctl get backends ontap_nas_auto_export -n trident -o yaml
items:
- backendUUID: 403b5326-8482-40db-96d0-d83fb3f4daec
  config:
    aggregate: ""
    autoExportCIDRs:
    - 192.168.0.0/24
    autoExportPolicy: true
    backendName: ontap_nas_auto_export
    chapInitiatorSecret: ""
    chapTargetInitiatorSecret: ""
    chapTargetUsername: ""
    chapUsername: ""
    dataLIF: 192.168.0.135
    debug: false
    debugTraceFlags: null
    defaults:
      encryption: "false"
      exportPolicy: <automatic>
      fileType: ext4
```

노드가 Kubernetes 클러스터에 추가되고 Astra Trident 컨트롤러에 등록되면 기존 백엔드의 내보내기 정책이 업데이트됩니다(에 지정된 주소 범위에 속하면 됨) autoExportCIDRs 백 엔드용).

노드가 제거되면 Astra Trident는 온라인 상태인 모든 백엔드를 검사하여 노드에 대한 액세스 규칙을 제거합니다. Astra Trident는 관리되는 백엔드의 내보내기 정책에서 이 노드 IP를 제거하여 불량 마운트를 방지합니다. 단, 클러스터의 새 노드에서 이 IP를 다시 사용하지 않는 한 마찬가지입니다.

기존 백엔드의 경우 백엔드를 로 업데이트합니다 tridentctl update backend Astra Trident가 익스포트 정책을 자동으로 관리하는지 확인합니다. 그러면 백엔드의 UUID 뒤에 이름이 지정된 새 내보내기 정책이 생성되고 백엔드에 있는 볼륨은 새로 생성된 내보내기 정책을 다시 마운트할 때 사용합니다.



자동 관리되는 내보내기 정책이 있는 백엔드를 삭제하면 동적으로 생성된 내보내기 정책이 삭제됩니다. 백엔드가 다시 생성되면 백엔드가 새 백엔드로 처리되어 새 익스포트 정책이 생성됩니다.

라이브 노드의 IP 주소가 업데이트되면 노드에서 Astra Trident POD를 다시 시작해야 합니다. 그런 다음 Astra Trident가 이 IP 변경 사항을 반영하도록 관리하는 백엔드에 대한 익스포트 정책을 업데이트합니다.

ONTAP NAS 구성 옵션 및 예

Astra Trident 설치를 통해 ONTAP NAS 드라이버를 생성하고 사용하는 방법에 대해 알아보십시오. 이 섹션에서는 백엔드 구성 예제 및 백엔드를 StorageClasses에 매핑하는 방법에 대한 세부 정보를 제공합니다.

백엔드 구성 옵션

백엔드 구성 옵션은 다음 표를 참조하십시오.

매개 변수	설명	기본값
version		항상 1
storageDriverName	스토리지 드라이버의 이름입니다	"ONTAP-NAS", "ONTAP-NAS-이코노미", "ONTAP-NAS-Flexgroup", "ONTAP-SAN", "ONTAP-SAN-이코노미"
backendName	사용자 지정 이름 또는 스토리지 백엔드	드라이버 이름 + "_" + dataLIF
managementLIF	클러스터 또는 SVM 관리 LIF의 IP 주소 MetroCluster의 원활한 전환을 위해 SVM 관리 LIF를 지정해야 합니다. FQDN(정규화된 도메인 이름)을 지정할 수 있습니다. 을 사용하여 Astra Trident가 설치된 경우 IPv6 주소를 사용하도록 설정할 수 있습니다 --use-ipv6 깃발. IPv6 주소는 대괄호로 묶어야 합니다(예: [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]).	"10.0.0.1", "[2001:1234:ABCD::fee]"
dataLIF	프로토콜 LIF의 IP 주소입니다. 지정할 것을 권장합니다 dataLIF. 제공되지 않는 경우 Astra Trident는 SVM에서 데이터 LIF를 가져옵니다. NFS 마운트 작업에 사용할 FQDN(정규화된 도메인 이름)을 지정하면 여러 데이터 LIF에서 로드 밸런싱을 위해 라운드 로빈 DNS를 생성할 수 있습니다. 초기 설정 후에 변경할 수 있습니다. 을 참조하십시오 . 을 사용하여 Astra Trident가 설치된 경우 IPv6 주소를 사용하도록 설정할 수 있습니다 --use-ipv6 깃발. IPv6 주소는 대괄호로 묶어야 합니다(예: [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]).	지정되지 않은 경우 SVM에서 지정 주소 또는 파생(권장하지 않음)
autoExportPolicy	자동 익스포트 정책 생성 및 업데이트 [Boolean] 활성화 를 사용합니다 autoExportPolicy 및 autoExportCIDRs 옵션, Astra Trident는 익스포트 정책을 자동으로 관리할 수 있습니다.	거짓

매개 변수	설명	기본값
autoExportCIDRs	Kubernetes 노드 IP를 언제 기준으로 필터링하는 CIDR 목록입니다 autoExportPolicy 가 활성화됩니다. 를 사용합니다 autoExportPolicy 및 autoExportCIDRs 옵션, Astra Trident는 익스포트 정책을 자동으로 관리할 수 있습니다.	["0.0.0.0/0", ":::/0"]
labels	볼륨에 적용할 임의의 JSON 형식 레이블 세트입니다	""
clientCertificate	Base64 - 클라이언트 인증서의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
clientPrivateKey	Base64 - 클라이언트 개인 키의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
trustedCACertificate	Base64 - 신뢰할 수 있는 CA 인증서의 인코딩된 값입니다. 선택 사항. 인증서 기반 인증에 사용됩니다	""
username	클러스터/SVM에 연결할 사용자 이름입니다. 자격 증명 기반 인증에 사용됩니다	
password	클러스터/SVM에 연결하는 암호 자격 증명 기반 인증에 사용됩니다	
svm	사용할 스토리지 가상 머신입니다	SVM에서 파생된 경우 managementLIF 이(가) 지정되었습니다
storagePrefix	SVM에서 새 볼륨을 프로비저닝할 때 사용되는 접두사 설정한 후에는 업데이트할 수 없습니다	"삼중류"
limitAggregateUsage	사용량이 이 비율을 초과하면 프로비저닝이 실패합니다. ONTAP * 용 아마존 FSx에는 * 가 적용되지 않습니다	""(기본적으로 적용되지 않음)
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다.	""(기본적으로 적용되지 않음)
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다. 또한 qtree 및 LUN에 대해 관리하는 볼륨의 최대 크기 및 을 제한합니다 qtreesPerFlexvol 옵션을 사용하면 FlexVol당 최대 qtree 수를 사용자 지정할 수 있습니다.	""(기본적으로 적용되지 않음)
lunsPerFlexvol	FlexVol당 최대 LUN 수는 범위[50, 200]에 있어야 합니다.	"100"

매개 변수	설명	기본값
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: {"api":false, "method":true} 사용하지 마십시오 debugTraceFlags 문제 해결 및 자세한 로그 덤프가 필요한 경우를 제외하고	null입니다
nfsMountOptions	심표로 구분된 NFS 마운트 옵션 목록입니다. Kubernetes 영구 볼륨의 마운트 옵션은 일반적으로 스토리지 클래스에서 지정되지만, 스토리지 클래스에 마운트 옵션을 지정하지 않으면 Astra Trident가 스토리지 백엔드의 구성 파일에 지정된 마운트 옵션을 사용하여 로 돌아갑니다. 스토리지 클래스 또는 구성 파일에 마운트 옵션이 지정되지 않은 경우 Astra Trident는 연결된 영구 볼륨에 마운트 옵션을 설정하지 않습니다.	""
qtreesPerFlexvol	FlexVol당 최대 qtree, 범위 [50, 300]에 있어야 함	"200"
useREST	ONTAP REST API를 사용하는 부울 매개 변수입니다. * 기술 미리 보기 * useREST 프로덕션 작업 부하가 아닌 테스트 환경에 권장되는 기술 미리 보기로 제공됩니다. 를 로 설정한 경우 true, Astra Trident는 ONTAP REST API를 사용하여 백엔드와 통신합니다. 이 기능을 사용하려면 ONTAP 9.11.1 이상이 필요합니다. 또한 사용되는 ONTAP 로그인 역할에 대한 액세스 권한이 있어야 합니다 ontap 응용 프로그램. 이는 사전 정의된 에 의해 충족됩니다 vsadmin 및 cluster-admin 역할. useREST 는 MetroCluster에서 지원되지 않습니다.	거짓

볼륨 프로비저닝을 위한 백엔드 구성 옵션

에서 이러한 옵션을 사용하여 기본 프로비저닝을 제어할 수 있습니다 defaults 섹션을 참조하십시오. 예를 들어, 아래 구성 예제를 참조하십시오.

매개 변수	설명	기본값
spaceAllocation	LUN에 대한 공간 할당	"참"
spaceReserve	공간 예약 모드, "없음"(씬) 또는 "볼륨"(일반)	"없음"
snapshotPolicy	사용할 스냅샷 정책입니다	"없음"

매개 변수	설명	기본값
qosPolicy	생성된 볼륨에 할당할 QoS 정책 그룹입니다. 스토리지 풀/백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다	""
adaptiveQosPolicy	생성된 볼륨에 할당할 적응형 QoS 정책 그룹입니다. 스토리지 풀/백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다. ONTAP에서 지원되지 않음 - NAS - 이코노미	""
snapshotReserve	스냅샷 "0"에 예약된 볼륨의 백분율	If(경우 snapshotPolicy "없음"이고, 그렇지 않으면 ""입니다.
splitOnClone	생성 시 상위 클론에서 클론을 분할합니다	"거짓"
encryption	새 볼륨에 NVE(NetApp Volume Encryption)를 활성화합니다. 기본값은 입니다 false. 이 옵션을 사용하려면 NVE 라이선스가 클러스터에서 활성화되어 있어야 합니다. 백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다. 자세한 내용은 다음을 참조하십시오. "Astra Trident가 NVE 및 NAE와 연동되는 방식" .	"거짓"
tieringPolicy	"없음"을 사용하는 계층화 정책	ONTAP 9.5 이전 SVM-DR 구성의 경우 "스냅샷 전용
unixPermissions	모드를 선택합니다	NFS 볼륨의 경우 "777", SMB 볼륨의 경우 비어 있음(해당 없음
snapshotDir	의 표시 여부를 제어합니다 .snapshot 디렉토리	"거짓"
exportPolicy	사용할 익스포트 정책	"기본값"
securityStyle	새로운 볼륨에 대한 보안 스타일 NFS를 지원합니다 mixed 및 unix 보안 스타일. SMB 지원 mixed 및 ntfs 보안 스타일.	NFS 기본값은 입니다 unix. SMB 기본값은 입니다 ntfs.



Astra Trident와 함께 QoS 정책 그룹을 사용하려면 ONTAP 9.8 이상이 필요합니다. 비공유 QoS 정책 그룹을 사용하고 정책 그룹이 각 구성요소별로 적용되었는지 확인하는 것이 좋습니다. 공유 QoS 정책 그룹은 모든 워크로드의 총 처리량에 대해 상한을 적용합니다.

볼륨 프로비저닝의 예

다음은 기본값이 정의된 예입니다.

```

---
version: 1
storageDriverName: ontap-nas
backendName: customBackendName
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
labels:
  k8scluster: dev1
  backend: dev1-nasbackend
svm: trident_svm
username: cluster-admin
password: password
limitAggregateUsage: 80%
limitVolumeSize: 50Gi
nfsMountOptions: nfsvers=4
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: premium
  exportPolicy: myk8scluster
  snapshotPolicy: default
  snapshotReserve: '10'

```

용 ontap-nas 및 ontap-nas-flexgroups`이제 Astra Trident가 새로운 계산을 사용하여 snapshotReserve Percentage 및 PVC로 FlexVol의 크기를 올바르게 지정합니다. 사용자가 PVC를 요청하면 Astra Trident는 새 계산을 사용하여 더 많은 공간을 가진 원본 FlexVol를 생성합니다. 이 계산을 통해 사용자는 PVC에서 요청한 쓰기 가능 공간을 확보할 수 있으며 요청된 공간보다 적은 공간을 확보할 수 있습니다. v21.07 이전에는 사용자가 스냅샷 보존 공간을 50%로 하여 PVC(예: 5GiB)를 요청할 때 쓰기 가능한 공간은 2.5GiB에 불과합니다. 사용자가 요청한 것은 전체 볼륨과 이기 때문입니다 `snapshotReserve` 이 백분율에 포함됩니다. Trident 21.07을 사용하면 사용자가 요청하는 것이 쓰기 가능한 공간이고 Astra Trident가 이를 정의합니다 snapshotReserve 전체 볼륨의 백분율로 표시됩니다. 예는 적용되지 않습니다 ontap-nas-economy. 이 작동 방식을 보려면 다음 예를 참조하십시오.

계산은 다음과 같습니다.

```

Total volume size = (PVC requested size) / (1 - (snapshotReserve
percentage) / 100)

```

snapshotReserve = 50%, PVC request = 5GiB의 경우, 총 볼륨 크기는 $2/5 = 10\text{GiB}$ 이고 사용 가능한 크기는 5GiB입니다. 이는 사용자가 PVC 요청에서 요청한 것입니다. 를 클릭합니다 volume show 명령은 이 예제와 유사한 결과를 표시해야 합니다.

Vserver	Volume	Aggregate	State	Type	Size	Available	Used%
		_pvc_89f1c156_3801_4de4_9f9d_034d54c395f4	online	RW	10GB	5.00GB	0%
		_pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba	online	RW	1GB	511.8MB	0%

2 entries were displayed.

이전 설치에서 기존 백엔드는 Astra Trident를 업그레이드할 때 위에서 설명한 대로 볼륨을 프로비저닝합니다. 업그레이드하기 전에 생성한 볼륨의 경우 변경 사항을 관찰하기 위해 볼륨의 크기를 조정해야 합니다. 예를 들어, 2GiB PVC가 인 경우 `snapshotReserve=50` 그 결과, 쓰기 가능한 공간 1GiB를 제공하는 볼륨이 탄생했습니다. 예를 들어, 볼륨을 3GiB로 조정하면 애플리케이션에 6GiB 볼륨의 쓰기 가능 공간이 3GiB로 표시됩니다.

예

최소 구성의 예

다음 예에서는 대부분의 매개 변수를 기본값으로 두는 기본 구성을 보여 줍니다. 이는 백엔드를 정의하는 가장 쉬운 방법입니다.



Trident가 있는 NetApp ONTAP에서 Amazon FSx를 사용하는 경우 IP 주소 대신 LIF에 대한 DNS 이름을 지정하는 것이 좋습니다.

의 기본 옵션 `ontap-nas-economy`

```

---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password

```

인증서 기반 인증입니다

이는 최소 백엔드 구성의 예입니다. `clientCertificate`, `clientPrivateKey`, 및 `trustedCACertificate` (신뢰할 수 있는 CA를 사용하는 경우 선택 사항)는 에 채워집니다 `backend.json` 그리고 각각 클라이언트 인증서, 개인 키 및 신뢰할 수 있는 CA 인증서의 base64로 인코딩된 값을 사용합니다.

```
---
version: 1
backendName: DefaultNASBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```


다음 예에서는 Astra Trident가 동적 익스포트 정책을 사용하여 익스포트 정책을 자동으로 생성하고 관리하도록 지시하는 방법을 보여 줍니다. 이 기능은 예 대해서도 동일하게 작동합니다 `ontap-nas-economy` 및 `ontap-nas-flexgroup` 드라이버.

ONTAP - NAS 드라이버

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-nasbackend
autoExportPolicy: true
autoExportCIDRs:
- 10.0.0.0/24
username: admin
password: password
nfsMountOptions: nfsvers=4
```

`ontap-nas-flexgroup` 드라이버

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
labels:
  k8scluster: test-cluster-east-1b
  backend: test1-ontap-cluster
svm: svm_nfs
username: vsadmin
password: password
```

IPv6 주소 사용

이 예에서는 를 보여 줍니다 managementLIF IPv6 주소 사용.

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

ontap-nas-economy 드라이버

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

ontap-nas **SMB** 볼륨을 사용하는 **ONTAP용 Amazon FSx** 드라이버입니다

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

가상 풀의 백엔드 예

아래 표시된 샘플 백엔드 정의 파일에서와 같은 모든 스토리지 풀에 대한 특정 기본값이 설정됩니다 `spaceReserve` 없음, `spaceAllocation` 거짓일 경우, 및 `encryption` 거짓일 때. 가상 풀은 스토리지 섹션에 정의됩니다.

Astra Trident는 "Comments" 필드에 제공 레이블을 설정합니다. 설명은 FlexVol for에서 설정됩니다 `ontap-nas` 또는 `FlexGroup for ontap-nas-flexgroup`. Astra Trident는 프로비저닝할 때 가상 풀에 있는 모든 레이블을 스토리지 볼륨에 복사합니다. 편의를 위해 스토리지 관리자는 가상 풀 및 그룹 볼륨별로 레이블을 레이블별로 정의할 수 있습니다.

이 예에서는 일부 스토리지 풀이 자체적으로 설정됩니다 `spaceReserve`, `spaceAllocation`, 및 `encryption` 값 및 일부 풀이 위에 설정된 기본값을 덮어씁니다.

<code>ontap-nas</code> 드라이버

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: admin
password: password
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: 'false'
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
- labels:
  app: msoffice
  cost: '100'
  zone: us_east_1a
  defaults:
    spaceReserve: volume
    encryption: 'true'
    unixPermissions: '0755'
    adaptiveQosPolicy: adaptive-premium
- labels:
  app: slack
  cost: '75'
  zone: us_east_1b
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0755'
- labels:
  app: wordpress
  cost: '50'
  zone: us_east_1c
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0775'
```

```
- labels:  
  app: mysqldb  
  cost: '25'  
  zone: us_east_1d  
  defaults:  
    spaceReserve: volume  
    encryption: 'false'  
    unixPermissions: '0775'
```

<code>ontap-nas-flexgroup</code> 드라이버

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
defaults:
  spaceReserve: none
  encryption: 'false'
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
- labels:
  protection: gold
  creditpoints: '50000'
  zone: us_east_1a
  defaults:
    spaceReserve: volume
    encryption: 'true'
    unixPermissions: '0755'
- labels:
  protection: gold
  creditpoints: '30000'
  zone: us_east_1b
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0755'
- labels:
  protection: silver
  creditpoints: '20000'
  zone: us_east_1c
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0775'
- labels:
  protection: bronze
  creditpoints: '10000'
```

```
zone: us_east_1d
defaults:
  spaceReserve: volume
  encryption: 'false'
  unixPermissions: '0775'
```

<code>ontap-nas-economy</code> 드라이버

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
defaults:
  spaceReserve: none
  encryption: 'false'
labels:
  store: nas_economy_store
region: us_east_1
storage:
- labels:
  department: finance
  creditpoints: '6000'
  zone: us_east_1a
  defaults:
    spaceReserve: volume
    encryption: 'true'
    unixPermissions: '0755'
- labels:
  department: legal
  creditpoints: '5000'
  zone: us_east_1b
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0755'
- labels:
  department: engineering
  creditpoints: '3000'
  zone: us_east_1c
  defaults:
    spaceReserve: none
    encryption: 'true'
    unixPermissions: '0775'
- labels:
  department: humanresource
  creditpoints: '2000'
  zone: us_east_1d
```



```
defaults:
  spaceReserve: volume
  encryption: 'false'
  unixPermissions: '0775'
```

업데이트 dataLIF 초기 구성 후

다음 명령을 실행하여 초기 구성 후에 데이터 LIF를 변경할 수 있으며, 업데이트된 데이터 LIF가 포함된 새 백엔드 JSON 파일을 제공할 수 있습니다.

```
tridentctl update backend <backend-name> -f <path-to-backend-json-file-
with-updated-dataLIF>
```



PVC가 하나 이상의 포드에 연결된 경우 해당 포드를 모두 내린 다음 다시 불러와서 새 데이터 LIF가 적용되도록 해야 합니다.

백엔드를 **StorageClasses**에 매핑합니다

다음 StorageClass 정의는 위의 가상 풀을 참조합니다. 를 사용합니다 parameters.selector 필드에서 각 StorageClass는 볼륨을 호스팅하는 데 사용할 수 있는 가상 풀을 호출합니다. 선택한 가상 풀에 볼륨이 정의되어 있습니다.

- 첫 번째 StorageClass입니다 (protection-gold)는 의 첫 번째, 두 번째 가상 풀에 매핑됩니다 ontap-nas-flexgroup 백엔드 및 의 첫 번째 가상 풀 ontap-san 백엔드. 골드 레벨 보호 기능을 제공하는 유일한 풀입니다.
- 두 번째 StorageClass입니다 (protection-not-gold)는 에서 세 번째, 네 번째 가상 풀에 매핑됩니다 ontap-nas-flexgroup 백엔드 및 에서 세 번째 가상 풀입니다 ontap-san 백엔드. 금 이외의 보호 수준을 제공하는 유일한 풀입니다.
- 세 번째 StorageClass입니다 (app-mysqldb)는 에서 네 번째 가상 풀에 매핑됩니다 ontap-nas 에 있는 백엔드 및 세 번째 가상 풀입니다 ontap-san-economy 백엔드. mysqldb 유형 앱에 대한 스토리지 풀 구성을 제공하는 유일한 풀입니다.
- 네 번째 StorageClass입니다 (protection-silver-creditpoints-20k)는 의 세 번째 가상 풀에 매핑됩니다 ontap-nas-flexgroup 의 백엔드 및 두 번째 가상 풀입니다 ontap-san 백엔드. 20000 크레딧 포인트에서 골드 레벨 보호 기능을 제공하는 유일한 풀입니다.
- 다섯 번째 StorageClass입니다 (creditpoints-5k)는 의 두 번째 가상 풀에 매핑됩니다 ontap-nas-economy 에 있는 백엔드 및 세 번째 가상 풀입니다 ontap-san 백엔드. 5000 크레딧 포인트에 있는 유일한 풀 서비스입니다.

Astra Trident가 선택한 가상 풀을 결정하고 스토리지 요구 사항을 충족시킵니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: netapp.io/trident
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

NetApp ONTAP용 Amazon FSx

NetApp ONTAP용 Amazon FSx와 함께 Astra Trident를 사용하십시오

"NetApp ONTAP용 Amazon FSx" NetApp ONTAP 스토리지 운영 체제가 제공하는 파일 시스템을 실행하고 실행할 수 있도록 완벽하게 관리되는 AWS 서비스입니다. ONTAP용 FSx를 사용하면 익숙한 NetApp 기능, 성능 및 관리 기능을 활용하는 동시에, AWS에 데이터를 저장하는 데 따른 단순성, 민첩성, 보안, 확장성을 활용할 수 있습니다. ONTAP용 FSx는 ONTAP 파일 시스템 기능 및 관리 API를 지원합니다.

파일 시스템은 Amazon FSx의 주요 리소스이며, 이는 사내 ONTAP 클러스터와 유사합니다. 각 SVM 내에서 파일 시스템에 파일과 폴더를 저장하는 데이터 컨테이너인 하나 이상의 볼륨을 생성할 수 있습니다. NetApp ONTAP용 Amazon FSx를 사용하면 클라우드에서 Data ONTAP가 관리형 파일 시스템으로 제공됩니다. 새로운 파일 시스템 유형을 * NetApp ONTAP * 라고 합니다.

NetApp ONTAP용 Amazon FSx와 Astra Trident를 사용하면 Amazon EKS(Elastic Kubernetes Service)에서 실행되는 Kubernetes 클러스터가 ONTAP에서 지원하는 블록 및 파일 영구 볼륨을 프로비저닝할 수 있습니다.

NetApp ONTAP용 Amazon FSx에서 사용됩니다. "FabricPool" 스토리지 계층을 관리합니다. 이 기능을 사용하면 데이터의 액세스 빈도 여부에 따라 데이터를 계층에 저장할 수 있습니다.

고려 사항

- SMB 볼륨:
 - SMB 볼륨은 를 사용하여 지원됩니다 `ontap-nas` 드라이버만 해당.
 - Astra Trident는 Windows 노드에서 실행되는 Pod에만 마운트된 SMB 볼륨을 지원합니다.
 - Astra Trident는 Windows ARM 아키텍처를 지원하지 않습니다.
- 자동 백업이 설정된 Amazon FSx 파일 시스템에서 생성된 볼륨은 Trident에서 삭제할 수 없습니다. PVC를 삭제하려면 ONTAP 체적에 대한 PV 및 FSx를 수동으로 삭제해야 합니다. 이 문제를 방지하려면:
 - **Quick create**를 사용하여 ONTAP 파일 시스템용 FSx를 생성하지 마십시오. 빠른 생성 워크플로에서는 자동 백업을 사용할 수 있으며 수신 거부 옵션은 제공하지 않습니다.
 - 표준 생성 을 사용하는 경우 자동 백업을 비활성화합니다. 자동 백업을 비활성화하면 Trident는 추가적인 수동 개입 없이 볼륨을 성공적으로 삭제할 수 있습니다.

▼ **Backup and maintenance - optional**

Daily automatic backup [Info](#)
Amazon FSx can protect your data through daily backups

Enabled

Disabled

드라이버

다음 드라이버를 사용하여 Astra Trident를 NetApp ONTAP용 Amazon FSx와 통합할 수 있습니다.

- `ontap-san`: 각 PV 프로비저닝은 고유한 Amazon FSx for NetApp ONTAP 볼륨 내에 있는 LUN입니다.
- `ontap-san-economy`: 각 PV 프로비저닝은 NetApp ONTAP 볼륨에 대해 Amazon FSx당 구성 가능한 LUN 수를 가진 LUN입니다.
- `ontap-nas`: 각 PV 프로비저닝은 NetApp ONTAP 볼륨에 대한 전체 Amazon FSx입니다.
- `ontap-nas-economy`: 각 PV 프로비저닝은 qtree이며, NetApp ONTAP 볼륨용 Amazon FSx당 구성 가능한 Qtree 수가 있습니다.
- `ontap-nas-flexgroup`: 각 PV 프로비저닝은 NetApp ONTAP FlexGroup 볼륨에 대한 전체 Amazon FSx입니다.

드라이버 세부 정보는 를 참조하십시오 ["ONTAP 드라이버"](#).

인증

Astra Trident는 두 가지 인증 모드를 제공합니다.

- 인증서 기반: Astra Trident는 SVM에 설치된 인증서를 사용하여 FSx 파일 시스템의 SVM과 통신합니다.
- 자격 증명 기반: 을 사용할 수 있습니다 `fsxadmin` 파일 시스템 또는 의 사용자입니다 `vsadmin` SVM을 위해 사용자가 구성됨



Astra Trident는 한 대로 실행될 것으로 예상합니다 `vsadmin` SVM 사용자 또는 동일한 역할을 가진 다른 이름의 사용자 NetApp ONTAP용 Amazon FSx에는 가 있습니다 `fsxadmin` ONTAP의 제한된 교체인 사용자입니다 `admin` 클러스터 사용자. 을 사용하는 것이 좋습니다 `vsadmin` Astra Trident와 함께.

자격 증명 기반 방법과 인증서 기반 방법 간에 이동하도록 백엔드를 업데이트할 수 있습니다. 그러나 * 자격 증명 및 인증서 * 를 제공하려고 하면 백엔드 생성이 실패합니다. 다른 인증 방법으로 전환하려면 백엔드 구성에서 기존 방법을 제거해야 합니다.

인증 활성화에 대한 자세한 내용은 드라이버 유형에 대한 인증을 참조하십시오.

- ["ONTAP NAS 인증"](#)
- ["ONTAP SAN 인증"](#)

자세한 내용을 확인하십시오

- ["NetApp ONTAP용 Amazon FSx 문서"](#)
- ["NetApp ONTAP용 Amazon FSx 블로그 게시물"](#)

NetApp ONTAP용 Amazon FSx를 통합합니다

Amazon Elastic Kubernetes Service(EKS)에서 실행되는 Kubernetes 클러스터가 ONTAP에서 지원하는 블록 및 파일 영구 볼륨을 프로비저닝할 수 있도록 NetApp ONTAP 파일 시스템용 Amazon FSx를 Astra Trident와 통합할 수 있습니다.

시작하기 전에

또한 "[Astra Trident 요구사항](#)", ONTAP용 FSx와 Astra Trident를 통합하려면 다음이 필요합니다.

- 기존 Amazon EKS 클러스터 또는 자체 관리형 Kubernetes 클러스터 `kubect1` 설치되어 있습니다.
- 클러스터의 작업자 노드에서 연결할 수 있는 NetApp ONTAP 파일 시스템용 기존 Amazon FSx 및 SVM(스토리지 가상 시스템)입니다.
- 에 대해 준비된 작업자 노드입니다 "[NFS 또는 iSCSI](#)".



Amazon Linux 및 Ubuntu에 필요한 노드 준비 단계를 따라야 합니다 "[Amazon Machine Images\(아마존 머신 이미지\)](#)" (AMI) EKS AMI 유형에 따라 다릅니다.

SMB 볼륨에 대한 추가 요구사항

- Linux 컨트롤러 노드 및 Windows Server 2019를 실행하는 Windows 작업자 노드가 있는 Kubernetes 클러스터 Astra Trident는 Windows 노드에서 실행되는 Pod에만 마운트된 SMB 볼륨을 지원합니다.
- Active Directory 자격 증명이 포함된 Astra Trident 암호가 하나 이상 있어야 합니다. 암호를 생성합니다 `smbcreds`:

```
kubect1 create secret generic smbcreds --from-literal username=user  
--from-literal password='password'
```

- Windows 서비스로 구성된 CSI 프록시. 를 구성합니다 `csi-proxy`를 참조하십시오 "[GitHub:CSI 프록시](#)" 또는 "[GitHub: Windows용 CSI 프록시](#)" Windows에서 실행되는 Kubernetes 노드의 경우:

ONTAP SAN 및 NAS 드라이버 통합



SMB 볼륨에 대해 를 구성하는 경우 을 읽어야 합니다 [SMB 볼륨 프로비저닝을 위한 준비 백엔드를 생성하기 전에](#)

단계

1. 중 하나를 사용하여 Astra Trident를 배포합니다 "[배포 방법](#)".
2. SVM 관리 LIF DNS 이름을 수집합니다. 예를 들어, AWS CLI를 사용하여 를 찾습니다 `DNSName` 에 입력 `Endpoints` → `Management` 다음 명령을 실행한 후:

```
aws fsx describe-storage-virtual-machines --region <file system region>
```

3. 에 대한 인증서를 만들고 설치합니다 "[NAS 백엔드 인증](#)" 또는 "[SAN 백엔드 인증](#)".



파일 시스템에 연결할 수 있는 모든 위치에서 SSH를 사용하여 파일 시스템(예: 인증서 설치)에 로그인할 수 있습니다. 를 사용합니다 `fsxadmin` 사용자, 파일 시스템을 생성할 때 구성한 암호 및 에서 관리 DNS 이름 `aws fsx describe-file-systems`.

4. 아래 예에 표시된 대로 인증서와 관리 LIF의 DNS 이름을 사용하여 백엔드 파일을 생성합니다.

YAML

```
---
version: 1
storageDriverName: ontap-san
backendName: customBackendName
managementLIF: svm-XXXXXXXXXXXXXXXXXXXX.fs-XXXXXXXXXXXXXXXXXXXX.fsx.us-
east-2.aws.internal
svm: svm01
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

JSON을 참조하십시오

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "customBackendName",
  "managementLIF": "svm-XXXXXXXXXXXXXXXXXXXX.fs-
XXXXXXXXXXXXXXXXXXXX.fsx.us-east-2.aws.internal",
  "svm": "svm01",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

백엔드 만들기에 대한 자세한 내용은 다음 링크를 참조하십시오.

- ["ONTAP NAS 드라이버를 사용하여 백엔드를 구성합니다"](#)
- ["ONTAP SAN 드라이버를 사용하여 백엔드를 구성합니다"](#)

결과

배포 후에는 을 만들 수 있습니다 ["스토리지 클래스, 볼륨 프로비저닝 및 POD에 볼륨 마운트"](#).

SMB 볼륨 프로비저닝을 위한 준비

를 사용하여 SMB 볼륨을 프로비저닝할 수 있습니다 `ontap-nas` 드라이버. 를 누릅니다 [ONTAP SAN 및 NAS 드라이버 통합](#) 다음 단계를 완료합니다.

단계

1. SMB 공유를 생성합니다. 다음 두 가지 방법 중 하나로 SMB 관리자 공유를 생성할 수 있습니다 ["Microsoft 관리 콘솔"](#) 공유 폴더 스냅인 또는 ONTAP CLI 사용 ONTAP CLI를 사용하여 SMB 공유를 생성하려면 다음을 따르십시오.

a. 필요한 경우 공유에 대한 디렉토리 경로 구조를 생성합니다.

를 클릭합니다 `vserver cifs share create` 명령은 공유를 생성하는 동안 `-path` 옵션에 지정된 경로를 확인합니다. 지정한 경로가 없으면 명령이 실패합니다.

b. 지정된 SVM과 연결된 SMB 공유를 생성합니다.

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

c. 공유가 생성되었는지 확인합니다.

```
vserver cifs share show -share-name share_name
```



을 참조하십시오 ["SMB 공유를 생성합니다"](#) 를 참조하십시오.

2. 백엔드를 생성할 때 SMB 볼륨을 지정하려면 다음을 구성해야 합니다. 모든 ONTAP 백엔드 구성 옵션에 대한 자세한 내용은 을 참조하십시오 ["ONTAP 구성 옵션 및 예제용 FSX"](#).

매개 변수	설명	예
smbShare	공유 폴더 Microsoft 관리 콘솔을 사용하여 생성한 SMB 공유의 이름입니다. 예: "smb-share". * SMB 볼륨에 필요합니다. *	smb-share
nasType	* 를 로 설정해야 합니다 smb. * null인 경우 기본값은 로 설정됩니다 nfs.	smb
securityStyle	새로운 볼륨에 대한 보안 스타일 * 를 로 설정해야 합니다 ntfs 또는 mixed SMB 볼륨용. *	ntfs 또는 mixed SMB 볼륨용
unixPermissions	모드를 선택합니다. SMB 볼륨에 대해서는 * 를 비워 두어야 합니다. *	""

ONTAP 구성 옵션 및 예제용 FSX

Amazon FSx for ONTAP의 백엔드 구성 옵션에 대해 알아보십시오. 이 섹션에서는 백엔드 구성 예를 제공합니다.

백엔드 구성 옵션

백엔드 구성 옵션은 다음 표를 참조하십시오.

매개 변수	설명	예
version		항상 1
storageDriverName	스토리지 드라이버의 이름입니다	"ONTAP-NAS", "ONTAP-NAS-이코노미", "ONTAP-NAS-Flexgroup", "ONTAP-SAN", "ONTAP-SAN-이코노미"
backendName	사용자 지정 이름 또는 스토리지 백엔드	드라이버 이름 + "_" + dataLIF
managementLIF	클러스터 또는 SVM 관리 LIF의 IP 주소 MetroCluster의 원활한 전환을 위해 SVM 관리 LIF를 지정해야 합니다. FQDN(정규화된 도메인 이름)을 지정할 수 있습니다. 을 사용하여 Astra Trident가 설치된 경우 IPv6 주소를 사용하도록 설정할 수 있습니다 --use-ipv6 깃발. IPv6 주소는 대괄호로 묶어야 합니다(예: [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]).	"10.0.0.1", "[2001:1234:ABCD::fee]"
dataLIF	프로토콜 LIF의 IP 주소입니다. * ONTAP NAS 드라이버 *: 데이터 LIF를 지정하는 것이 좋습니다. 제공되지 않는 경우 Astra Trident는 SVM에서 데이터 LIF를 가져옵니다. NFS 마운트 작업에 사용할 FQDN(정규화된 도메인 이름)을 지정하면 여러 데이터 LIF에서 로드 밸런싱을 위해 라운드 로빈 DNS를 생성할 수 있습니다. 초기 설정 후에 변경할 수 있습니다. 을 참조하십시오. * ONTAP SAN 드라이버 *: iSCSI에 대해 지정하지 마십시오. Astra Trident는 ONTAP 선택적 LUN 맵을 사용하여 다중 경로 세션을 설정하는데 필요한 iSCSI LIF를 검색합니다. 데이터 LIF가 명시적으로 정의되어 있으면 경고가 생성됩니다. 을 사용하여 Astra Trident가 설치된 경우 IPv6 주소를 사용하도록 설정할 수 있습니다 --use-ipv6 깃발. IPv6 주소는 대괄호로 묶어야 합니다(예: [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]).	
autoExportPolicy	자동 익스포트 정책 생성 및 업데이트 [Boolean] 활성화 를 사용합니다 autoExportPolicy 및 autoExportCIDRs 옵션, Astra Trident는 익스포트 정책을 자동으로 관리할 수 있습니다.	"거짓"

매개 변수	설명	예
autoExportCIDRs	Kubernetes 노드 IP를 언제 기준으로 필터링하는 CIDR 목록입니다 autoExportPolicy 가 활성화됩니다. 를 사용합니다 autoExportPolicy 및 autoExportCIDRs 옵션, Astra Trident는 익스포트 정책을 자동으로 관리할 수 있습니다.	"["0.0.0.0/0", "::/0"]"
labels	볼륨에 적용할 임의의 JSON 형식 레이블 세트입니다	""
clientCertificate	Base64 - 클라이언트 인증서의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
clientPrivateKey	Base64 - 클라이언트 개인 키의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다	""
trustedCACertificate	Base64 - 신뢰할 수 있는 CA 인증서의 인코딩된 값입니다. 선택 사항. 인증서 기반 인증에 사용됩니다.	""
username	클러스터 또는 SVM에 연결할 사용자 이름입니다. 자격 증명 기반 인증에 사용됩니다. 예: vsadmin.	
password	클러스터 또는 SVM에 연결하는 암호 자격 증명 기반 인증에 사용됩니다.	
svm	사용할 스토리지 가상 머신입니다	SVM 관리 LIF가 지정된 경우에 파생됩니다.
igroupName	사용할 SAN 볼륨에 대한 igroup의 이름입니다. 을 참조하십시오 .	"삼중 - <backend-UUID>"
storagePrefix	SVM에서 새 볼륨을 프로비저닝할 때 사용되는 접두사 생성 후에는 수정할 수 없습니다. 이 매개 변수를 업데이트하려면 새 백엔드를 생성해야 합니다.	"삼중류"
limitAggregateUsage	* NetApp ONTAP * 용 Amazon FSx는 지정하지 마십시오 fsxadmin 및 vsadmin 애그리게이트 사용을 검색하고 Astra Trident를 사용하여 제한하는 데 필요한 권한이 포함되어 있지 않습니다.	사용하지 마십시오.

매개 변수	설명	예
limitVolumeSize	요청된 볼륨 크기가 이 값보다 큰 경우 용량 할당에 실패합니다. 또한 qtree 및 LUN에 대해 관리하는 볼륨의 최대 크기 및 을 제한합니다 qtreesPerFlexvol 옵션을 사용하면 FlexVol당 최대 qtree 수를 사용자 지정할 수 있습니다.	""(기본적으로 적용되지 않음)
lunsPerFlexvol	FlexVol당 최대 LUN 수는 [50, 200] 범위 내에 있어야 합니다. SAN만 해당.	"100"
debugTraceFlags	문제 해결 시 사용할 디버그 플래그입니다. 예: {"api":false, "method":true} 사용하지 마십시오 debugTraceFlags 문제 해결 및 자세한 로그 덤프가 필요한 경우를 제외하고	null입니다
nfsMountOptions	심표로 구분된 NFS 마운트 옵션 목록입니다. Kubernetes 영구 볼륨의 마운트 옵션은 일반적으로 스토리지 클래스에서 지정되지만, 스토리지 클래스에 마운트 옵션을 지정하지 않으면 Astra Trident가 스토리지 백엔드의 구성 파일에 지정된 마운트 옵션을 사용하여 로 돌아갑니다. 스토리지 클래스 또는 구성 파일에 마운트 옵션이 지정되지 않은 경우 Astra Trident는 연결된 영구 볼륨에 마운트 옵션을 설정하지 않습니다.	""
nasType	NFS 또는 SMB 볼륨 생성을 구성합니다. 옵션은 nfs, smb 또는 null입니다. * 를 로 설정해야 합니다 `smb` SMB 볼륨의 경우. * null로 설정하면 기본적으로 NFS 볼륨이 설정됩니다.	"NFS"
qtreesPerFlexvol	FlexVol당 최대 qtree, 범위 [50, 300]에 있어야 함	"200"
smbShare	공유 폴더 Microsoft 관리 콘솔을 사용하여 생성한 SMB 공유의 이름입니다. * SMB 볼륨에 필요합니다. *	"SMB 공유"

매개 변수	설명	예
useREST	ONTAP REST API를 사용하는 부울 매개 변수입니다. * 기술 미리 보기 * useREST 프로덕션 작업 부하가 아닌 테스트 환경에 권장되는 기술 미리 보기로 제공됩니다. 를 로 설정한 경우 true, Astra Trident는 ONTAP REST API를 사용하여 백엔드와 통신합니다. 이 기능을 사용하려면 ONTAP 9.11.1 이상이 필요합니다. 또한 사용되는 ONTAP 로그인 역할에 대한 액세스 권한이 있어야 합니다 ontap 응용 프로그램. 이는 사전 정의된 에 의해 충족됩니다 vsadmin 및 cluster-admin 역할.	"거짓"

에 대한 세부 정보 igroupName

igroupName ONTAP 클러스터에서 이미 생성된 igroup으로 설정할 수 있습니다. 지정되지 않은 경우 Astra Trident가 이름이 인 igroup을 자동으로 생성합니다 trident-<backend-UUID>.

미리 정의된 횡수 이름을 제공하는 경우 환경 간에 SVM을 공유하려면 Kubernetes 클러스터 당 하나의 igroup을 사용하는 것이 좋습니다. 이는 Astra Trident가 IQN 추가 및 삭제를 자동으로 유지 관리하는 데 필요합니다.

- igroupName Astra Trident 외부의 SVM에서 생성 및 관리되는 새로운 igroup을 가리키도록 업데이트할 수 있습니다.
- igroupName 생략할 수 있습니다. 이 경우 Astra Trident가 이름을 가진 igroup을 생성하고 관리합니다 trident-<backend-UUID> 자동으로.

두 경우 모두 볼륨 첨부 파일에 계속 액세스할 수 있습니다. 향후 볼륨 첨부 파일은 업데이트된 igroup을 사용합니다. 이 업데이트는 백엔드에 있는 볼륨에 대한 액세스를 방해하지 않습니다.

업데이트 dataLIF 초기 구성 후

다음 명령을 실행하여 초기 구성 후에 데이터 LIF를 변경할 수 있으며, 업데이트된 데이터 LIF가 포함된 새 백엔드 JSON 파일을 제공할 수 있습니다.

```
tridentctl update backend <backend-name> -f <path-to-backend-json-file-with-updated-dataLIF>
```



PVC가 하나 이상의 포드에 연결된 경우 해당 포드를 모두 내린 다음 다시 불러와서 새 데이터 LIF가 적용되도록 해야 합니다.

볼륨 프로비저닝을 위한 백엔드 구성 옵션

에서 이러한 옵션을 사용하여 기본 프로비저닝을 제어할 수 있습니다 defaults 섹션을 참조하십시오. 예를 들어, 아래 구성 예제를 참조하십시오.

매개 변수	설명	기본값
spaceAllocation	LUN에 대한 공간 할당	"참"
spaceReserve	공간 예약 모드, "없음"(씬) 또는 "볼륨"(일반)	"없음"
snapshotPolicy	사용할 스냅샷 정책입니다	"없음"
qosPolicy	생성된 볼륨에 할당할 QoS 정책 그룹입니다. 스토리지 풀 또는 백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다. Astra Trident와 함께 QoS 정책 그룹을 사용하려면 ONTAP 9.8 이상이 필요합니다. 비공유 QoS 정책 그룹을 사용하고 정책 그룹이 각 구성요소별로 적용되도록 하는 것이 좋습니다. 공유 QoS 정책 그룹은 모든 워크로드의 총 처리량에 대해 상한을 적용합니다.	""
adaptiveQosPolicy	생성된 볼륨에 할당할 적응형 QoS 정책 그룹입니다. 스토리지 풀 또는 백엔드에서 qosPolicy 또는 adapativeQosPolicy 중 하나를 선택합니다. ONTAP에서 지원되지 않음 - NAS - 이코노미	""
snapshotReserve	스냅샷 "0"에 예약된 볼륨의 백분율	If(경우 snapshotPolicy "없음"이고, 그렇지 않으면 ""입니다.
splitOnClone	생성 시 상위 클론에서 클론을 분할합니다	"거짓"
encryption	새 볼륨에 NVE(NetApp Volume Encryption)를 활성화합니다. 기본값은 입니다 false. 이 옵션을 사용하려면 NVE 라이선스가 클러스터에서 활성화되어 있어야 합니다. 백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다. 자세한 내용은 다음을 참조하십시오. "Astra Trident가 NVE 및 NAE와 연동되는 방식" .	"거짓"
luksEncryption	LUKS 암호화를 사용합니다. 을 참조하십시오 "LUKS(Linux Unified Key Setup) 사용" . SAN만 해당.	""
tieringPolicy	"없음"을 사용하는 계층화 정책	ONTAP 9.5 이전 SVM-DR 구성의 경우 "스냅샷 전용"
unixPermissions	모드를 선택합니다. * SMB 볼륨의 경우 비워 둡니다. *	""

매개 변수	설명	기본값
securityStyle	새로운 볼륨에 대한 보안 스타일 NFS를 지원합니다 mixed 및 unix 보안 스타일. SMB 지원 mixed 및 ntfs 보안 스타일.	NFS 기본값은 입니다 unix. SMB 기본값은 입니다 ntfs.

예

사용 nasType, node-stage-secret-name, 및 node-stage-secret-namespace, SMB 볼륨을 지정하고 필요한 Active Directory 자격 증명을 제공할 수 있습니다. SMB 볼륨은 를 사용하여 지원됩니다 ontap-nas 드라이버만 해당.

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nas-smb-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"

```

kubectl로 백엔드를 만듭니다

백엔드는 Astra Trident와 스토리지 시스템 간의 관계를 정의합니다. Astra Trident가 스토리지 시스템과 통신하는 방법과 Astra Trident가 스토리지 시스템에서 볼륨을 프로비저닝하는 방법을 알려줍니다. Astra Trident를 설치한 후 다음 단계는 백엔드를 생성하는 것입니다. 를 클릭합니다 TridentBackendConfig CRD(Custom Resource Definition)를 사용하면 Kubernetes 인터페이스를 통해 Trident 백엔드를 직접 생성 및 관리할 수 있습니다. 를 사용하여 이 작업을 수행할 수 있습니다 kubectl 또는 Kubernetes 배포를 위해 동등한 CLI 도구를 사용할 수 있습니다.

TridentBackendConfig

TridentBackendConfig (tbc, tbconfig, tbackendconfig)는 을 사용하여 Astra Trident 백엔드를 관리할 수 있는 프론트엔드, 이름 있는 CRD입니다 kubectl. 이제 Kubernetes 및 스토리지 관리자는 전용 명령줄 유틸리티 없이도 Kubernetes CLI를 통해 직접 백엔드를 생성 및 관리할 수 있습니다 (tridentctl)를 클릭합니다.

를 생성할 때 TridentBackendConfig 오브젝트, 다음과 같은 현상이 발생합니다.

- 백엔드는 사용자가 제공하는 구성에 따라 Astra Trident에서 자동으로 생성합니다. 이 항목은 내부적으로 로 표시됩니다 TridentBackend (tbe, tridentbackend) CR.
- 를 클릭합니다 TridentBackendConfig 에 고유하게 바인딩됩니다 TridentBackend Astra Trident가 작성했습니다.

각각 TridentBackendConfig 을 사용하여 일대일 매핑을 유지합니다 TridentBackend. 전자는 백엔드를 설계 및 구성하기 위해 사용자에게 제공되는 인터페이스이며, 후자는 Trident가 실제 백엔드 객체를 나타내는 방법입니다.



TridentBackend CRS는 Astra Trident에서 자동으로 생성합니다. 수정할 수 없습니다. 백엔드를 업데이트하려면 을 수정하여 이 작업을 수행합니다 TridentBackendConfig 오브젝트.

의 형식은 다음 예를 참조하십시오 TridentBackendConfig CR:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

의 예를 살펴볼 수도 있습니다 "[Trident - 장착 도구](#)" 원하는 스토리지 플랫폼/서비스의 샘플 구성을 위한 디렉토리입니다.

를 클릭합니다 spec 백엔드 관련 구성 매개 변수를 사용합니다. 이 예에서는 백엔드에서 를 사용합니다 ontap-san 여기에 표로 제공된 구성 매개 변수를 사용하여 스토리지 드라이버를 다운로드합니다. 원하는 스토리지 드라이버에 대한 구성 옵션 목록은 를 참조하십시오 "[스토리지 드라이버에 대한 백엔드 구성 정보입니다](#)".

를 클릭합니다 spec 섹션에는 도 포함되어 있습니다 credentials 및 deletionPolicy 에 새로 도입된 필드입니다 TridentBackendConfig CR:

- credentials: 이 매개 변수는 필수 필드이며 스토리지 시스템/서비스를 인증하는 데 사용되는 자격 증명을 포함합니다. 사용자 생성 Kubernetes Secret으로 설정됩니다. 자격 증명을 일반 텍스트로 전달할 수 없으며 오류가 발생합니다.
- deletionPolicy: 이 필드는 에서 수행해야 하는 작업을 정의합니다 TridentBackendConfig 이(가) 삭제됩니다. 다음 두 가지 값 중 하나를 사용할 수 있습니다.
 - delete: 그러면 두 항목이 모두 삭제됩니다 TridentBackendConfig CR 및 관련 백엔드 이 값이 기본값입니다.
 - retain: 가 TridentBackendConfig CR이 삭제되어도 백엔드 정의가 계속 존재하고 로 관리할 수 있습니다 tridentctl. 삭제 정책을 로 설정합니다 retain 사용자가 이전 릴리스(21.04 이전)로 다운그레이드하고 생성된 백엔드를 유지할 수 있습니다. 이 필드의 값은 이후에 업데이트할 수 있습니다 TridentBackendConfig 이 생성됩니다.



백엔드 이름은 를 사용하여 설정됩니다 spec.backendName. 지정하지 않으면 백엔드 이름이 의 이름으로 설정됩니다 TridentBackendConfig 오브젝트(metadata.name). 을 사용하여 백엔드 이름을 명시적으로 설정하는 것이 좋습니다 spec.backendName.



로 만든 백엔드 tridentctl 연결된 가 없습니다 TridentBackendConfig 오브젝트. 에서 이러한 백엔드를 관리하도록 선택할 수 있습니다 kubectl 을 생성합니다 TridentBackendConfig 있습니다. 동일한 구성 매개 변수(예 spec.backendName, spec.storagePrefix, spec.storageDriverName` 등). Astra Trident가 새로 생성된 을 자동으로 바인딩합니다 `TridentBackendConfig 기존 백엔드를 사용합니다.

단계 개요

을 사용하여 새 백엔드를 생성합니다 `kubectl` 다음을 수행해야 합니다.

1. 을 생성합니다 "쿠버네티스 비밀". 비밀에는 Astra Trident가 스토리지 클러스터/서비스와 통신하는 데 필요한 자격 증명이 포함되어 있습니다.
2. 을 생성합니다 TridentBackendConfig 오브젝트. 스토리지 클러스터/서비스에 대한 자세한 내용과 이전 단계에서 생성한 암호를 참조하십시오.

백엔드를 생성한 후 을 사용하여 해당 상태를 확인할 수 있습니다 kubectl get tbc <tbc-name> -n <trident-namespace> 추가 세부 정보를 수집합니다.

1단계: Kubernetes Secret 생성

백엔드에 대한 액세스 자격 증명이 포함된 암호를 생성합니다. 이는 각 스토리지 서비스/플랫폼마다 다릅니다. 예를 들면 다음과 같습니다.

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: t@Ax@7q(>
```

이 표에는 각 스토리지 플랫폼의 비밀에 포함되어야 하는 필드가 요약되어 있습니다.

스토리지 플랫폼 암호 필드 설명입니다	비밀	필드 설명입니다
Azure NetApp Files	클라이언트 ID입니다	앱 등록에서 클라이언트 ID
GCP용 Cloud Volumes Service	private_key_id	개인 키의 ID입니다. CVS 관리자 역할을 가진 GCP 서비스 계정에 대한 API 키의 일부
GCP용 Cloud Volumes Service	개인 키	개인 키. CVS 관리자 역할을 가진 GCP 서비스 계정에 대한 API 키의 일부

스토리지 플랫폼 암호 필드 설명입니다	비밀	필드 설명입니다
요소(NetApp HCI/SolidFire)	엔드포인트	테넌트 자격 증명이 있는 SolidFire 클러스터의 MVIP입니다
ONTAP	사용자 이름	클러스터/SVM에 연결할 사용자 이름입니다. 자격 증명 기반 인증에 사용됩니다
ONTAP	암호	클러스터/SVM에 연결하는 암호 자격 증명 기반 인증에 사용됩니다
ONTAP	clientPrivateKey를 선택합니다	Base64 - 클라이언트 개인 키의 인코딩된 값입니다. 인증서 기반 인증에 사용됩니다
ONTAP	챠퍼 사용자 이름	인바운드 사용자 이름입니다. useCHAP = TRUE인 경우 필수입니다. 용 ontap-san 및 ontap-san-economy
ONTAP	챠퍼시토키트	CHAP 이니시에이터 암호입니다. useCHAP = TRUE인 경우 필수입니다. 용 ontap-san 및 ontap-san-economy
ONTAP	chapTargetUsername 을 선택합니다	대상 사용자 이름입니다. useCHAP = TRUE인 경우 필수입니다. 용 ontap-san 및 ontap-san-economy
ONTAP	챠퍼타겟이니터시키키트	CHAP 타겟 이니시에이터 암호입니다. useCHAP = TRUE인 경우 필수입니다. 용 ontap-san 및 ontap-san-economy

이 단계에서 생성된 암호는 에서 참조됩니다 spec.credentials 의 필드 TridentBackendConfig 다음 단계에서 만든 개체입니다.

2단계: 을 작성합니다 TridentBackendConfig 있습니다

이제 을 만들 준비가 되었습니다 TridentBackendConfig 있습니다. 이 예에서는 를 사용하는 백엔드를 보여 줍니다 ontap-san 드라이버는 를 사용하여 만듭니다 TridentBackendConfig 아래 개체:

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```



```

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret

```

3단계: 의 상태를 확인합니다 TridentBackendConfig 있습니다

이제 를 만들었습니다 TridentBackendConfig CR, 상태를 확인할 수 있습니다. 다음 예를 참조하십시오.

```

kubectl -n trident get tbc backend-tbc-ontap-san

```

NAME	BACKEND NAME	BACKEND UUID
backend-tbc-ontap-san	ontap-san-backend	8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
Bound	Success	

백엔드가 생성되어 에 바인딩되었습니다 TridentBackendConfig 있습니다.

위상은 다음 값 중 하나를 사용할 수 있습니다.

- **Bound:** TridentBackendConfig CR은 백엔드에 연결되어 있으며 해당 백엔드에는 가 포함되어 있습니다 configRef 로 설정합니다 TridentBackendConfig CR의 uid.
- **Unbound:** 를 사용하여 나타냅니다 "". 를 클릭합니다 TridentBackendConfig 객체가 백엔드에 바인딩되지 않습니다. 모두 새로 생성되었습니다 TridentBackendConfig CRS는 기본적으로 이 단계에 있습니다. 단계가 변경된 후에는 다시 바인딩되지 않은 상태로 되돌릴 수 없습니다.
- **Deleting:** TridentBackendConfig CR deletionPolicy 이(가) 삭제되도록 설정되었습니다. 를 누릅니다 TridentBackendConfig CR이 삭제되어 삭제 상태로 전환됩니다.
 - 백엔드에 지속성 용적 클레임(PVC)이 없는 경우 를 삭제합니다 TridentBackendConfig Astra Trident가 백엔드를 삭제할 뿐 아니라 도 삭제합니다 TridentBackendConfig 있습니다.
 - 백엔드에 PVC가 하나 이상 있는 경우 삭제 상태로 전환됩니다. 를 클릭합니다 TridentBackendConfig 이후에 CR은 삭제 단계를 시작합니다. 백엔드 및 TridentBackendConfig 모든 PVC가 삭제된 후에만 삭제됩니다.
- **Lost:** 와 연결된 백엔드가 있습니다 TridentBackendConfig CR이 실수로 또는 고의적으로 삭제되었으며 및 이(가) 삭제되었습니다 TridentBackendConfig CR에는 삭제된 백엔드에 대한 참조가 여전히 있습니다. 를 클릭합니다 TridentBackendConfig CR은 와 상관없이 삭제할 수 있습니다 deletionPolicy 값.

- Unknown: Astra Trident가 와 연결된 백엔드의 상태 또는 존재를 확인할 수 없습니다 TridentBackendConfig 있습니다. 예를 들어, API 서버가 응답하지 않거나 가 응답하지 않는 경우 tridentbackends.trident.netapp.io CRD가 누락되었습니다. 이 경우 사용자의 개입이 필요할 수 있습니다.

이 단계에서는 백엔드가 성공적으로 생성됩니다! 다음과 같은 몇 가지 작업을 추가로 처리할 수 있습니다 "[백엔드 업데이트 및 백엔드 삭제](#)".

(선택 사항) 4단계: 자세한 내용을 확인하십시오

다음 명령을 실행하여 백엔드에 대한 자세한 정보를 얻을 수 있습니다.

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

NAME	PHASE	STATUS	STORAGE DRIVER	BACKEND NAME	DELETION POLICY	BACKEND UUID
backend-tbc-ontap-san		Bound	Success	ontap-san	delete	8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8

또한 의 YAML/JSON 덤프를 얻을 수도 있습니다 TridentBackendConfig.

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```

apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: "2021-04-21T20:45:11Z"
  finalizers:
  - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound

```

backendInfo 에 가 포함되어 있습니다 backendName 및 backendUUID 에 대한 응답으로 생성된 백엔드 TridentBackendConfig 있습니다. 를 클릭합니다 lastOperationStatus 필드는 의 마지막 작업 상태를 나타냅니다 TridentBackendConfig CR - 사용자가 트리거할 수 있습니다(예: 사용자가 에서 항목을 변경함) spec) 또는 Astra Trident(예: Astra Trident 재시작 중)에 의해 트리거됩니다. 성공 또는 실패일 수 있습니다. phase 간의 관계 상태를 나타냅니다 TridentBackendConfig CR 및 백엔드 위의 예에서 phase 에 값이 바인딩되어 있습니다. 즉, 이 에 대한 것입니다 TridentBackendConfig CR이 백엔드에 연결되어 있습니다.

를 실행할 수 있습니다 `kubectl -n trident describe tbc <tbc-cr-name>` 이벤트 로그의 세부 정보를 가져오는 명령입니다.



연결된 가 포함된 백엔드는 업데이트하거나 삭제할 수 없습니다 TridentBackendConfig 오브젝트 사용 `tridentctl`. 를 사용하여 전환 단계를 이해합니다 `tridentctl` 및 TridentBackendConfig, "[여기 를 참조하십시오](#)".

kubeck을 사용하여 백엔드 관리 수행

을 사용하여 백엔드 관리 작업을 수행하는 방법에 대해 알아봅니다 `kubect1`.

백엔드를 삭제합니다

를 삭제합니다 `TridentBackendConfig`, `Astra Trident`가 백엔드 삭제/보존(을 기반으로 함)을 수행하도록 지시합니다 `deletionPolicy`)를 클릭합니다. 백엔드를 삭제하려면 을 확인하십시오 `deletionPolicy` 가 삭제되도록 설정되어 있습니다. 을 눌러 만 삭제합니다 `TridentBackendConfig``을 참조하십시오 ``deletionPolicy` 유지로 설정되어 있습니다. 이렇게 하면 백엔드가 계속 존재하고 를 사용하여 관리할 수 있습니다 `tridentctl`.

다음 명령을 실행합니다.

```
kubect1 delete tbc <tbc-name> -n trident
```

`Astra Trident`는 에서 사용 중인 `Kubernetes Secrets`를 삭제하지 않습니다 `TridentBackendConfig`. `Kubernetes` 사용자는 기밀을 정해야 합니다. 비밀 정보를 삭제할 때는 주의해야 합니다. 암호는 백엔드에서 사용하지 않는 경우에만 삭제해야 합니다.

기존 백엔드를 봅니다

다음 명령을 실행합니다.

```
kubect1 get tbc -n trident
```

을 실행할 수도 있습니다 `tridentctl get backend -n trident` 또는 `tridentctl get backend -o yaml -n trident` 존재하는 모든 백엔드의 목록을 가져옵니다. 이 목록에는 로 만든 백엔드도 포함됩니다 `tridentctl`.

백엔드를 업데이트합니다

백엔드를 업데이트해야 하는 이유는 여러 가지가 있을 수 있습니다.

- 스토리지 시스템에 대한 자격 증명이 변경되었습니다. 자격 증명을 업데이트하기 위해 에서 사용되는 `Kubernetes Secret`입니다 `TridentBackendConfig` 객체를 업데이트해야 합니다. `Astra Trident`가 자동으로 백엔드를 제공된 최신 자격 증명으로 업데이트합니다. 다음 명령을 실행하여 `Kubernetes Secret`를 업데이트하십시오.

```
kubect1 apply -f <updated-secret-file.yaml> -n trident
```

- 매개 변수(예: 사용 중인 `ONTAP SVM`의 이름)를 업데이트해야 합니다. 이 경우 `TridentBackendConfig` `Kubernetes`를 통해 오브젝트를 직접 업데이트할 수 있습니다.

```
kubect1 apply -f <updated-backend-file.yaml>
```

또는 기존 을 변경합니다 TridentBackendConfig 다음 명령을 실행하여 CR을 실행합니다.

```
kubectl edit tbc <tbc-name> -n trident
```

백엔드 업데이트에 실패하면 백엔드는 마지막으로 알려진 구성으로 계속 유지됩니다. 를 실행하여 로그를 보고 원인을 확인할 수 있습니다 `kubectl get tbc <tbc-name> -o yaml -n trident` 또는 `kubectl describe tbc <tbc-name> -n trident`.

구성 파일의 문제를 확인하고 수정한 후 `update` 명령을 다시 실행할 수 있습니다.

tridentctl을 사용하여 백엔드 관리를 수행합니다

을 사용하여 백엔드 관리 작업을 수행하는 방법에 대해 알아봅니다 `tridentctl`.

백엔드를 생성합니다

을 만든 후 "[백엔드 구성 파일](#)"에서 다음 명령을 실행합니다.

```
tridentctl create backend -f <backend-file> -n trident
```

백엔드 생성에 실패하면 백엔드 구성에 문제가 있는 것입니다. 다음 명령을 실행하여 로그를 보고 원인을 확인할 수 있습니다.

```
tridentctl logs -n trident
```

구성 파일의 문제를 확인하고 수정한 후 를 실행하면 됩니다 `create` 다시 명령을 내립니다.

백엔드를 삭제합니다

Astra Trident에서 백엔드를 삭제하려면 다음을 수행합니다.

1. 백엔드 이름 검색:

```
tridentctl get backend -n trident
```

2. 백엔드를 삭제합니다.

```
tridentctl delete backend <backend-name> -n trident
```



Astra Trident가 백엔드에서 여전히 존재하는 볼륨 및 스냅샷을 프로비저닝한 경우 백엔드를 삭제하면 새 볼륨이 백엔드에서 프로비저닝되지 않습니다. 백엔드는 계속해서 "삭제" 상태에 있으며, Trident는 삭제될 때까지 해당 볼륨 및 스냅샷을 계속 관리합니다.

기존 백엔드를 봅니다

Trident가 알고 있는 백엔드를 보려면 다음을 실행합니다.

- 요약을 보려면 다음 명령을 실행합니다.

```
tridentctl get backend -n trident
```

- 모든 세부 정보를 보려면 다음 명령을 실행합니다.

```
tridentctl get backend -o json -n trident
```

백엔드를 업데이트합니다

새 백엔드 구성 파일을 생성한 후 다음 명령을 실행합니다.

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

백엔드 업데이트에 실패하면 백엔드 구성에 문제가 있거나 잘못된 업데이트를 시도했습니다. 다음 명령을 실행하여 로그를 보고 원인을 확인할 수 있습니다.

```
tridentctl logs -n trident
```

구성 파일의 문제를 확인하고 수정한 후 를 실행하면 됩니다 update 다시 명령을 내립니다.

백엔드를 사용하는 스토리지 클래스를 식별합니다

JSON으로 답할 수 있는 질문의 예입니다 tridentctl 백엔드 객체에 대한 출력입니다. 이 옵션은 를 사용합니다 jq 설치해야 하는 유틸리티입니다.

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name, storageClasses: [.storage[].storageClasses]|unique}]'
```

을 사용하여 만든 백엔드에 대해서도 적용됩니다 TridentBackendConfig.

백엔드 관리 옵션 간 이동

Astra Trident에서 백엔드를 관리하는 다양한 방법에 대해 알아보십시오. 을 소개합니다 `TridentBackendConfig`이제 관리자는 두 가지 고유한 방식으로 백엔드를 관리할 수 있습니다. 이 질문은 다음과 같습니다.

- 을(를) 사용하여 만들 수 있습니다 tridentctl 을 사용하여 관리합니다 TridentBackendConfig?
- 을(를) 사용하여 만들 수 있습니다 TridentBackendConfig 을 사용하여 관리합니다 tridentctl?

관리 tridentctl 을 사용하여 백엔드를 만듭니다 TridentBackendConfig

이 섹션에서는 을 사용하여 만든 백엔드를 관리하는 데 필요한 단계를 설명합니다 tridentctl Kubernetes 인터페이스를 통해 직접 작성합니다 TridentBackendConfig 오브젝트.

이 내용은 다음 시나리오에 적용됩니다.

- 기존 백엔드는 가 없습니다 TridentBackendConfig 을 사용하여 생성되었기 때문입니다 tridentctl.
- 로 만든 새 백 엔드 tridentctl`기타 `TridentBackendConfig 객체가 있습니다.

두 시나리오 모두 Astra Trident가 볼륨을 예약하고 운영하면서 백엔드가 계속 존재할 것입니다. 관리자는 다음 두 가지 옵션 중 하나를 선택할 수 있습니다.

- 를 계속 사용합니다 tridentctl 를 사용하여 만든 백엔드를 관리합니다.
- 을 사용하여 만든 백엔드 바인딩 tridentctl 새로운 제품으로 TridentBackendConfig 오브젝트. 이렇게 하면 백엔드가 를 사용하여 관리됩니다 kubect1 그렇지 않습니다 tridentctl.

를 사용하여 기존 백엔드를 관리합니다 kubect1`을 만들어야 합니다 `TridentBackendConfig 기존 백엔드에 바인딩합니다. 작동 방식에 대한 개요는 다음과 같습니다.

1. Kubernetes 암호를 생성하십시오. 비밀에는 Astra Trident가 스토리지 클러스터/서비스와 통신하는 데 필요한 자격 증명이 포함되어 있습니다.
2. 을 생성합니다 TridentBackendConfig 오브젝트. 스토리지 클러스터/서비스에 대한 자세한 내용과 이전 단계에서 생성한 암호를 참조하십시오. 동일한 구성 매개 변수(예 spec.backendName, spec.storagePrefix, spec.storageDriverName`등). `spec.backendName 기존 백엔드의 이름으로 설정해야 합니다.

단계 0: 백엔드를 식별합니다

을(를) 생성합니다 TridentBackendConfig 이 경우 기존 백엔드에 바인딩되므로 백엔드 구성을 확보해야 합니다. 이 예에서는 다음과 같은 JSON 정의를 사용하여 백엔드를 생성했다고 가정합니다.

```
tridentctl get backend ontap-nas-backend -n trident
+-----+-----+
+-----+-----+-----+-----+
|           NAME           | STORAGE DRIVER |           UUID           |
| STATE   | VOLUMES |           |           |           |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

```
| ontap-nas-backend | ontap-nas | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online | 25 |
+-----+-----+
+-----+-----+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",

  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {"store": "nas_store"},
  "region": "us_east_1",
  "storage": [
    {
      "labels": {"app": "msoffice", "cost": "100"},
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {"app": "mysqldb", "cost": "25"},
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```


1단계: Kubernetes Secret 생성

이 예에 표시된 것처럼 백엔드에 대한 자격 증명이 포함된 암호를 생성합니다.

```
cat tbc-ontap-nas-backend-secret.yaml

apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password

kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

2단계: A를 작성합니다 TridentBackendConfig 있습니다

다음 단계는 을 생성하는 것입니다 TridentBackendConfig 기존 에 자동으로 바인딩되는 CR입니다 ontap-nas-backend (이 예에서와 같이) 다음 요구 사항이 충족되는지 확인합니다.

- 에 동일한 백엔드 이름이 정의되어 있습니다 spec.backendName.
- 구성 매개 변수는 원래 백엔드와 동일합니다.
- 가상 풀(있는 경우)은 원래 백엔드와 동일한 순서를 유지해야 합니다.
- 자격 증명은 일반 텍스트가 아닌 Kubernetes Secret을 통해 제공됩니다.

이 경우, 입니다 TridentBackendConfig 다음과 같이 표시됩니다.

```

cat backend-tbc-ontap-nas.yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
    app: msoffice
    cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
    app: mysqldb
    cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'

kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created

```

3단계: 의 상태를 확인합니다 TridentBackendConfig 있습니다

를 누릅니다 TridentBackendConfig 이(가) 생성되었으며 해당 단계는 이어야 합니다 Bound. 또한 기존 백엔드의 백엔드 이름과 UUID도 동일하게 반영되어야 합니다.

```
kubectl -n trident get tbc tbc-ontap-nas-backend -n trident
NAME                                BACKEND NAME                BACKEND UUID
PHASE    STATUS
tbc-ontap-nas-backend  ontap-nas-backend          52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7    Bound    Success
```

#confirm that no new backends were created (i.e., TridentBackendConfig did not end up creating a new backend)

```
tridentctl get backend -n trident
```

```
+-----+-----+
+-----+-----+-----+-----+
|          NAME          | STORAGE DRIVER |          UUID          |
| STATE  | VOLUMES |          |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ontap-nas-backend     | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |          25 |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

이제 백엔드는 를 사용하여 완전히 관리됩니다 tbc-ontap-nas-backend TridentBackendConfig 오브젝트.

관리 TridentBackendConfig 을 사용하여 백엔드를 만듭니다 tridentctl

```
`tridentctl` 을 사용하여 만든 백엔드를 나열하는 데 사용할 수 있습니다
`TridentBackendConfig`. 또한 관리자는 에서 이러한 백엔드를 완전히 관리하도록 선택할
수도 있습니다 `tridentctl` 삭제합니다 `TridentBackendConfig` 그리고 확실합니다
`spec.deletionPolicy` 가 로 설정되어 있습니다 `retain`.
```

단계 0: 백엔드를 식별합니다

예를 들어, 다음 백엔드가 를 사용하여 생성되었다고 가정해 보겠습니다 TridentBackendConfig:

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                                BACKEND NAME                BACKEND UUID
PHASE  STATUS    STORAGE DRIVER  DELETION POLICY
backend-tbc-ontap-san  ontap-san-backend  81abcb27-ea63-49bb-b606-
0a5315ac5f82  Bound  Success  ontap-san        delete
```

```
tridentctl get backend ontap-san-backend -n trident
+-----+-----+
+-----+-----+-----+-----+
|      NAME      | STORAGE DRIVER |                UUID
| STATE  | VOLUMES |
+-----+-----+
+-----+-----+-----+-----+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-----+-----+
+-----+-----+-----+-----+
```

출력에서 해당 결과가 표시됩니다 TridentBackendConfig 성공적으로 생성되었으며 백엔드에 바인딩되었습니다 [백엔드의 UUID 확인].

1단계: 확인 deletionPolicy 가 로 설정되어 있습니다 retain

의 가치를 살펴보겠습니다 deletionPolicy. 이 설정은 로 설정해야 합니다 retain. 이렇게 하면 가 다음과 같은 경우에 사용할 수 있습니다 TridentBackendConfig CR이 삭제되어도 백엔드 정의가 계속 존재하고 로 관리할 수 있습니다 tridentctl.

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                                BACKEND NAME                BACKEND UUID
PHASE  STATUS    STORAGE DRIVER  DELETION POLICY
backend-tbc-ontap-san  ontap-san-backend  81abcb27-ea63-49bb-b606-
0a5315ac5f82  Bound  Success  ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
 '{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                                BACKEND NAME                BACKEND UUID
PHASE  STATUS    STORAGE DRIVER  DELETION POLICY
backend-tbc-ontap-san  ontap-san-backend  81abcb27-ea63-49bb-b606-
0a5315ac5f82  Bound  Success  ontap-san        retain
```



다음 단계를 진행하지 마십시오 deletionPolicy 가 로 설정되어 있습니다 retain.

2단계: 를 삭제합니다 TridentBackendConfig 있습니다

마지막 단계는 를 삭제하는 것입니다 TridentBackendConfig 있습니다. 를 확인한 후 deletionPolicy 가 로 설정되어 있습니다 `retain`삭제 작업을 계속 수행할 수 있습니다.

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-----+-----+-----+
+-----+-----+-----+
|          NAME          | STORAGE DRIVER |          UUID          |
| STATE | VOLUMES | |
+-----+-----+-----+
+-----+-----+-----+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-0a5315ac5f82 |
| online |      33 | |
+-----+-----+-----+
+-----+-----+-----+
```

를 삭제할 때 TridentBackendConfig Object, Astra Trident는 실제로 백엔드 자체를 삭제하지 않고 간단히 제거합니다.

스토리지 클래스를 관리합니다

스토리지 클래스 생성, 스토리지 클래스 삭제 및 기존 스토리지 클래스 보기에 대한 정보를 찾을 수 있습니다.

스토리지 클래스를 설계합니다

을 참조하십시오 ["여기"](#) 스토리지 클래스의 정의 및 구성 방법에 대한 자세한 내용은 를 참조하십시오.

스토리지 클래스를 생성합니다

스토리지 클래스 파일이 있으면 다음 명령을 실행합니다.

```
kubectl create -f <storage-class-file>
```

<storage-class-file> 스토리지 클래스 파일 이름으로 바꿔야 합니다.

스토리지 클래스를 삭제합니다

Kubernetes에서 스토리지 클래스를 삭제하려면 다음 명령을 실행합니다.

```
kubectl delete storageclass <storage-class>
```

<storage-class> 스토리지 클래스로 교체해야 합니다.

이 스토리지 클래스를 통해 생성된 영구 볼륨은 변경되지 않으며 Astra Trident는 계속 관리합니다.



Astra Trident가 공백을 적용합니다 fsType 볼륨에 대해 생성할 수 있습니다. iSCSI 백엔드의 경우 적용하는 것이 좋습니다 parameters.fsType StorageClass에 있습니다. 기존 StorageClasses를 삭제하고 를 사용하여 다시 생성해야 합니다 parameters.fsType 지정됨.

기존 스토리지 클래스를 봅니다

- 기존 Kubernetes 스토리지 클래스를 보려면 다음 명령을 실행합니다.

```
kubectl get storageclass
```

- Kubernetes 스토리지 클래스 세부 정보를 보려면 다음 명령을 실행합니다.

```
kubectl get storageclass <storage-class> -o json
```

- Astra Trident의 동기화된 스토리지 클래스를 보려면 다음 명령을 실행합니다.

```
tridentctl get storageclass
```

- Astra Trident의 동기화된 스토리지 클래스 세부 정보를 보려면 다음 명령을 실행합니다.

```
tridentctl get storageclass <storage-class> -o json
```

기본 스토리지 클래스를 설정합니다

Kubernetes 1.6에는 기본 스토리지 클래스를 설정하는 기능이 추가되었습니다. 사용자가 영구 볼륨 클레임(PVC)에 영구 볼륨을 지정하지 않는 경우 영구 볼륨을 프로비저닝하는 데 사용되는 스토리지 클래스입니다.

- 주석을 설정하여 기본 스토리지 클래스를 정의합니다 storageclass.kubernetes.io/is-default-class 스토리지 클래스 정의에서 true로 설정합니다. 사양에 따라 다른 값이나 주석 부재는 FALSE로 해석됩니다.
- 다음 명령을 사용하여 기존 스토리지 클래스를 기본 스토리지 클래스로 구성할 수 있습니다.

```
kubectl patch storageclass <storage-class-name> -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 마찬가지로 다음 명령을 사용하여 기본 스토리지 클래스 주석을 제거할 수 있습니다.

```
kubectl patch storageclass <storage-class-name> -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

또한 Trident 설치 프로그램 번들에는 이 주석을 포함하는 예제도 있습니다.



언제든지 클러스터에는 기본 스토리지 클래스가 하나만 있어야 합니다. Kubernetes에서 둘 이상의 작업을 수행하는 것을 기술적으로 금지하지는 않지만 기본 스토리지 클래스가 없는 것처럼 동작합니다.

스토리지 클래스에 대한 백엔드를 식별합니다

JSON으로 답할 수 있는 질문의 예입니다 `tridentctl Astra Trident` 백엔드 객체의 출력입니다. 이 옵션은 `l`를 사용합니다 `jq` 먼저 설치해야 할 수도 있는 유틸리티입니다.

```
tridentctl get storageclass -o json | jq '[.items[] | {storageClass: .Config.name, backends: [.storage]|unique}]'
```

볼륨 작업을 수행합니다

볼륨 관리를 위한 Astra Trident의 기능에 대해 알아보십시오.

- ["CSI 토폴로지를 사용합니다"](#)
- ["스냅샷 작업"](#)
- ["볼륨 확장"](#)
- ["볼륨 가져오기"](#)

CSI 토폴로지를 사용합니다

Astra Trident는 을 사용하여 Kubernetes 클러스터에 있는 노드를 선택적으로 생성하여 연결할 수 있습니다 ["CSI 토폴로지 기능"](#). 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 포드의 예약 요구 사항에 의존하지 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:"1e11e4a2108024935ecfcb2912226cedaafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedaafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 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","kuber-
netes.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","kuber-
netes.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","kuber-
netes.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 `StorageClasses`의 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 `StorageClasses`의 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-centrall
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-a
- topology.kubernetes.io/region: us-centrall
  topology.kubernetes.io/zone: us-centrall-b
storage:
- labels:
    workload: production
    region: Iowa-DC
    zone: Iowa-DC-A
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-a
- labels:
    workload: dev
    region: Iowa-DC
    zone: Iowa-DC-B
    supportedTopologies:
    - topology.kubernetes.io/region: us-centrall
      topology.kubernetes.io/zone: us-centrall-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 **StorageClasses**를 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.

```

kubect1 create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubect1 get pvc
NAME          STATUS      VOLUME      CAPACITY      ACCESS MODES      STORAGECLASS
AGE
pvc-san      Pending
2s
kubect1 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
  ----          -
  Normal        WaitForFirstConsumer 6s    persistentvolume-controller
waiting
for first consumer to be created before binding
  Message
  -----

```

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: voll
      persistentVolumeClaim:
        claimName: pvc-san
  containers:
    - name: sec-ctx-demo
      image: busybox
      command: [ "sh", "-c", "sleep 1h" ]
      volumeMounts:
        - name: voll
          mountPath: /data/demo
      securityContext:
        allowPrivilegeEscalation: false

```

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.

```

kubect1 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>
kubect1 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

PVS(Persistent Volumes)의 Kubernetes VolumeSnapshots(볼륨 스냅샷)을 생성하여 Astra Trident 볼륨의 시점 복제본을 유지할 수 있습니다. 또한 기존 볼륨 스냅샷에서 `_clone_`이라고도 하는 새 볼륨을 생성할 수 있습니다. 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

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

Kubernetes 배포 시 스냅샷 컨트롤러 및 CRD가 포함되지 않은 경우 를 참조하십시오 [볼륨 스냅샷 컨트롤러 배포](#).



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

1단계: 을 작성합니다 `VolumeSnapshotClass`

이 예에서는 볼륨 스냅샷 클래스를 생성합니다.

```

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

```

를 클릭합니다 driver Astra Trident의 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.

자세한 내용은 [LINK:./trident-reference/objects.html#Kubernetes-volumesshotshotclass-objects](#)를 참조하십시오 [VolumeSnapshotClass].

2단계: 기존 PVC의 스냅샷을 생성합니다

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 예제에서는 이름이 인 PVC에 대해 스냅샷이 생성됩니다 pvc1 스냅샷 이름이 로 설정되어 있습니다 pvc1-snap.

```

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

kubectl get volumesnapshots
NAME                AGE
pvc1-snap           50s

```

이렇게 하면 가 생성됩니다 VolumeSnapshot 오브젝트. VolumeSnapshot은 PVC와 유사하며 와 관련이 있습니다 VolumeSnapshotContent 실제 스냅샷을 나타내는 객체입니다.

를 식별할 수 있습니다 VolumeSnapshotContent 의 개체 pvc1-snap VolumeSnapshot을 설명합니다.


```

kubect1 describe volumesnapshots pvcl-snap
Name:          pvcl-snap
Namespace:    default
.
.
.
Spec:
  Snapshot Class Name:  pvcl-snap
  Snapshot Content Name: snapcontent-e8d8a0ca-9826-11e9-9807-525400f3f660
  Source:
    API Group:
    Kind:      PersistentVolumeClaim
    Name:      pvcl
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:  true
  Restore Size:  3Gi
.
.

```

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.

3단계: VolumeSnapshots에서 PVC를 생성합니다

다음은 스냅샷을 사용하여 PVC를 생성하는 예제입니다.

```

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: pvcl-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io

```

dataSource 는 이름이 인 VolumeSnapshot을 사용하여 PVC를 생성해야 함을 나타냅니다 pvc1-snap 데이터 소스로 사용됩니다. 이렇게 하면 Astra Trident가 스냅샷에서 PVC를 생성하도록 지시합니다. PVC가 생성된 후 POD에 부착하여 다른 PVC와 마찬가지로 사용할 수 있습니다.



연결된 스냅샷이 있는 영구 볼륨을 삭제하면 해당 Trident 볼륨이 "삭제 상태"로 업데이트됩니다. Astra Trident 볼륨을 삭제하려면 볼륨의 스냅샷을 제거해야 합니다.

볼륨 스냅샷 컨트롤러 배포

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.yaml
1
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 원하는 네임스페이스에 스냅샷 컨트롤러를 생성합니다. 아래 YAML 매니페스트를 편집하여 네임스페이스를 수정하십시오.

```
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
```

관련 링크

- ["볼륨 스냅샷"](#)
- ["VolumeSnapshotClass"](#)

볼륨 확장

Astra Trident를 사용하면 Kubernetes 사용자가 볼륨을 생성한 후 확장할 수 있습니다. iSCSI 및 NFS 볼륨을 확장하는데 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.

개요

iSCSI PV를 확장하면 다음 단계가 포함됩니다.

- StorageClass 정의를 편집하여 를 설정합니다 allowVolumeExpansion 필드를 눌러 로 이동합니다 true.
- PVC 정의 편집 및 업데이트 spec.resources.requests.storage 원래 크기보다 커야 하는 새로 원하는 크기를 반영합니다.
- 크기를 조절하려면 PV를 포드에 부착해야 합니다. 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.

아래 예에서는 iSCSI PVS의 확장 작동 방식을 보여 줍니다.

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

```
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

```

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.

```

kubect1 get pvc
NAME          STATUS      VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
san-pvc      Bound       pvc-8a814d62-bd58-4253-b0d1-82f2885db671  1Gi
RWO           ontap-san    8s

kubect1 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: PVC를 부착하는 POD를 정의합니다

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: PV를 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.

```

kubect1 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
kubect1 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 kubect1 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 1GB:


```
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.

```

kubect1 get pvc ontapnas20mb
NAME                STATUS      VOLUME
CAPACITY  ACCESS MODES  STORAGECLASS  AGE
ontapnas20mb  Bound        pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7  1Gi
RWO                ontapnas                4m44s

kubect1 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`.

볼륨 가져오기를 지원하는 드라이버입니다

이 표에는 볼륨 가져오기를 지원하는 드라이버와 볼륨 가져오기가 도입된 릴리스가 나와 있습니다.

드라이버	놓습니다
ontap-nas	19.04
ontap-nas-flexgroup	19.04
solidfire-san	19.04
azure-netapp-files	19.04
gcp-cvs	19.04

드라이버	놓습니다
ontap-san	19.04

볼륨을 가져와야 하는 이유는 무엇입니까?

Trident로 볼륨을 가져오는 데는 다음과 같은 몇 가지 사용 사례가 있습니다.

- 응용 프로그램을 Containerizing 하고 기존 데이터 집합을 다시 사용합니다
- 수명이 짧은 애플리케이션에 대한 데이터 세트 클론 사용
- 오류가 발생한 Kubernetes 클러스터를 리빌드합니다
- 재해 복구 중에 애플리케이션 데이터 마이그레이션

가져오기는 어떻게 작동합니까?

영구 볼륨 클레임(PVC) 파일은 볼륨 가져오기 프로세스에서 PVC를 생성하는 데 사용됩니다. 최소한 PVC 파일에는 다음 예제와 같이 이름, 네임스페이스, accessModes 및 storageClassName 필드가 포함되어야 합니다.

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

를 클릭합니다 tridentctl 클라이언트는 기존 스토리지 볼륨을 가져오는 데 사용됩니다. Trident는 볼륨 메타데이터를 유지하고 PVC 및 PV를 생성하여 볼륨을 가져옵니다.

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

스토리지 볼륨을 가져오려면 볼륨을 포함하는 Astra Trident 백엔드의 이름과 스토리지에서 볼륨을 고유하게 식별하는 이름을 지정합니다(예: ONTAP FlexVol, Element Volume, CVS 볼륨 경로). 스토리지 볼륨은 읽기/쓰기 액세스를 허용해야 하며 지정된 Astra Trident 백엔드에서 액세스할 수 있어야 합니다. 를 클릭합니다 -f 문자열 인수가 필요하며 YAML 또는 JSON PVC 파일의 경로를 지정합니다.

Astra Trident가 볼륨 가져오기 요청을 받으면 기존 볼륨 크기를 결정하고 PVC에 설정합니다. 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.

를 사용하여 볼륨을 가져올 때 --no-manage argument, Trident는 개체의 수명 주기 동안 PVC 또는 PV에 대한 추가 작업을 수행하지 않습니다. Trident는 에 대한 PV 및 PVC 이벤트를 무시하므로 --no-manage 객체, PV 삭제 시

스토리지 볼륨이 삭제되지 않습니다. 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.

Trident 19.07 이상 버전에서는 PVS 연결을 처리하고 볼륨을 가져오는 과정에서 볼륨을 마운트합니다. 이전 버전의 Astra Trident를 사용하여 가져오는 경우 데이터 경로에 작업이 없으며 볼륨 가져오기에서 볼륨을 마운트할 수 있는지 여부를 확인하지 않습니다. 볼륨 가져오기에서 오류가 발생한 경우(예: StorageClass가 올바르지 않은 경우) PV에 대한 부가세 반환 청구액 정책을 로 변경하여 복구할 수 있습니다 `retain`PVC와 PV를 삭제하고 볼륨 가져오기 명령을 다시 시도합니다.

ontap-nas 및 ontap-nas-flexgroup 가져오기

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 볼륨을 Trident에서 가져오려면 RW 유형이어야 합니다. 볼륨이 DP 유형인 경우 SnapMirror 대상 볼륨이므로 볼륨을 Trident로 가져오기 전에 미리 관계를 끊어야 합니다.



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

예를 들어, 이라는 이름의 볼륨을 가져옵니다 managed_volume 백엔드에서 을(를) 선택합니다 `ontap_nas`에서 다음 명령을 사용합니다.

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

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

을 눌러 라는 이름의 볼륨을 가져옵니다 unmanaged_volume (에서 ontap_nas backend), Trident에서 관리하지 않을 경우 다음 명령을 사용합니다.

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

를 사용할 때 `--no-manage` argument, Trident는 볼륨의 이름을 바꾸거나 볼륨이 마운트되었는지 검증하지 않습니다. 볼륨이 수동으로 마운트되지 않은 경우 볼륨 가져오기 작업이 실패합니다.



사용자 지정 UnixPermissions를 사용하여 볼륨을 가져오는 기존 버그가 수정되었습니다. PVC 정의 또는 백엔드 구성에서 `unixPermissions`를 지정하고 Astra Trident에 볼륨을 가져오도록 지시할 수 있습니다.

ontap-san 가져오기

Astra Trident는 단일 LUN이 포함된 ONTAP SAN FlexVol도 가져올 수 있습니다. 이는 와 일치합니다 `ontap-san` 드라이버 - 각 PVC 및 FlexVol 내의 LUN에 대한 FlexVol를 생성합니다. 를 사용할 수 있습니다 `tridentctl import` 다른 경우와 동일한 방식으로 명령을 실행합니다.

- 의 이름을 포함합니다 `ontap-san` 백엔드.
- 가져올 FlexVol의 이름을 입력합니다. 이 FlexVol에는 가져와야 하는 LUN이 하나만 포함되어 있습니다.
- 와 함께 사용해야 하는 PVC 정의의 경로를 제공합니다 `-f` 깃발.
- PVC 관리 또는 비관리형 중에서 선택합니다. 기본적으로 Trident는 PVC를 관리하고 백엔드에서 FlexVol 및 LUN의 이름을 바꿉니다. 관리되지 않는 볼륨으로 가져오려면 를 전달합니다 `--no-manage` 깃발.



관리되지 않는 파일을 가져올 때 `ontap-san` 볼륨, FlexVol의 LUN 이름이 지정되었는지 확인해야 합니다 `lun0` 원하는 이니시에이터가 있는 `igroup`에 매핑되어 있습니다. Astra Trident에서 관리되는 가져오기를 위해 이 작업을 자동으로 처리합니다.

그러면 Astra Trident가 FlexVol를 가져와 PVC 정의와 연결합니다. Astra Trident도 FlexVol의 이름을 로 바꿉니다 `pvc-<uuid>` 및 FlexVol 내의 LUN을 에 포맷합니다 `lun0`.



기존 활성 연결이 없는 볼륨을 가져오는 것이 좋습니다. 사용 중인 볼륨을 가져오려는 경우 먼저 볼륨을 클론한 다음 가져오기를 수행합니다.

예

을(를) 가져옵니다 ontap-san-managed 에 있는 FlexVol입니다 ontap_san_default 백엔드에서 를 실행합니다 tridentctl import 명령:

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

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



Astra Trident에서 가져오려면 ONTAP 볼륨이 RW 유형이어야 합니다. 볼륨이 DP 유형인 경우 SnapMirror 대상 볼륨이므로 볼륨을 Astra Trident로 가져오기 전에 미리 관계를 끊어야 합니다.

element 가져오기

Trident를 사용하여 NetApp Element 소프트웨어/NetApp HCI 볼륨을 Kubernetes 클러스터로 가져올 수 있습니다. Astra Trident 백엔드의 이름과 볼륨의 고유 이름 및 PVC 파일이 의 인수로 필요합니다 tridentctl import 명령.

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

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



Element 드라이버는 중복 볼륨 이름을 지원합니다. 중복된 볼륨 이름이 있는 경우 Trident의 볼륨 가져오기 프로세스에서 오류가 반환됩니다. 이 문제를 해결하려면 볼륨을 복제하여 고유한 볼륨 이름을 제공합니다. 그런 다음 복제된 볼륨을 가져옵니다.

gcp-cvs 가져오기



GCP에서 NetApp Cloud Volumes Service가 지원하는 볼륨을 가져오려면 해당 이름 대신 볼륨 경로를 기준으로 볼륨을 식별합니다.

을(를) 가져옵니다 gcp-cvs 백엔드의 볼륨을 호출했습니다 gcpcvs_YEppr 볼륨 경로 포함 `adroit-jolly-swift`에서 다음 명령을 사용합니다.

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

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



볼륨 경로는 / 이후의 볼륨 내보내기 경로 부분입니다. 예를 들어, 내보내기 경로가 인 경우 10.0.0.1:/adroit-jolly-swift, 볼륨 경로는 입니다 adroit-jolly-swift.

azure-netapp-files 가져오기

을(를) 가져옵니다 azure-netapp-files 백엔드의 볼륨을 호출했습니다 azurenetappfiles_40517 볼륨 경로 포함 `importvol1`에서 다음 명령을 실행합니다.

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident
```

PROTOCOL	NAME	BACKEND UUID	SIZE	STATE	STORAGE CLASS	MANAGED
file	pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab	1c01274f-d94b-44a3-98a3-04c953c9a51e	100 GiB	online	anf-storage	true



ANF 볼륨의 볼륨 경로는 다음:/ 이후의 마운트 경로에 있습니다. 예를 들어, 마운트 경로가 인 경우 10.0.0.2:/importvol1, 볼륨 경로는 입니다 importvol1.

네임스페이스 전체에서 NFS 볼륨을 공유합니다

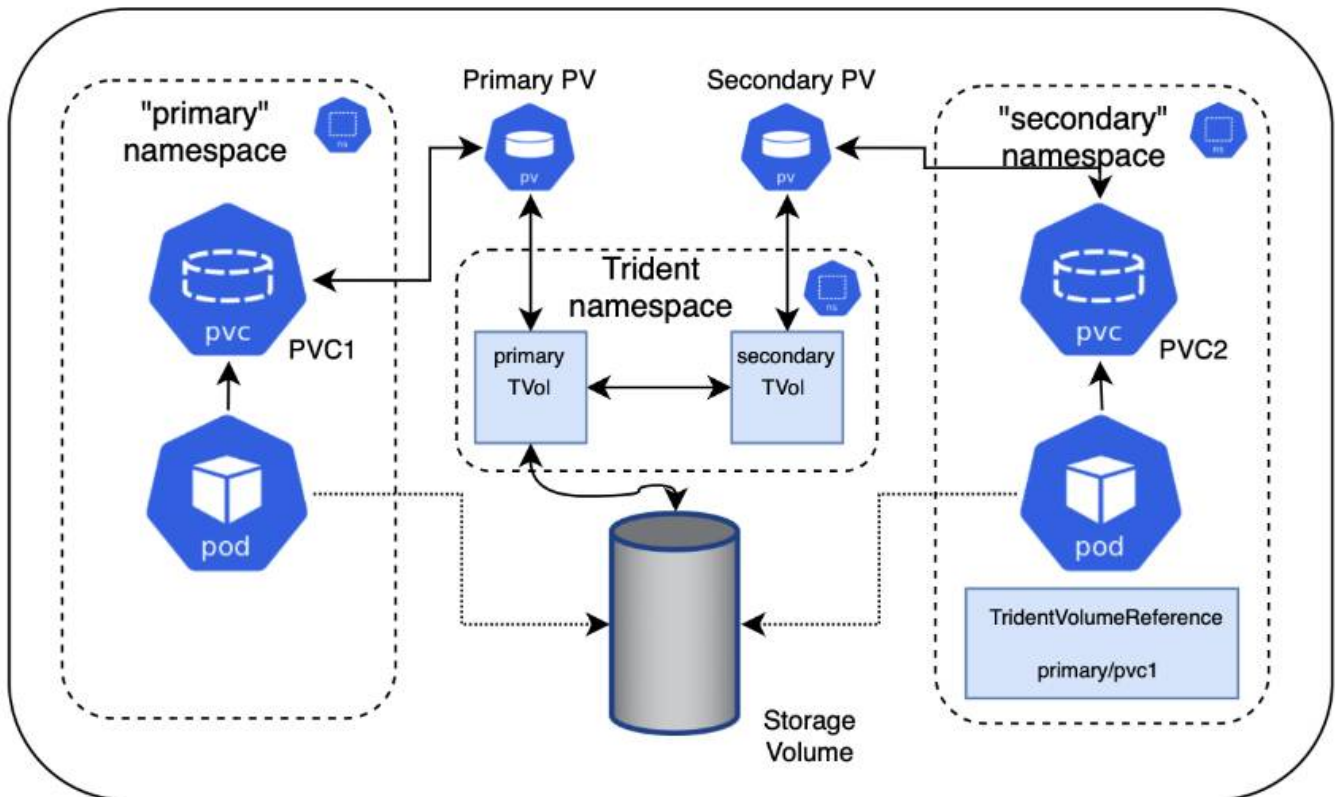
Astra Trident를 사용하면 기본 네임스페이스에서 볼륨을 생성하고 하나 이상의 보조 네임스페이스에서 공유할 수 있습니다.

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 트리엔**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.



- 심표로 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.

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.../trident-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 `hello`를 사용합니다".
- 데모를 시청해보시기 바랍니다 "NetAppTV를 참조하십시오".

Astra Trident를 모니터링합니다

Astra Trident는 Astra Trident의 성능을 모니터링하는 데 사용할 수 있는 Prometheus 메트릭 엔드포인트 세트를 제공합니다.

Astra Trident에서 제공하는 메트릭을 통해 다음을 수행할 수 있습니다.

- Astra Trident의 상태 및 구성을 계속 확인하십시오. 성공적인 작업이 어떻게 이루어지는지, 예상대로 백엔드와 통신할 수 있는지 확인할 수 있습니다.
- 백엔드 사용 정보를 검토하고 백엔드에서 프로비저닝되는 볼륨 수와 사용된 공간 등을 파악합니다.
- 사용 가능한 백엔드에 프로비저닝된 볼륨 양의 매핑을 유지합니다.
- 성과 추적. Astra Trident가 백엔드 및 작업을 수행하는 데 걸리는 시간을 확인할 수 있습니다.



기본적으로 Trident의 메트릭은 타겟 포트에 표시됩니다 8001 를 누릅니다 /metrics 엔드포인트. Trident가 설치된 경우 이러한 메트릭은 기본적으로 * 활성화됩니다.

필요한 것

- Astra Trident가 설치된 Kubernetes 클러스터
- 프로메테우스(Prometheus) 인스턴스. 이것은 일 수 있습니다 "[컨테이너형 Prometheus 구축](#)" 또는 Prometheus를 로 실행하도록 선택할 수 있습니다 "[네이티브 애플리케이션](#)".

1단계: Prometheus 목표를 정의합니다

메트릭을 수집하고 백엔드 Astra Trident가 관리하는, 생성하는 볼륨 등에 대한 정보를 얻으려면 Prometheus 타겟을 정의해야 합니다. 여기 "[블로그](#)" Prometheus 및 Grafana를 Astra Trident와 함께 사용하여 메트릭을 검색하는 방법에 대해 설명합니다. 이 블로그에서는 Kubernetes 클러스터에서 운영자로 Prometheus를 실행하고 Astra Trident의 메트릭을 얻기 위해 ServiceMonitor를 생성하는 방법을 설명합니다.

2단계: Prometheus ServiceMonitor를 만듭니다

Trident 메트릭을 소모하려면 을 감시하는 Prometheus ServiceMonitor를 만들어야 합니다 trident-csi 에 대한 서비스와 수신 metrics 포트. 샘플 ServiceMonitor의 모양은 다음과 같습니다.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - port: metrics
      interval: 15s
```

이 ServiceMonitor 정의는 에서 반환된 메트릭을 검색합니다 trident-csi 을(를) 위해 특별히 찾습니다 metrics 서비스의 끝점입니다. 따라서 이제 Prometheus가 Astra Trident의 메트릭을 이해하도록 구성되었습니다.

Astra Trident에서 직접 제공하는 메트릭 외에도 kubelet은 많은 정보를 제공합니다 kubelet volume * 자체 메트릭 엔드포인트를 통해 측정 Kubelet는 연결된 볼륨, Pod 및 처리하는 기타 내부 작업에 대한 정보를 제공할 수 있습니다. 을 참조하십시오 "여기".

3단계: PromQL을 사용하여 Trident 메트릭 쿼리

PromQL은 시계열 또는 표 형식 데이터를 반환하는 식을 만드는 데 적합합니다.

다음은 사용할 수 있는 몇 가지 PromQL 쿼리입니다.

Trident 상태 정보를 가져옵니다

- Astra Trident** 의 HTTP 2XX 응답률

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()  
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- Astra Trident에서 상태 코드를 통해 얻은 REST 응답의 비율**

```
(sum (trident_rest_ops_seconds_total_count) by (status_code) / scalar  
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- Astra Trident 에 의해 수행된 작업의 평균 지속 시간(ms)

```
sum by (operation)  
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by  
(operation)  
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident 사용 정보를 확인하십시오

- 평균 볼륨 크기

```
trident_volume_allocated_bytes/trident_volume_count
```

- 각 백엔드에서 프로비저닝된 총 볼륨 공간

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

개별 볼륨 사용량을 가져옵니다



이 기능은 kubelet 메트릭도 수집한 경우에만 사용할 수 있습니다.

- 각 볼륨에 사용된 공간의 비율

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes * 100
```

Astra Trident AutoSupport 텔레메트리 에 대해 자세히 알아보십시오

기본적으로 Astra Trident는 Prometheus 메트릭 및 기본 백엔드 정보를 매일 NetApp에 보냅니다.

- Astra Trident가 Prometheus 메트릭 및 기본 백엔드 정보를 NetApp에 보내는 것을 중지하려면 `el` 전달합니다 `--silence-autosupport` Astra Trident를 설치하는 동안 플래그를 지정합니다.
- Astra Trident는 또한 `el` 통해 컨테이너 로그를 NetApp Support On-Demand로 보낼 수 있습니다 `tridentctl send autosupport`. 로그를 업로드하려면 Astra Trident를 트리거해야 합니다. 로그를 제출하기 전에 NetApp의 내용에 동의해야 합니다 <https://www.netapp.com/company/legal/privacy-policy/>["개인 정보 보호 정책"]].
- 지정되지 않은 경우 Astra Trident는 지난 24시간 동안 로그를 가져옵니다.
- `el` 사용하여 로그 보존 기간을 지정할 수 있습니다 `--since` 깃발. 예를 들면 다음과 같습니다. `tridentctl send autosupport --since=1h`. 이 정보는 `el` 통해 수집 및 전송됩니다 `trident-autosupport` Astra Trident와 함께 설치되는 컨테이너입니다. `el`에서 컨테이너 이미지를 얻을 수 있습니다 "[Trident AutoSupport를 누릅니다](#)".
- Trident AutoSupport는 개인 식별 정보(PII) 또는 개인 정보를 수집하거나 전송하지 않습니다. 이 제품은 `el`와 함께 제공됩니다 "[EULA](#)" Trident 컨테이너 이미지 자체에는 적용되지 않습니다. 데이터 보안 및 신뢰에 대한 NetApp의 노력에 대해 자세히 알아볼 수 있습니다 "[여기](#)".

Astra Trident에서 보낸 페이로드의 예는 다음과 같습니다.

```

---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true

```

- AutoSupport 메시지는 NetApp의 AutoSupport 엔드포인트로 전송됩니다. 개인 레지스트리를 사용하여 컨테이너 이미지를 저장하는 경우 를 사용할 수 있습니다 `--image-registry` 깃발.
- 또한 설치 YAML 파일을 생성하여 프록시 URL을 구성할 수도 있습니다. 이 작업은 를 사용하여 수행할 수 있습니다 `tridentctl install --generate-custom-yaml` YAML 파일을 생성하고 을 추가합니다 `--proxy -url` 에 대한 인수 `trident-autosupport` 컨테이너 인합니다 `trident-deployment.yaml`.

Astra Trident 메트릭을 비활성화합니다

- 메트릭을 보고하지 않으려면 을 사용하여 사용자 지정 YAML을 생성해야 합니다 `--generate-custom-yaml` Flag)을 사용하여 를 제거합니다 `--metrics` 에 대해 호출되는 플래그 `'trident-main'` 컨테이너.

Docker를 위한 Astra Trident

배포를 위한 사전 요구 사항

Astra Trident를 구축하기 전에 호스트에 필수 프로토콜 사전 요구 사항을 설치하고 구성해야 합니다.

요구 사항을 확인합니다

- 배포가 모든 를 충족하는지 확인합니다 "요구 사항".
- 지원되는 버전의 Docker가 설치되어 있는지 확인합니다. Docker 버전이 오래되었으면 "설치 또는 업데이트합니다".

```
docker --version
```

- 프로토콜 사전 요구 사항이 호스트에 설치 및 구성되어 있는지 확인합니다.

프로토콜	운영 체제	명령
NFS 를 참조하십시오	RHEL 8+	<code>sudo yum install -y nfs-utils</code>
NFS 를 참조하십시오	우분투	<code>sudo apt-get install -y nfs-common</code>

프로토콜	운영 체제	명령
iSCSI	RHEL 8+	<p>1. 다음 시스템 패키지를 설치합니다.</p> <pre>sudo yum install -y lsscsi iscsi-initiator- utils sg3_utils device- mapper-multipath</pre> <p>2. iscsi-initiator-utils 버전이 6.2.0.874-2.el7 이상인지 확인합니다.</p> <pre>rpm -q iscsi-initiator- utils</pre> <p>3. 스캔을 수동으로 설정합니다.</p> <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> <p>4. 다중 경로 설정:</p> <pre>sudo mpathconf --enable --with_multipathd y --find_multipaths n</pre> <div style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> <p style="text-align: right;">확인합니다</p> <p>etc/multipat h.conf 포함 find_multipa ths no 에서 defaults.</p> </div> <p>5. 확인하십시오 iscsid 및 multipathd 실행 중:</p> <pre>sudo systemctl enable --now iscsid multipathd</pre> <p>6. 활성화 및 시작 iscsi:</p> <pre>sudo systemctl enable --now iscsi</pre>

프로토콜	운영 체제	명령
iSCSI	우분투	<p>1. 다음 시스템 패키지를 설치합니다.</p> <pre>sudo apt-get install -y open-iscsi lsscsi sg3- utils multipath-tools scsitools</pre> <p>2. open-iscsi 버전이 2.0.874-5ubuntu2.10 이상(bionic) 또는 2.0.874-7.1ubuntu6.1 이상(focal)인지 확인합니다.</p> <pre>dpkg -l open-iscsi</pre> <p>3. 스캔을 수동으로 설정합니다.</p> <pre>sudo sed -i 's/^\(node.session.scan \).*\/\1 = manual/' /etc/iscsi/iscsid.conf</pre> <p>4. 다중 경로 설정:</p> <pre>sudo tee /etc/multipath.conf < ←'EOF' defaults { user_friendly_names yes find_multipaths no } EOF sudo systemctl enable --now multipath- tools.service sudo service multipath- tools restart</pre> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>확인합니다</p> <p>etc/multipath.conf 포함 find_multipaths no 에서 defaults.</p> </div> <p>5. 확인하십시오 open-iscsi 및 multipath-tools 활성화 및 실행:</p> <pre>sudo systemctl status multipath-tools sudo systemctl enable --now open- iscsi.service sudo systemctl status</pre>

Astra Trident 구축

open-iscsi

Docker를 위한 Astra Trident는 NetApp의 스토리지 플랫폼을 위한 Docker 에코시스템과 직접 통합됩니다. 또한 향후 추가 플랫폼을 추가할 수 있도록 스토리지 플랫폼에서 Docker 호스트에 이르는 스토리지 리소스의 프로비저닝 및 관리를 지원합니다.

Astra Trident의 여러 인스턴스를 동일한 호스트에서 동시에 실행할 수 있습니다. 이를 통해 여러 스토리지 시스템 및 스토리지 유형에 동시에 연결할 수 있으며, Docker 볼륨에 사용되는 스토리지를 사용자 지정할 수 있습니다.

필요한 것

를 참조하십시오 ["배포를 위한 사전 요구 사항"](#). 필수 구성 요소가 충족되었는지 확인한 후 Astra Trident를 배포할 준비가 되었습니다.

Docker 관리형 플러그인 방법(버전 1.13/17.03 이상)



시작하기 전에

기존 데몬 방법으로 Astra Trident pRE Docker 1.13/17.03을 사용한 경우, 관리 플러그인 방법을 사용하기 전에 Astra Trident 프로세스를 중지하고 Docker 데몬을 다시 시작해야 합니다.

1. 실행 중인 모든 인스턴스 중지:

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. Docker를 다시 시작합니다.

```
systemctl restart docker
```

3. Docker Engine 17.03(새로운 1.13) 이상이 설치되어 있는지 확인합니다.

```
docker --version
```

버전이 최신 버전이 아니면 ["설치를 설치하거나 업데이트합니다"](#).

단계

1. 구성 파일을 생성하고 다음과 같이 옵션을 지정합니다.

- config 기본 파일 이름은 `입니다`config.json`` 그러나 `를 지정하여 원하는 이름을 사용할 수 있습니다`config`` 옵션을 선택합니다. 구성 파일은 `에 있어야 합니다 /etc/netappdvp` 호스트 시스템의 디렉토리입니다.
- log-level: 로깅 수준을 지정합니다 (debug, info, warn, error, fatal)를 클릭합니다. 기본값은 `입니다 info`.
- debug: 디버그 로깅이 설정되었는지 여부를 지정합니다. 기본값은 `false` 입니다. TRUE인 경우 로그 수준을

재정의합니다.

- i. 구성 파일의 위치를 생성합니다.

```
sudo mkdir -p /etc/netappdvp
```

- ii. 구성 파일을 생성합니다.

```
cat << EOF > /etc/netappdvp/config.json
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. 관리 플러그인 시스템을 사용하여 Astra Trident를 시작합니다. 대치 <version> 사용 중인 플러그인 버전(xxx.xx.x)을 사용합니다.

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. Astra Trident를 사용하여 구성된 시스템의 스토리지를 소모합니다.

- a. "firstVolume"이라는 이름의 볼륨을 생성합니다.

```
docker volume create -d netapp --name firstVolume
```

- b. 컨테이너가 시작될 때 기본 볼륨을 생성합니다.

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

- c. "firstVolume" 볼륨을 제거합니다.

```
docker volume rm firstVolume
```

기존 방법(버전 1.12 이하)

시작하기 전에

1. Docker 버전 1.10 이상이 설치되어 있는지 확인합니다.

```
docker --version
```

버전이 최신 버전이 아니면 설치를 업데이트하십시오.

```
curl -fsSL https://get.docker.com/ | sh
```

또는, "배포 지침을 따릅니다".

2. 시스템에 NFS 및/또는 iSCSI가 구성되어 있는지 확인합니다.

단계

1. NetApp Docker Volume Plugin 설치 및 구성:

- a. 응용 프로그램 다운로드 및 압축 풀기:

```
wget  
https://github.com/NetApp/trident/releases/download/v23.01.1/trident-  
installer-23.01.1.tar.gz  
tar xzf trident-installer-23.01.1.tar.gz
```

- b. 용지함 경로의 위치로 이동:

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/  
sudo chown root:root /usr/local/bin/trident  
sudo chmod 755 /usr/local/bin/trident
```

- c. 구성 파일의 위치를 생성합니다.

```
sudo mkdir -p /etc/netappdvp
```

- d. 구성 파일을 생성합니다.

```
cat << EOF > /etc/netappdvp/ontap-nas.json
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. 바이너리를 배치하고 구성 파일을 생성한 후 원하는 구성 파일을 사용하여 Trident 데몬을 시작합니다.

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```



지정하지 않는 한 볼륨 드라이버의 기본 이름은 "NetApp"입니다.

데몬이 시작된 후에는 Docker CLI 인터페이스를 사용하여 볼륨을 생성하고 관리할 수 있습니다

3. 볼륨 생성:

```
docker volume create -d netapp --name trident_1
```

4. 컨테이너를 시작할 때 Docker 볼륨 프로비저닝:

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
alpine ash
```

5. Docker 볼륨 제거:

```
docker volume rm trident_1
docker volume rm trident_2
```

시스템 시작 시 **Astra Trident**를 시작합니다

시스템 기반 시스템용 샘플 장치 파일은 에서 찾을 수 있습니다 contrib/trident.service.example Git repo에 있습니다. RHEL에서 파일을 사용하려면 다음을 수행하십시오.

1. 파일을 올바른 위치에 복사합니다.

실행 중인 인스턴스가 두 개 이상인 경우 단위 파일에 고유한 이름을 사용해야 합니다.

```
cp contrib/trident.service.example
/usr/lib/systemd/system/trident.service
```

- 파일을 편집하고 설명(행 2)을 드라이버 이름과 구성 파일 경로(줄 9)에 맞게 변경하여 환경을 반영합니다.
- IT 부서에서 변경 사항을 수집하도록 시스템 다시 로드:

```
systemctl daemon-reload
```

- 서비스를 활성화합니다.

이 이름은 에서 파일의 이름을 지정하는 항목에 따라 달라집니다 `/usr/lib/systemd/system` 디렉토리.

```
systemctl enable trident
```

- 서비스를 시작합니다.

```
systemctl start trident
```

- 상태를 봅니다.

```
systemctl status trident
```



단위 파일을 수정할 때마다 를 실행합니다 `systemctl daemon-reload` IT 부서에서 변경 사항을 인지할 수 있는 명령입니다.

Astra Trident를 업그레이드하거나 제거합니다

사용 중인 볼륨에 영향을 주지 않고 Docker용 Astra Trident를 안전하게 업그레이드할 수 있습니다. 업그레이드 프로세스 중에는 잠시 시간이 있습니다 `docker volume` 플러그인으로 지정된 명령이 성공하지 않으며 플러그인이 다시 실행될 때까지 응용 프로그램이 볼륨을 마운트할 수 없습니다. 대부분의 경우 몇 초 이내에 완료됩니다.

업그레이드

Docker를 위한 Astra Trident를 업그레이드하려면 다음 단계를 수행하십시오.

단계

- 기존 볼륨 나열:

```
docker volume ls
DRIVER          VOLUME NAME
netapp:latest   my_volume
```

2. 플러그인 비활성화:

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                NAME          DESCRIPTION
ENABLED
7067f39a5df5     netapp:latest nDVP - NetApp Docker Volume
Plugin   false
```

3. 플러그인 업그레이드:

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```



Astra Trident의 18.01 릴리스는 nDVP를 대체합니다. 에서 직접 업그레이드해야 합니다
netapp/ndvp-plugin 에 대한 영상 netapp/trident-plugin 이미지.

4. 플러그인 활성화:

```
docker plugin enable netapp:latest
```

5. 플러그인이 활성화되어 있는지 확인합니다.

```
docker plugin ls
ID                NAME          DESCRIPTION
ENABLED
7067f39a5df5     netapp:latest Trident - NetApp Docker Volume
Plugin   true
```

6. 볼륨이 표시되는지 확인합니다.

```
docker volume ls
DRIVER          VOLUME NAME
netapp:latest   my_volume
```




Astra Trident(20.10 이전)의 이전 버전에서 Astra Trident 20.10 이상으로 업그레이드하는 경우 오류가 발생할 수 있습니다. 자세한 내용은 을 참조하십시오 ["알려진 문제"](#). 오류가 발생하면 먼저 플러그인을 비활성화한 다음 플러그인을 제거한 다음 추가 구성 매개 변수를 전달하여 필요한 Astra Trident 버전을 설치해야 합니다.

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json
```

설치 제거

Docker용 Astra Trident를 제거하려면 다음 단계를 수행하십시오.

단계

1. 플러그인이 생성한 모든 볼륨을 제거합니다.
2. 플러그인 비활성화:

```
docker plugin disable netapp:latest
docker plugin ls
ID                NAME                DESCRIPTION
ENABLED
7067f39a5df5     netapp:latest      nDVP - NetApp Docker Volume
Plugin    false
```

3. 플러그인 제거:

```
docker plugin rm netapp:latest
```

볼륨 작업

표준을 사용하여 볼륨을 쉽게 생성, 클론 복제 및 제거할 수 있습니다 `docker volume` 필요한 경우 지정된 Astra Trident 드라이버 이름의 명령입니다.

볼륨을 생성합니다

- 기본 이름을 사용하여 드라이버로 볼륨을 생성합니다.

```
docker volume create -d netapp --name firstVolume
```

- 특정 Astra Trident 인스턴스를 사용하여 볼륨 생성:

```
docker volume create -d ntap_bronze --name bronzeVolume
```



지정하지 않은 경우 "옵션", 드라이버 기본값이 사용됩니다.

- 기본 볼륨 크기를 재정의합니다. 다음 예를 참조하여 드라이버로 20GiB 볼륨을 생성합니다.

```
docker volume create -d netapp --name my_vol --opt size=20G
```



볼륨 크기는 옵션 단위(예: 10G, 20GB, 3TiB)가 포함된 정수 값이 포함된 문자열로 표시됩니다. 단위를 지정하지 않으면 기본값은 G입니다. 크기 단위는 2(B, KiB, MiB, GiB, TiB) 또는 10(B, KB, MB, GB, TB)의 거듭제곱으로 표현될 수 있습니다. 단축 단위는 2의 거듭제곱을 사용합니다(G=GiB, T=TiB,...).

볼륨을 제거합니다

- 다른 Docker 볼륨과 마찬가지로 볼륨을 제거합니다.

```
docker volume rm firstVolume
```



를 사용할 때 solidfire-san 드라이버, 위의 예에서는 볼륨을 삭제 및 삭제합니다.

Docker를 위한 Astra Trident를 업그레이드하려면 다음 단계를 수행하십시오.

볼륨의 클론을 생성합니다

를 사용할 때 ontap-nas, ontap-san, solidfire-san, 및 gcp-cvs storage drivers, Astra Trident는 볼륨을 복제할 수 있습니다. 를 사용할 때 ontap-nas-flexgroup 또는 ontap-nas-economy 드라이버, 복제는 지원되지 않습니다. 기존 볼륨에서 새 볼륨을 생성하면 새 스냅샷이 생성됩니다.

- 볼륨을 검사하여 스냅샷을 열거합니다.

```
docker volume inspect <volume_name>
```

- 기존 볼륨에서 새 볼륨을 생성합니다. 이렇게 하면 새 스냅샷이 생성됩니다.

```
docker volume create -d <driver_name> --name <new_name> -o  
from=<source_docker_volume>
```

- 볼륨의 기존 스냅샷에서 새 볼륨을 생성합니다. 새 스냅샷은 생성하지 않습니다.

```
docker volume create -d <driver_name> --name <new_name> -o  
from=<source_docker_volume> -o fromSnapshot=<source_snapshot_name>
```

예

```
docker volume inspect firstVolume

[
  {
    "Driver": "ontap-nas",
    "Labels": null,
    "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
    "Name": "firstVolume",
    "Options": {},
    "Scope": "global",
    "Status": {
      "Snapshots": [
        {
          "Created": "2017-02-10T19:05:00Z",
          "Name": "hourly.2017-02-10_1505"
        }
      ]
    }
  }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

외부에서 생성된 볼륨에 액세스합니다

파티션이 없고 파일 시스템이 Astra Trident(예: an)에서 지원되는 경우 Trident * Oonly * 를 사용하여 컨테이너를 통해 외부에서 생성된 블록 장치(또는 해당 클론)에 액세스할 수 있습니다 ext4`형식 지정됨 `/dev/sdc1 Astra Trident를 통해 액세스할 수 없음).

드라이버별 볼륨 옵션

각 스토리지 드라이버에는 다양한 옵션이 있으며, 볼륨 생성 시 이를 지정하여 결과를 사용자 지정할 수 있습니다. 구성된 스토리지 시스템에 적용되는 옵션은 아래를 참조하십시오.

볼륨 생성 작업 중에 이러한 옵션을 사용하는 것은 간단합니다. 을 사용하여 옵션과 값을 제공합니다 -o CLI 작업 중

operator이다. 이러한 값은 JSON 구성 파일의 모든 등가 값을 재정의합니다.

ONTAP 볼륨 옵션

NFS 및 iSCSI에 대한 볼륨 생성 옵션은 다음과 같습니다.

옵션을 선택합니다	설명
size	볼륨의 크기는 기본적으로 1GiB로 설정됩니다.
spaceReserve	볼륨을 씬 또는 일반 프로비저닝합니다. 기본값은 Thin입니다. 유효한 값은 <code>none</code> (씬 프로비저닝됨) 및 <code>volume</code> (일반 프로비저닝).
snapshotPolicy	그러면 스냅샷 정책이 원하는 값으로 설정됩니다. 기본값은 <code>none</code> 즉, 볼륨에 대해 스냅샷이 자동으로 생성되지 않습니다. 스토리지 관리자가 수정하지 않는 한, "default"라는 정책이 모든 ONTAP 시스템에 존재하며, 이 정책은 6시간, 2일, 2주 스냅샷을 생성하고 유지합니다. 스냅샷에 보존된 데이터는 로 이동하여 복구할 수 있습니다 `.snapshot` 디렉토리에 저장됩니다.
snapshotReserve	이렇게 하면 스냅샷 예비 공간이 원하는 비율로 설정됩니다. 기본값은 값이 없습니다. 즉, <code>snapshotPolicy</code> 를 선택한 경우 ONTAP가 <code>snapshotReserve</code> (일반적으로 5%)를 선택하거나 <code>snapshotPolicy</code> 가 <code>none</code> 인 경우 0%를 선택합니다. 모든 ONTAP 백엔드에 대한 구성 파일에서 기본 <code>snapshotReserve</code> 값을 설정할 수 있으며 ONTAP-NAS-이코노미 를 제외한 모든 ONTAP 백엔드에 대한 볼륨 생성 옵션으로 사용할 수 있습니다.
splitOnClone	볼륨을 클론 생성할 때 ONTAP가 상위 클론에서 즉시 클론을 분할합니다. 기본값은 <code>false</code> . 볼륨을 클론 복제하는 사용 사례에는 스토리지 효율성을 높일 기회가 없을 것 같기 때문에 생성 즉시 클론을 상위 볼륨에서 분리하는 것이 가장 좋습니다. 예를 들어 빈 데이터베이스를 복제하면 많은 시간을 절약할 수 있지만 스토리지 절감 효과는 거의 없으므로 즉시 클론을 분리하는 것이 좋습니다.

옵션을 선택합니다	설명
encryption	<p>새 볼륨에 NVE(NetApp Volume Encryption)를 활성화합니다. 기본값은 <code>false</code>. 이 옵션을 사용하려면 NVE 라이선스가 클러스터에서 활성화되어 있어야 합니다.</p> <p>백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다.</p> <p>자세한 내용은 다음을 참조하십시오. "Astra Trident가 NVE 및 NAE와 연동되는 방식".</p>
tieringPolicy	<p>볼륨에 사용할 계층화 정책을 설정합니다. 비활성(콜드) 상태일 때 데이터를 클라우드 계층으로 이동할지 결정합니다.</p>

다음 추가 옵션은 NFS * 에만 적용됩니다 *.

옵션을 선택합니다	설명
unixPermissions	<p>볼륨 자체에 대한 권한 집합을 제어합니다. 기본적으로 사용 권한은 <code>0755</code> 로 설정됩니다. 또는 숫자 표기 <code>0755</code>, 및 <code>root</code> 소유자가 됩니다. 텍스트 또는 숫자 형식이 작동합니다.</p>
snapshotDir	<p>를 <code>0755</code> 로 설정합니다 <code>true</code> 가 <code>0755</code> 을(를) 만듭니다 <code>.snapshot</code> 볼륨을 액세스하는 클라이언트가 볼 수 있는 디렉토리입니다. 기본값은 <code>false</code> `이는 의 가시성을 의미합니다 ` <code>.snapshot</code> 디렉토리는 기본적으로 사용되지 않습니다. 예를 들어, 공식 MySQL 이미지와 같은 일부 이미지는 에서 예상대로 작동하지 않습니다 <code>.snapshot</code> 디렉토리가 표시됩니다.</p>
exportPolicy	<p>볼륨에 사용할 익스포트 정책을 설정합니다. 기본값은 <code>default</code> 입니다.</p>
securityStyle	<p>볼륨에 액세스하는 데 사용할 보안 스타일을 설정합니다. 기본값은 <code>unix</code> 입니다. 유효한 값은 <code>unix</code> 및 <code>mixed</code>.</p>

다음 추가 옵션은 iSCSI * 에만 적용됩니다 *.

옵션을 선택합니다	설명
fileSystemType	<p>iSCSI 볼륨을 포맷하는 데 사용되는 파일 시스템을 설정합니다. 기본값은 <code>ext4</code> 입니다. 유효한 값은 <code>ext3</code>, <code>ext4</code>, 및 <code>xf</code>.</p>

옵션을 선택합니다	설명
spaceAllocation	를 로 설정합니다 false에서는 LUN의 공간 할당 기능을 해제합니다. 기본값은 입니다 `true`즉, ONTAP는 볼륨에 공간이 부족하고 볼륨의 LUN이 쓰기를 허용할 수 없을 때 호스트에 알립니다. 또한 이 옵션을 사용하면 호스트가 데이터를 삭제할 때 ONTAP에서 자동으로 공간을 재확보할 수 있습니다.

예

아래 예를 참조하십시오.

- 10GiB 볼륨 생성:

```
docker volume create -d netapp --name demo -o size=10G -o encryption=true
```

- 스냅샷을 사용하여 100GiB 볼륨 생성:

```
docker volume create -d netapp --name demo -o size=100G -o snapshotPolicy=default -o snapshotReserve=10
```

- setuid 비트가 설정된 볼륨을 생성합니다.

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

최소 볼륨 크기는 20MiB입니다.

스냅샷 예비 공간이 지정되지 않은 경우 스냅샷 정책이 입니다 none, Trident는 0%의 스냅샷 예비 공간을 사용합니다.

- 스냅샷 정책이 없고 스냅샷 예비 공간이 없는 볼륨을 생성합니다.

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- 스냅샷 정책이 없는 볼륨 및 10%의 사용자 지정 스냅샷 예비 공간을 생성합니다.

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none --opt snapshotReserve=10
```

- 스냅샷 정책 및 10%의 사용자 지정 스냅샷 예비 공간이 있는 볼륨을 생성합니다.

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- 스냅샷 정책을 사용하여 볼륨을 생성하고 ONTAP의 기본 스냅샷 예약 공간(일반적으로 5%)을 적용합니다.

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element 소프트웨어 볼륨 옵션

Element 소프트웨어 옵션은 볼륨과 연관된 서비스 품질(QoS) 정책의 크기 및 크기를 표시합니다. 볼륨이 생성되면 볼륨을 사용하여 연결된 QoS 정책을 지정합니다 -o type=service_level 명칭.

Element 드라이버로 QoS 서비스 수준을 정의하는 첫 번째 단계는 하나 이상의 유형을 생성하고 구성 파일의 이름과 연결된 최소, 최대 및 버스트 IOPS를 지정하는 것입니다.

기타 Element 소프트웨어 볼륨 생성 옵션에는 다음이 포함됩니다.

옵션을 선택합니다	설명
size	볼륨 크기, 기본값 1GiB 또는 구성 항목... "기본값":{" 크기":"5G"}.
blocksize	512 또는 4096 중 하나를 사용합니다. 기본값은 512 또는 구성 항목 DefaultBlockSize 입니다.

예

QoS 정의가 포함된 다음 샘플 구성 파일을 참조하십시오.

```

{
  "...": "...",
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}

```

위 구성에서는 Bronze, Silver, Gold의 세 가지 정책 정의가 있습니다. 이러한 이름은 임의로 지정됩니다.

- 10GiB 골드 볼륨 생성:

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 100GiB Bronze 볼륨 생성:

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```


로그를 수집합니다

문제 해결에 도움이 되는 로그를 수집할 수 있습니다. 로그를 수집하는 방법은 Docker 플러그인을 실행하는 방법에 따라 다릅니다.

문제 해결을 위해 로그를 수집합니다

단계

1. 권장 관리 플러그인 방법(예: 사용)을 사용하여 Astra Trident를 실행하는 경우 `docker plugin` 명령 참조, 다음과 같이 봅니다.

```
docker plugin ls
ID                NAME                DESCRIPTION
ENABLED
4fb97d2b956b     netapp:latest      nDVP - NetApp Docker Volume
Plugin           false
journalctl -u docker | grep 4fb97d2b956b
```

표준 로깅 수준에서는 대부분의 문제를 진단할 수 있어야 합니다. 충분하지 않은 경우 디버그 로깅을 활성화할 수 있습니다.

2. 디버그 로깅을 사용하려면 디버그 로깅을 사용하도록 설정한 플러그인을 설치합니다.

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

또는 플러그인이 이미 설치된 경우 디버그 로깅을 활성화합니다.

```
docker plugin disable <plugin>
docker plugin set <plugin> debug=true
docker plugin enable <plugin>
```

3. 호스트에서 바이너리 자체를 실행하는 경우 호스트 에서 로그를 사용할 수 있습니다 `/var/log/netappdvp` 디렉토리. 디버그 로깅을 활성화하려면 를 지정합니다 `-debug` 플러그인을 실행할 때

일반적인 문제 해결 팁

- 새 사용자가 실행하는 가장 일반적인 문제는 플러그 인을 초기화할 수 없도록 잘못 구성된 것입니다. 이 경우 플러그인을 설치하거나 활성화하려고 할 때 다음과 같은 메시지가 표시될 수 있습니다.

```
Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
connect: no such file or directory
```

즉, 플러그인을 시작하지 못했습니다. 다행히 플러그인은 사용자가 겪을 수 있는 대부분의 문제를 진단하는 데

도움이 되는 포괄적인 로깅 기능을 갖추고 있습니다.

- 컨테이너에 PV를 장착하는 데 문제가 있는 경우, 반드시 확인하십시오 `rpcbind` 가 설치되어 실행 중입니다. 호스트 OS에 필요한 패키지 관리자를 사용하여 확인합니다 `rpcbind` 가 실행 중입니다. 를 실행하여 `rpcbind` 서비스의 상태를 확인할 수 있습니다 `systemctl status rpcbind` 또는 그와 동등한 제품이라는 것입니다.

여러 Astra Trident 인스턴스를 관리합니다

여러 스토리지 구성을 동시에 사용할 수 있도록 하려는 경우 Trident의 여러 인스턴스가 필요합니다. 여러 인스턴스의 키는 `l` 를 사용하여 다른 이름을 지정하는 것입니다 `--alias` 컨테이너형 플러그인 옵션, 또는 `--volume-driver` 호스트에서 Trident를 인스턴스화할 때 선택할 수 있는 옵션입니다.

Docker 관리 플러그인 단계(버전 1.13/17.03 이상)

1. 별칭 및 구성 파일을 지정하는 첫 번째 인스턴스를 시작합니다.

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. 다른 별칭과 구성 파일을 지정하여 두 번째 인스턴스를 시작합니다.

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. 별칭을 드라이버 이름으로 지정하는 볼륨을 생성합니다.

예를 들어 금괴 볼륨의 경우:

```
docker volume create -d gold --name ntapGold
```

예를 들어, 은 볼륨의 경우:

```
docker volume create -d silver --name ntapSilver
```

기존(버전 1.12 이하) 단계

1. 사용자 지정 드라이버 ID를 사용하여 NFS 구성으로 플러그인을 시작합니다.

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. 사용자 지정 드라이버 ID를 사용하여 iSCSI 구성으로 플러그인을 시작합니다.

```
sudo trident --volume-driver=netapp-san --config=/path/to/config-iscsi.json
```

3. 각 드라이버 인스턴스에 Docker 볼륨 프로비저닝:

예를 들어, NFS의 경우:

```
docker volume create -d netapp-nas --name my_nfs_vol
```

예를 들어 iSCSI의 경우:

```
docker volume create -d netapp-san --name my_iscsi_vol
```

스토리지 구성 옵션

Astra Trident 구성에 사용할 수 있는 구성 옵션을 확인하십시오.

글로벌 구성 옵션

이러한 구성 옵션은 사용 중인 스토리지 플랫폼에 관계없이 모든 Astra Trident 구성에 적용됩니다.

옵션을 선택합니다	설명	예
version	구성 파일 버전 번호입니다	1
storageDriverName	스토리지 드라이버 이름입니다	ontap-nas, ontap-san, ontap-nas-economy, ontap-nas-flexgroup, solidfire-san
storagePrefix	볼륨 이름에 대한 선택적 접두어입니다. 기본값: netappdvp_.	staging_
limitVolumeSize	볼륨 크기에 대한 선택적 제한. 기본값: ""(강제 적용 안 됨)	10g



사용하지 마십시오 storagePrefix 요소 백엔드에 대한 (기본값 포함). 기본적으로 은(는) 입니다 solidfire-san 드라이버는 이 설정을 무시하고 접두어를 사용하지 않습니다. Docker 볼륨 매핑에 특정 tenantID를 사용하거나 이름 문자가 사용된 경우 Docker의 Docker 버전, 드라이버 정보 및 원시 이름으로 채워진 특성 데이터를 사용하는 것이 좋습니다.

생성하는 모든 볼륨에서 기본 옵션을 지정하지 않아도 됩니다. 를 클릭합니다 size 옵션은 모든 컨트롤러 유형에 사용할 수 있습니다. 기본 볼륨 크기를 설정하는 방법은 ONTAP 구성 섹션을 참조하십시오.

옵션을 선택합니다	설명	예
size	새 볼륨의 선택적 기본 크기입니다. 기본값: 1G	10G

ONTAP 구성

위의 글로벌 구성 값 외에도 ONTAP를 사용할 경우 다음과 같은 최상위 옵션을 사용할 수 있습니다.

옵션을 선택합니다	설명	예
managementLIF	ONTAP 관리 LIF의 IP 주소입니다. FQDN(정규화된 도메인 이름)을 지정할 수 있습니다.	10.0.0.1
dataLIF	<p>프로토콜 LIF의 IP 주소입니다.</p> <ul style="list-style-type: none"> • ONTAP NAS 드라이버 *: 지정할 것을 권장합니다 dataLIF. 제공되지 않는 경우 Astra Trident는 SVM에서 데이터 LIF를 가져옵니다. NFS 마운트 작업에 사용할 FQDN(정규화된 도메인 이름)을 지정하면 여러 데이터 LIF에서 로드 밸런싱을 위해 라운드 로빈 DNS를 생성할 수 있습니다. • ONTAP SAN 드라이버 *: iSCSI에 대해 지정하지 마십시오. Astra Trident가 사용합니다 "ONTAP 선택적 LUN 맵" 다중 경로 세션을 설정하는 데 필요한 iSCI LIF를 검색하려면 경고 발생 시 dataLIF 명시적으로 정의됩니다. 	10.0.0.2
svm	사용할 스토리지 가상 머신(관리 LIF가 클러스터 LIF인 경우 필요)	svm_nfs
username	스토리지 디바이스에 접속할 사용자 이름입니다	vsadmin
password	스토리지 디바이스에 연결하는 암호입니다	secret

옵션을 선택합니다	설명	예
aggregate	프로비저닝을 위한 애그리게이트(선택 사항, SVM에 셋팅해야 하는 경우) 의 경우 <code>ontap-nas-flexgroup</code> 드라이버, 이 옵션은 무시됩니다. SVM에 할당된 모든 애그리게이트는 FlexGroup 볼륨을 프로비저닝하는 데 사용됩니다.	aggr1
limitAggregateUsage	선택 사항으로, 사용량이 이 백분율보다 높을 경우 용량 할당에 실패합니다	75%
nfsMountOptions	NFS 마운트 옵션을 세밀하게 제어합니다. 기본값은 "-o nfsvers=3"입니다. * 에서만 사용할 수 있습니다 <code>ontap-nas</code> 및 <code>ontap-nas-economy</code> 드라이버 * . "NFS 호스트 구성 정보는 여기 를 참조하십시오".	-o nfsvers=4
igroupName	플러그인에서 사용되는 igroup. 기본값은 <code>netappdvp</code> . * ONTAP-SAN driver * 에서만 사용 가능.	myigroup
limitVolumeSize	최대 요청 가능 볼륨 크기 및 <code>qtree</code> 상위 볼륨 크기. 의 경우 * <code>ontap-nas-economy</code> 드라이버, 이 옵션은 생성된 FlexVol의 크기를 추가로 제한합니다.	300g
qtreesPerFlexvol	FlexVol당 최대 <code>qtree</code> 는 범위 [50, 300]에 있어야 하며 기본값은 200입니다. 의 경우 * <code>ontap-nas-economy</code> 드라이버, 이 옵션을 사용하면 FlexVol*당 최대 <code>qtree</code> 수를 사용자 지정할 수 있습니다.	300

생성하는 모든 볼륨에 기본 옵션을 지정하지 않아도 됩니다.

옵션을 선택합니다	설명	예
spaceReserve	공간 예약 모드; none (싹 프로비저닝됨) 또는 <code>volume</code> (두꺼운)	none

옵션을 선택합니다	설명	예
snapshotPolicy	사용할 스냅샷 정책이며 기본값은 입니다 none	none
snapshotReserve	스냅샷 예비 공간 비율, 기본값은 "이며 ONTAP 기본값을 사용합니다	10
splitOnClone	생성 시 상위 클론에서 클론을 분할합니다. 기본값은 입니다 false	false
encryption	새로운 볼륨에 NVE(NetApp Volume Encryption)를 지원하며 기본값은 로 설정됩니다 false. 이 옵션을 사용하려면 NVE 라이선스가 클러스터에서 활성화되어 있어야 합니다. 백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다. 자세한 내용은 다음을 참조하십시오. "Astra Trident가 NVE 및 NAE와 연동되는 방식" .	참
unixPermissions	프로비저닝된 NFS 볼륨에 대한 NAS 옵션, 기본값은 로 설정됩니다 777	777
snapshotDir	에 액세스하기 위한 NAS 옵션입니다 .snapshot Directory(디렉터리), 기본값 false	true
exportPolicy	NFS 익스포트 정책에서 사용할 NAS 옵션이며 기본값은 입니다 default	default
securityStyle	프로비저닝된 NFS 볼륨에 액세스하기 위한 NAS 옵션입니다. NFS를 지원합니다 mixed 및 unix 보안 스타일. 기본값은 입니다 unix.	unix
fileSystemType	SAN 옵션 - 파일 시스템 유형을 선택합니다. 기본값은 입니다 ext4	xf
tieringPolicy	사용할 계층화 정책, 기본값은 입니다 none; snapshot-only ONTAP 9.5 이전 SVM-DR 구성용	none

축척 옵션

를 클릭합니다 `ontap-nas` 및 `ontap-san` 드라이버는 각 Docker 볼륨에 대해 ONTAP FlexVol을 생성합니다. ONTAP는 최대 12,000개의 FlexVol 클러스터를 사용하여 클러스터 노드당 최대 1,000개의 FlexVol을 지원합니다. Docker 볼륨 요구 사항이 이러한 제한 범위 내에 있으면 를 참조하십시오 `ontap-nas` 드라이버는 Docker 볼륨 세부 스냅샷 및 클론 복제와 같은 FlexVol에서 제공되는 추가 기능 때문에 선호되는 NAS 솔루션입니다.

FlexVol 제한에 수용 가능한 것보다 더 많은 Docker 볼륨이 필요한 경우 를 선택합니다 `ontap-nas-economy` 또는 을 누릅니다 `ontap-san-economy` 드라이버.

를 클릭합니다 `ontap-nas-economy` 드라이버는 자동으로 관리되는 FlexVol 풀 내에 Docker 볼륨을 ONTAP `qtree`로 생성합니다. `qtree`는 일부 기능을 희생하여 클러스터 노드당 최대 100,000 및 클러스터당 2,400,000까지 훨씬 더 뛰어난 확장을 제공합니다. 를 클릭합니다 `ontap-nas-economy` 드라이버는 Docker 볼륨 세부 스냅샷 또는 클론 복제를 지원하지 않습니다.



를 클릭합니다 `ontap-nas-economy` Swarm은 여러 노드에서 볼륨 생성을 오케스트레이션하지 않으므로 현재 Docker Swarm에서 드라이버가 지원되지 않습니다.

를 클릭합니다 `ontap-san-economy` 드라이버는 자동으로 관리되는 FlexVol의 공유 풀 내에 Docker 볼륨을 ONTAP LUN으로 생성합니다. 이렇게 하면 각 FlexVol이 하나의 LUN에만 제한되지 않으며 SAN 워크로드에 더 나은 확장성을 제공합니다. 스토리지 시스템에 따라 ONTAP는 클러스터당 최대 16384개의 LUN을 지원합니다. 볼륨이 그 아래에 LUN이 있으므로 이 드라이버는 Docker 볼륨 세부 스냅샷 및 클론 복제를 지원합니다.

를 선택합니다 `ontap-nas-flexgroup` 수십억 개의 파일로 구성된 페타바이트 규모로 성장할 수 있는 단일 볼륨으로 병렬 처리를 늘립니다. FlexGroups의 이상적인 사용 사례로는 AI/ML/DL, 빅데이터 및 분석, 소프트웨어 빌드, 스트리밍, 파일 저장소 등이 있습니다. Trident는 FlexGroup 볼륨을 프로비저닝할 때 SVM에 할당된 모든 애그리게이트를 사용합니다. Trident의 FlexGroup 지원에도 다음과 같은 고려 사항이 있습니다.

- ONTAP 버전 9.2 이상이 필요합니다.
- 이번 작부터 FlexGroups는 NFS v3만 지원합니다.
- SVM에 대해 64비트 NFSv3 식별자를 사용하는 것이 좋습니다.
- 최소 권장 FlexGroup 크기는 100GB입니다.
- FlexGroup 볼륨에 대해서는 클론 생성이 지원되지 않습니다.

FlexGroups에 적합한 FlexGroups 및 워크로드에 대한 자세한 내용은 를 참조하십시오 "[NetApp FlexGroup 볼륨 모범 사례 및 구현 가이드](#) 를 참조하십시오".

같은 환경에서 고급 기능과 대규모 확장을 위해 Docker Volume Plugin의 여러 인스턴스를 를 사용하여 실행할 수 있습니다 `ontap-nas` 또 다른 방법은 을 사용하는 것입니다 `ontap-nas-economy`.

ONTAP 구성 파일의 예

- NFS의 예 `ontap-nas` 드라이버 *

```

{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}

```

- NFS의 예 | ontap-nas-flexgroup 드라이버 *

```

{
  "version": 1,
  "storageDriverName": "ontap-nas-flexgroup",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}

```

- NFS의 예 | ontap-nas-economy 드라이버 *


```

{
  "version": 1,
  "storageDriverName": "ontap-nas-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}

```

- iSCSI의 예 ontap-san 드라이버 *

```

{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}

```

- NFS의 예 ontap-san-economy 드라이버 *

```

{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "myigroup"
}

```

Element 소프트웨어 구성

Element 소프트웨어(NetApp HCI/SolidFire)를 사용하는 경우 글로벌 구성 값 외에도 이러한 옵션을 사용할 수 있습니다.

옵션을 선택합니다	설명	예
Endpoint	<a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">https://<login>:<password>@<mvip>/json-rpc/<element-version>; 으로 문의하십시오	https://admin:admin@192.168.160.3/json-rpc/8.0 으로 문의하십시오
SVIP	iSCSI IP 주소 및 포트	10.0.0.7:3260
TenantName	사용할 SolidFireF 테넌트(찾을 수 없는 경우 생성됨)	docker
InitiatorIFace	iSCSI 트래픽을 기본 인터페이스가 아닌 인터페이스로 제한할 때 인터페이스를 지정합니다	default
Types	QoS 사양	아래 예를 참조하십시오
LegacyNamePrefix	업그레이드된 Trident 설치의 접두사 1.3.2 이전 버전의 Trident를 사용하고 기존 볼륨을 업그레이드하는 경우 이 값을 설정하여 볼륨 이름 방법을 통해 매핑된 이전 볼륨에 액세스해야 합니다.	netappdvp-

를 클릭합니다 `solidfire-san` 드라이버가 Docker Swarm을 지원하지 않습니다.

Element 소프트웨어 구성 파일의 예

```

{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
  "SVIP": "10.0.0.7:3260",
  "TenantName": "docker",
  "InitiatorIFace": "default",
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}

```

알려진 문제 및 제한 사항

Docker에서 Astra Trident를 사용할 때 알려진 문제 및 제한 사항에 대한 정보를 확인하십시오.

이전 버전에서 **Trident Docker** 볼륨 플러그인을 **20.10** 이상으로 업그레이드하면 해당 파일 또는 디렉터리 오류가 없는 업그레이드 오류가 발생합니다.

해결 방법

1. 플러그인을 비활성화합니다.

```
docker plugin disable -f netapp:latest
```

2. 플러그인을 제거합니다.

```
docker plugin rm -f netapp:latest
```

3. 추가 를 제공하여 플러그인을 다시 설치합니다 config 매개 변수.

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant  
-all-permissions config=config.json
```

볼륨 이름은 최소 **2**자 이상이어야 합니다.



이는 Docker 클라이언트의 제한 사항입니다. 클라이언트는 단일 문자 이름을 Windows 경로로 해석합니다. "[버그 25773을 참조하십시오](#)".

Docker Swarm에는 **Astra Trident**가 모든 스토리지 및 드라이버 조합에서 이를 지원하지 않는 특정 동작이 있습니다.

- 현재 Docker Swarm은 볼륨 ID 대신 볼륨 이름을 고유한 볼륨 식별자로 사용합니다.
- 볼륨 요청은 Swarm 클러스터의 각 노드로 동시에 전송됩니다.
- 볼륨 플러그인(Astra Trident 포함)은 Swarm 클러스터의 각 노드에서 독립적으로 실행해야 합니다. ONTAP의 작동 방식 및 방법 때문입니다 ontap-nas 및 ontap-san 운전자 기능, 이러한 제한 내에서 작동할 수 있는 유일한 운전자이다.

나머지 드라이버에는 명확한 "승자" 없이 단일 요청에 대해 대량의 볼륨을 생성할 수 있는 경합 상태와 같은 문제가 있습니다. 예를 들어, Element에는 볼륨의 이름이 같지만 ID가 다를 수 있는 기능이 있습니다.

NetApp은 Docker 팀에 피드백을 제공했지만, 향후 소구에 대한 표시는 제공하지 않습니다.

FlexGroup를 프로비저닝하고 있는 경우, 두 번째 **FlexGroup**에 프로비저닝되는 **FlexGroup**와 공통되는 하나 이상의 애그리게이트가 있는 경우 **ONTAP**는 두 번째 **FlexGroup**를 프로비저닝하지 않습니다.

자주 묻는 질문

Astra Trident의 설치, 구성, 업그레이드 및 문제 해결에 대해 자주 묻는 질문에 대한 답변을 찾아보십시오.

일반적인 질문

Astra Trident는 얼마나 자주 출시됩니까?

Astra Trident는 1월, 4월, 7월, 10월에 3개월마다 출시됩니다. Kubernetes 릴리스 한 달 후입니다.

Astra Trident가 특정 버전의 Kubernetes에서 릴리스된 모든 기능을 지원합니까?

Astra Trident는 일반적으로 Kubernetes의 알파 기능을 지원하지 않습니다. Trident는 Kubernetes 베타 릴리즈를 따르는 두 Trident 릴리스 내에서 베타 기능을 지원할 수 있습니다.

Astra Trident가 작동을 위해 다른 NetApp 제품에 종속되어 있습니까?

Astra Trident는 다른 NetApp 소프트웨어 제품에 종속되어 있지 않으며 독립 실행형 애플리케이션으로 작동합니다. 그러나 NetApp 백엔드 스토리지 디바이스가 있어야 합니다.

Astra Trident 구성 세부 정보를 모두 얻으려면 어떻게 해야 합니까?

를 사용합니다 `tridentctl get` Astra Trident 구성에 대한 자세한 내용을 보려면 명령을 입력합니다.

Astra Trident에서 스토리지를 프로비저닝하는 방법에 대한 메트릭을 얻을 수 있습니까?

예. Trident 20.01에는 관리되는 백엔드 수, 프로비저닝된 볼륨 수, 사용된 바이트 수 등 Astra Trident의 작업에 대한 정보를 수집하는 데 사용할 수 있는 Prometheus 엔드포인트가 도입되었습니다. Cloud Insights를 사용하여 모니터링 및 분석할 수도 있습니다.

CSI Provisioner로 Astra Trident를 사용할 때 사용자 환경이 달라집니까?

아니요 사용자 환경과 기능에 관한 한 변경 사항은 없습니다. 사용된 공급자 이름은 `csi.trident.netapp.io`. 현재 및 향후 릴리즈에서 제공하는 모든 새로운 기능을 사용하려는 경우 이 Astra Trident 설치 방법을 사용하는 것이 좋습니다.

Kubernetes 클러스터에 Astra Trident를 설치 및 사용합니다

지원되는 버전의 은 무엇입니까 etcd?

Astra Trident가 더 이상 필요하지 않습니다 `etcd`. CRD를 사용하여 상태를 유지합니다.

Astra Trident가 개인 레지스트리에서 오프라인 설치를 지원합니까?

예, Astra Trident는 오프라인으로 설치할 수 있습니다. 을 참조하십시오 ["여기"](#).

Astra Trident를 원격으로 설치할 수 있습니까?

예. Astra Trident 18.10 이상은 가 설치된 모든 시스템에서 원격 설치 기능을 지원합니다 `kubectl` 클러스터에 대한 액세스. 이후 `kubectl` 액세스가 확인됩니다(예: 시작) `kubectl get nodes` 확인할 원격 시스템의 명령), 설치 지침을 따릅니다.

Astra Trident를 사용하여 고가용성을 구성할 수 있습니까?

Astra Trident는 하나의 인스턴스로 Kubernetes 배포(ReplicaSet)로 설치되었으므로 HA가 내장되어 있습니다. 구축 시 복제본 수를 늘리지 않아야 합니다. Astra Trident가 설치된 노드가 손실되거나 POD에 액세스할 수 없는 경우 Kubernetes가 자동으로 Pod를 클러스터의 정상 노드에 다시 배포합니다. Astra Trident는 컨트롤 플레인만 지원하므로 Astra Trident를 다시 구축할 경우 현재 마운트된 Pod는 영향을 받지 않습니다.

Astra Trident에서 kube-system 네임스페이스에 액세스해야 합니까?

Astra Trident가 Kubernetes API Server에서 읽어 애플리케이션에서 새로운 PVC를 요청할 시기를 결정하므로 `kube-system`에 액세스해야 합니다.

Astra Trident에서 사용하는 역할 및 권한은 무엇입니까?

Trident 설치 프로그램은 Kubernetes 클러스터의 PersistentVolume, PersistentVolumeClaim, StorageClass 및 Secret 리소스에 대한 특정 액세스 권한을 가진 Kubernetes ClusterRole을 생성합니다. 을 참조하십시오 ["여기"](#).

설치에 Astra Trident가 사용하는 정확한 매니페스트 파일을 로컬로 생성할 수 있습니까?

필요한 경우 설치에 Astra Trident가 사용하는 정확한 매니페스트 파일을 로컬에서 생성하고 수정할 수 있습니다. 을 참조하십시오 ["여기"](#).

두 개의 개별 Kubernetes 클러스터를 위한 두 개의 개별 Astra Trident 인스턴스에 동일한 ONTAP 백엔드 SVM을 공유할 수 있습니까?

권장되지 않지만 두 개의 Astra Trident 인스턴스에 동일한 백엔드 SVM을 사용할 수 있습니다. 설치 중에 각 인스턴스의 고유한 볼륨 이름을 지정하거나 고유한 볼륨 이름을 지정하십시오 `StoragePrefix` 매개 변수 을 선택합니다 `setup/backend.json` 파일. 이는 동일한 FlexVol이 두 인스턴스에 모두 사용되지 않도록 하기 위한 것입니다.

ContainerLinux(이전의 CoreOS) 아래에 Astra Trident를 설치할 수 있습니까?

Astra Trident는 Kubernetes Pod로, Kubernetes를 실행 중인 모든 위치에 설치할 수 있습니다.

NetApp Cloud Volumes ONTAP에서 Astra Trident를 사용할 수 있습니까?

예. Astra Trident는 AWS, Google Cloud 및 Azure에서 지원됩니다.

Astra Trident가 Cloud Volumes Services와 작동합니까?

예, Astra Trident는 Azure의 Azure NetApp Files 서비스와 GCP의 Cloud Volumes Service를 지원합니다.

문제 해결 및 지원

NetApp은 Astra Trident를 지원합니까?

Astra Trident는 오픈 소스이며 무료로 제공되지만, NetApp 백엔드가 지원되는 경우 NetApp은 이를 완벽하게 지원합니다.

지원 케이스를 어떻게 제기합니까?

지원 케이스를 제기하려면 다음 중 하나를 수행합니다.

1. 지원 어카운트 매니저에게 연락하여 티켓을 발급하는 데 도움을 받으십시오.
2. 에 연락하여 지원 케이스를 제출하십시오 "[NetApp 지원](#)".

지원 로그 번들을 생성하려면 어떻게 해야 합니까?

을 실행하여 지원 번들을 생성할 수 있습니다 `tridentctl logs -a`. 번들에 캡처된 로그 외에 kubelet 로그를 캡처하여 Kubernetes 측의 마운트 문제를 진단합니다. 쿠벨릿 로그를 얻는 지침은 Kubernetes 설치 방법에 따라 다릅니다.

새 기능에 대한 요청을 제기해야 하는 경우 어떻게 해야 합니까?

에서 문제를 만듭니다 "[Astra Trident GitHub를 참조하십시오](#)" 제목에 * RFE * 를 기재하고 문제에 대한 설명을 기재하십시오.

결함은 어디에서 제기합니까?

에서 문제를 만듭니다 "[Astra Trident GitHub를 참조하십시오](#)". 문제와 관련된 모든 필수 정보와 로그를 포함해야 합니다.

Astra Trident에 대한 간단한 질문을 하면 어떻게 됩니까? 커뮤니티나 포럼이 있습니까?

질문, 문제 또는 요청이 있을 경우 Astra를 통해 문의해 주십시오 "[불화 채널](#)" 또는 GitHub 를 선택합니다.

스토리지 시스템의 암호가 변경되었고 Astra Trident가 더 이상 작동하지 않습니다. 어떻게 복구합니까?

로 백엔드 암호를 업데이트합니다 `tridentctl update backend myBackend -f`
</path/to_new_backend.json> -n trident. 대치 myBackend 백엔드 이름을 포함하는 예에서 및 입니다
`/path/to_new_backend.json 올바른 경로를 사용하여 backend.json 파일.

Astra Trident에서 내 Kubernetes 노드를 찾을 수 없습니다. 이 문제를 해결하려면 어떻게 합니까?

Astra Trident가 Kubernetes 노드를 찾을 수 없는 두 가지 시나리오가 있을 수 있습니다. Kubernetes의 네트워킹 문제 또는 DNS 문제 때문일 수 있습니다. 각 Kubernetes 노드에서 실행되는 Trident 노드 데모는 Trident 컨트롤러와 통신하여 노드를 Trident에 등록할 수 있어야 합니다. Astra Trident를 설치한 후 네트워킹 변경이 발생하면 클러스터에 추가된 새 Kubernetes 노드에서만 이 문제가 발생합니다.

Trident POD가 제거되면 데이터를 손실합니까?

Trident POD를 제거할 경우 데이터가 손실되지 않습니다. Trident의 메타데이터는 CRD 객체에 저장됩니다. Trident에서 프로비저닝한 모든 PVS가 정상적으로 작동합니다.

Astra Trident를 업그레이드합니다

이전 버전에서 새 버전으로 직접 업그레이드할 수 있습니까(일부 버전을 건너뛰는 경우)?

NetApp은 하나의 주요 릴리즈에서 바로 다음 주요 릴리즈로 Astra Trident를 업그레이드할 수 있도록 지원합니다. 버전 18.xx에서 19.xx, 19.xx에서 20.xx로 업그레이드할 수 있습니다. 운영 구축 전에 연구소에서 업그레이드를 테스트해야 합니다.

Trident를 이전 릴리즈로 다운그레이드할 수 있습니까?

다운그레이드를 원할 경우 다양한 요소를 평가해야 합니다. 을 참조하십시오 ["다운그레이드 섹션을 참조하십시오"](#).

백엔드 및 볼륨 관리

ONTAP 백엔드 정의 파일에서 관리 및 데이터 LIF를 모두 정의해야 합니까?

관리 LIF는 필수입니다. 데이터 LIF는 다양합니다.

- ONTAP SAN: iSCSI에 대해 지정하지 마십시오. Astra Trident가 사용합니다 ["ONTAP 선택적 LUN 맵"](#) 다중 경로 세션을 설정하는 데 필요한 iSCSI LIF를 검색하려면 경고 발생 시 `dataLIF` 명시적으로 정의됩니다. 을 참조하십시오 ["ONTAP SAN 구성 옵션 및 예"](#) 를 참조하십시오.
- ONTAP NAS: 을 지정하는 것이 좋습니다 `dataLIF`. 제공되지 않는 경우 Astra Trident는 SVM에서 데이터 LIF를 가져옵니다. NFS 마운트 작업에 사용할 FQDN(정규화된 도메인 이름)을 지정하면 여러 데이터 LIF에서 로드 밸런싱을 위해 라운드 로빈 DNS를 생성할 수 있습니다. 을 참조하십시오 ["ONTAP NAS 구성 옵션 및 예"](#) 를 참조하십시오

Astra Trident에서 ONTAP 백엔드에 대한 CHAP를 구성할 수 있습니까?

예. 20.04부터 Astra Trident는 ONTAP 백엔드에 대한 양방향 CHAP를 지원합니다. 이 작업을 수행하려면 설정이 필요합니다 `useCHAP=true` 백엔드 구성

Astra Trident를 사용하여 익스포트 정책을 관리하려면 어떻게 해야 합니까?

Astra Trident는 버전 20.04 이상에서 내보내기 정책을 동적으로 생성하고 관리할 수 있습니다. 따라서 스토리지 관리자는 백엔드 구성에서 하나 이상의 CIDR 블록을 제공할 수 있으며, 이러한 범위에 속하는 Trident 추가 노드 IP를 생성한 익스포트 정책에 추가할 수 있습니다. 이러한 방식으로 Astra Trident는 주어진 CIDR 내에 IP가 있는 노드의 규칙 추가 및 삭제를 자동으로 관리합니다. 이 기능을 사용하려면 CSI Trident가 필요합니다.

DataLIF에 포트를 지정할 수 있습니까?

Astra Trident 19.01 이후 버전에서는 DataLIF에 포트를 지정할 수 있습니다. 에서 구성합니다 `backend.json` 파일 형식 `"managementLIF": <ip address>:<port>`. 예를 들어 관리 LIF의 IP 주소가 192.0.2.1이고 포트가 1000인 경우 를 구성합니다 `"managementLIF": "192.0.2.1:1000"`.

관리 및 데이터 LIF에 IPv6 주소를 사용할 수 있습니까?

Astra Trident는 다음에 대한 IPv6 주소 정의를 지원합니다.

- managementLIF 및 dataLIF ONTAP NAS 백엔드의 경우
- managementLIF ONTAP SAN 백엔드의 경우 지정할 수 없습니다 dataLIF ONTAP SAN 백엔드에서

을 사용하여 Astra Trident를 설치해야 합니다 `--use-ipv6` IPv6를 통해 작동할 플래그입니다.

백엔드에서 관리 LIF를 업데이트할 수 있습니까?

예, 를 사용하여 백엔드 관리 LIF를 업데이트할 수 있습니다 `tridentctl update backend` 명령.

백엔드에서 데이터 LIF를 업데이트할 수 있습니까?

에서 데이터 LIF를 업데이트할 수 있습니다 `ontap-nas` 및 `ontap-nas-economy` 만 해당.

Kubernetes용 Astra Trident에서 여러 개의 백엔드를 생성할 수 있습니까?

Astra Trident는 동일한 드라이버나 다른 드라이버를 사용하여 동시에 많은 백엔드를 지원할 수 있습니다.

Astra Trident는 백엔드 자격 증명을 어떻게 저장합니까?

Astra Trident는 백엔드 자격 증명을 Kubernetes Secrets로 저장합니다.

Astra Trident는 특정 백엔드를 어떻게 선택합니까?

백엔드 속성을 사용하여 클래스에 적합한 풀을 자동으로 선택할 수 없는 경우 를 참조하십시오 `storagePools` 및 `additionalStoragePools` 매개 변수는 특정 풀 세트를 선택하는 데 사용됩니다.

Astra Trident가 특정 백엔드에서 프로비저닝하지 않도록 하려면 어떻게 해야 합니까?

를 클릭합니다 `excludeStoragePools` 매개 변수는 Astra Trident가 프로비저닝에 사용할 풀 세트를 필터링하고 일치하는 풀을 제거하는 데 사용됩니다.

동일한 종류의 백엔드가 여러 개 있는 경우 Astra Trident는 어떤 백엔드를 사용할 것인지 어떻게 선택할 수 있습니까?

동일한 유형의 백엔드가 여러 개 구성되어 있는 경우 Astra Trident는 에 있는 매개 변수를 기반으로 적절한 백엔드를 선택합니다 `StorageClass` 및 `PersistentVolumeClaim`. 예를 들어, 여러 ONTAP-NAS 드라이버 백엔드가 있을 경우 Astra Trident는 의 매개 변수와 일치시키려고 합니다 `StorageClass` 및 `PersistentVolumeClaim` 에 나와 있는 요구 사항을 제공할 수 있는 백엔드를 결합 및 일치시킵니다 `StorageClass` 및 `PersistentVolumeClaim`. 요청과 일치하는 백엔드가 여러 개 있는 경우, Astra Trident는 임의의 백엔드 중 하나를 선택합니다.

Astra Trident가 Element/SolidFire를 사용하는 양방향 CHAP를 지원합니까?

예.

Astra Trident는 ONTAP 볼륨에 qtree를 어떻게 배포합니까? 단일 볼륨에 몇 개의 qtree를 구축할 수 있습니까?

를 클릭합니다 `ontap-nas-economy` 드라이버는 동일한 FlexVol에서 최대 200개의 qtree(50~300 사이에서 구성 가능), 클러스터 노드당 100,000 qtree, 클러스터당 2.4M qtree를 생성합니다. 새 을 입력할 때 `PersistentVolumeClaim` 이코노미 운전자가 서비스를 제공하는 경우 운전자는 새 Qtree에 서비스를 제공할 수 있는 FlexVol이 이미 있는지 확인합니다. Qtree를 처리할 수 있는 FlexVol이 없으면 새 FlexVol이 생성됩니다.

ONTAP NAS에 프로비저닝된 볼륨에 대해 Unix 권한을 설정하려면 어떻게 해야 합니까?

백엔드 정의 파일에 매개 변수를 설정하여 Astra Trident가 프로비저닝한 볼륨에 대해 Unix 권한을 설정할 수 있습니다.

볼륨을 프로비저닝하는 동안 명시적 ONTAP NFS 마운트 옵션 세트를 구성하려면 어떻게 합니까?

기본적으로 Astra Trident는 Kubernetes의 마운트 옵션을 아무 값으로도 설정하지 않습니다. Kubernetes 스토리지 클래스에서 마운트 옵션을 지정하려면 지정된 예제를 따르십시오 ["여기"](#).

프로비저닝된 볼륨을 특정 익스포트 정책으로 설정하려면 어떻게 해야 합니까?

적절한 호스트가 볼륨에 액세스할 수 있도록 하려면 를 사용합니다 `exportPolicy` 백엔드 정의 파일에 구성된 매개 변수입니다.

ONTAP가 있는 Astra Trident를 통해 볼륨 암호화를 설정하려면 어떻게 해야 합니까?

백엔드 정의 파일의 암호화 매개 변수를 사용하여 Trident에서 프로비저닝한 볼륨에 대한 암호화를 설정할 수 있습니다. 자세한 내용은 다음을 참조하십시오. ["Astra Trident가 NVE 및 NAE와 연동되는 방식"](#)

Astra Trident를 통해 ONTAP에 대한 QoS를 구축하는 가장 좋은 방법은 무엇입니까?

사용 `StorageClasses` ONTAP에 대한 QoS를 구현합니다.

Astra Trident를 통해 씬 또는 일반 프로비저닝을 지정하려면 어떻게 해야 합니까?

ONTAP 드라이버는 씬 또는 일반 프로비저닝을 지원합니다. ONTAP 드라이버는 기본적으로 씬 프로비저닝입니다. 일반 프로비저닝이 필요한 경우 백엔드 정의 파일 또는 을 구성해야 합니다 `StorageClass`. 둘 다 구성된 경우 `StorageClass` 이(가) 우선합니다. ONTAP에 대해 다음을 구성합니다.

1. 커짐 `StorageClass``를 설정합니다 ``provisioningType` 속성을 굵은선 로 지정합니다.
2. 백엔드 정의 파일에서 설정을 통해 일반 볼륨을 설정합니다 `backend spaceReserve parameter` 볼륨.

실수로 PVC를 삭제한 경우에도 사용 중인 볼륨이 삭제되지 않도록 하려면 어떻게 해야 합니까?

PVC 보호는 버전 1.10부터 Kubernetes에서 자동으로 활성화됩니다.

Astra Trident에서 만든 NFS PVC를 늘릴 수 있습니까?

예. Astra Trident에서 만든 PVC를 확장할 수 있습니다. 볼륨 자동 증가 기능은 Trident에 적용되지 않는 ONTAP 기능입니다.

Astra Trident 외부에서 생성된 볼륨이 있는 경우 Astra Trident로 가져올 수 있습니까?

19.04부터는 볼륨 가져오기 기능을 사용하여 Kubernetes에 볼륨을 가져올 수 있습니다.

SnapMirror 데이터 보호(DP) 또는 오프라인 모드일 때 볼륨을 가져올 수 있습니까?

외부 볼륨이 DP 모드이거나 오프라인인 경우 볼륨 가져오기가 실패합니다. 다음과 같은 오류 메시지가 나타납니다.

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

Astra Trident에서 만든 iSCSI PVC를 확장할 수 있습니까?

Trident 19.10은 CSI Provisioner를 사용하여 iSCSI PVS를 확장할 수 있도록 지원합니다.

리소스 할당량은 NetApp 클러스터로 어떻게 변환됩니까?

NetApp 스토리지의 용량이 있는 경우 Kubernetes 스토리지 리소스 할당량이 작동합니다. 용량 부족으로 인해 NetApp 스토리지가 Kubernetes 할당량 설정을 적용할 수 없을 경우 Astra Trident가 프로비저닝하려고 하지만 오류를 해결합니다.

Astra Trident를 사용하여 볼륨 스냅샷을 생성할 수 있습니까?

예. Astra Trident는 스냅샷에서 필요 시 볼륨 스냅샷 및 영구 볼륨 생성을 지원합니다. 스냅샷에서 PVS를 생성하려면 를 확인하십시오 VolumeSnapshotDataSource 기능 게이트가 활성화되었습니다.

Astra Trident 볼륨 스냅샷을 지원하는 드라이버는 무엇입니까?

현재 NetApp에서는 온디맨드 스냅샷 지원을 이용할 수 있습니다 ontap-nas, ontap-nas-flexgroup, ontap-san, ontap-san-economy, solidfire-san, gcp-cvs, 및 azure-netapp-files 백엔드 드라이버.

ONTAP를 사용하여 Astra Trident가 프로비저닝한 볼륨의 스냅샷 백업을 어떻게 생성합니까?

이 기능은 에서 사용할 수 있습니다 ontap-nas, ontap-san, 및 ontap-nas-flexgroup 드라이버. 을 지정할 수도 있습니다 snapshotPolicy 의 경우 ontap-san-economy FlexVol 레벨의 운전자.

이 기능은 에서도 사용할 수 있습니다 ontap-nas-economy 운전기사는 qtree 레벨의 정밀도가 아니라 FlexVol 레벨의 정밀도로 표시됩니다. Astra Trident에서 프로비저닝한 볼륨을 스냅샷하는 기능을 사용하려면 backend parameter 옵션을 설정합니다 snapshotPolicy ONTAP 백엔드에 정의된 대로 원하는 스냅샷 정책으로 이동합니다. 스토리지 컨트롤러에서 생성한 스냅샷은 Astra Trident에서 알 수 없습니다.

Astra Trident를 통해 프로비저닝된 볼륨에 대한 스냅샷 예약 비율을 설정할 수 있습니까?

예, 를 설정하여 Astra Trident를 통해 스냅샷 복사본을 저장할 디스크 공간의 특정 비율을 예약할 수 있습니다 snapshotReserve 백엔드 정의의 파일의 속성입니다. 을(를) 구성한 경우 snapshotPolicy 및

snapshotReserve 백엔드 정의 파일에서 스냅샷 예약 비율은 에 따라 설정됩니다 snapshotReserve 백엔드 파일에 언급된 비율입니다. 를 누릅니다 snapshotReserve 백분율 수치는 언급되지 않습니다. 기본적으로 ONTAP는 스냅샷 예비 공간 비율을 5로 사용합니다. 를 누릅니다 snapshotPolicy 옵션이 없으므로 설정되고 스냅샷 예비 공간 백분율이 0으로 설정됩니다.

볼륨 스냅샷 디렉토리에 직접 액세스하고 파일을 복사할 수 있습니까?

예. Trident에서 를 설정하여 프로비저닝된 볼륨의 스냅샷 디렉토리에 액세스할 수 있습니다 snapshotDir 백엔드 정의 파일의 매개 변수입니다.

Astra Trident를 통해 볼륨에 대해 SnapMirror를 설정할 수 있습니까?

현재 ONTAP CLI 또는 OnCommand System Manager를 사용하여 외부에서 SnapMirror를 설정해야 합니다.

영구 볼륨을 특정 ONTAP 스냅샷으로 복원하려면 어떻게 합니까?

ONTAP 스냅샷에 볼륨을 복원하려면 다음 단계를 수행하십시오.

1. 영구 볼륨을 사용하는 응용 프로그램 포드를 중지합니다.
2. ONTAP CLI 또는 OnCommand System Manager를 통해 필요한 스냅샷으로 되돌립니다.
3. 응용 프로그램 포드를 다시 시작합니다.

Trident가 로드 공유 미러가 구성된 SVM에서 볼륨을 프로비저닝할 수 있습니까?

NFS를 통해 데이터를 제공하는 SVM의 루트 볼륨에 로드 공유 미러를 생성할 수 있습니다. ONTAP는 Trident에서 생성한 볼륨의 로드 공유 미러를 자동으로 업데이트합니다. 이로 인해 볼륨 마운팅이 지연될 수 있습니다. Trident를 사용하여 여러 볼륨을 생성할 경우 볼륨 프로비저닝은 ONTAP에서 로드 공유 미러 업데이트에 따라 달라집니다.

각 고객/테넌트에 대해 스토리지 클래스 사용을 어떻게 분리할 수 있습니까?

Kubernetes에서는 네임스페이스의 스토리지 클래스를 허용하지 않습니다. 그러나 Kubernetes를 사용하여 네임스페이스당 사용되는 스토리지 리소스 할당량을 사용하여 네임스페이스당 특정 스토리지 클래스의 사용을 제한할 수 있습니다. 특정 스토리지에 대한 특정 네임스페이스 액세스를 거부하려면 해당 스토리지 클래스에 대한 리소스 할당량을 0으로 설정합니다.

지원

Astra Trident는 공식적으로 지원되는 NetApp 프로젝트입니다. 표준 메커니즘을 사용하여 NetApp에 연락하면 필요한 엔터프라이즈급 지원을 받을 수 있습니다.

또한 Astra의 컨테이너 사용자(Astra Trident 개발자 포함)로 구성된 활발한 공용 커뮤니티가 있습니다 ["불화 채널"](#). 프로젝트에 대한 일반적인 질문을 하고 비슷한 생각을 가진 동료와 관련 주제를 논의할 수 있는 좋은 장소입니다.

문제 해결

Astra Trident를 설치 및 사용하는 동안 발생할 수 있는 문제를 해결하려면 여기에 제공된 포인터를 사용하십시오.



Astra Trident에 대한 도움을 받으려면 을 사용하여 지원 번들을 생성합니다 `tridentctl logs -a -n trident` 로 전송합니다 NetApp Support <Getting Help>.



문제 해결 문서의 전체 목록을 보려면 을 참조하십시오 "[NetApp Knowledgebase\(로그인 필요\)](#)". Astra와 관련된 문제 해결에 대한 정보도 찾을 수 있습니다 "[여기](#)".

일반 문제 해결

- Trident 포드가 제대로 표시되지 않는 경우(예: Trident 포드가 에 걸린 경우 ContainerCreating 2개 미만의 준비된 컨테이너로 단계를 진행). 를 실행합니다 `kubectl -n trident describe deployment trident` 및 `kubectl -n trident describe pod trident--**` 더 많은 통찰력을 제공할 수 있습니다. kubelet 로그 얻기(예: `VIA journalctl -xeu kubelet`) 또한 도움이 될 수 있습니다.
- Trident 로그에 충분한 정보가 없으면 를 전달하여 Trident에 대한 디버그 모드를 사용하도록 설정할 수 있습니다 `-d` 설치 옵션에 따라 설치 매개 변수에 플래그를 지정합니다.

그런 다음 디버그가 를 사용하여 설정되었는지 확인합니다 `./tridentctl logs -n trident` 검색 중입니다 `level=debug msg` 를 로그로 이동합니다.

운영자와 함께 설치됩니다

```
kubectl patch torc trident -n <namespace> --type=merge -p '{"spec":{"debug":true}}'
```

그러면 모든 Trident 포드가 다시 시작됩니다. 이 작업은 몇 초 정도 걸릴 수 있습니다. 의 출력에서 '나이' 열을 관찰하여 이를 확인할 수 있습니다 `kubectl get pod -n trident`.

Astra Trident 20.07 및 20.10용 `tprov` 대신 `torc`.

헬름과 함께 설치

```
helm upgrade <name> trident-operator-21.07.1-custom.tgz --set tridentDebug=true`
```

`tridentctl`과 함께 설치됩니다

```
./tridentctl uninstall -n trident  
./tridentctl install -d -n trident
```

- 을 포함하여 각 백엔드에 대한 디버그 로그를 얻을 수도 있습니다 `debugTraceFlags` 백엔드 정의에서 예를 들어, 을 포함합니다 `debugTraceFlags: {"api":true, "method":true,}` Trident 로그에서 API 호출 및

메서드 트레이스된 정보를 얻습니다. 기존 백엔드에 가 있을 수 있습니다 `debugTraceFlags` 로 구성됩니다 `tridentctl backend update`.

- RedHat CoreOS를 사용할 때는 다음을 확인하십시오 `iscsid` 는 작업자 노드에서 활성화되고 기본적으로 시작됩니다. 이 작업은 `OpenShift MachineConfigs`를 사용하거나 점화 템플릿을 수정하여 수행할 수 있습니다.
- 에서 Trident를 사용할 때 일반적으로 발생할 수 있는 문제입니다 **"Azure NetApp Files"** 테넌트 및 클라이언트 암호가 권한이 부족한 앱 등록에서 나오는 경우 입니다. Trident 요구사항의 전체 목록은 를 참조하십시오 **"Azure NetApp Files"** 구성.
- 컨테이너에 PV를 장착하는 데 문제가 있는 경우, 반드시 확인하십시오 `rpcbind` 가 설치되어 실행 중입니다. 호스트 OS에 필요한 패키지 관리자를 사용하여 확인합니다 `rpcbind` 가 실행 중입니다. 의 상태를 확인할 수 있습니다 `rpcbind` 를 실행하여 서비스를 제공합니다 `systemctl status rpcbind` 또는 그와 동등한 제품이라는 것입니다.
- Trident 백엔드가 에 포함되어 있다고 보고하는 경우 `failed State`: 이전에 작업을 수행했지만 백엔드와 연결된 SVM/관리 자격 증명을 변경했기 때문일 수 있습니다. 을 사용하여 백엔드 정보를 업데이트합니다 `tridentctl update backend` 또는 Trident Pod를 바운드하면 이 문제가 해결됩니다.
- 베타 볼륨 스냅샷을 사용하도록 Kubernetes 클러스터 및/또는 Trident를 업그레이드할 경우 기존의 모든 알파 스냅샷 CRS가 완전히 제거되었는지 확인합니다. 그런 다음 를 사용할 수 있습니다 `tridentctl obliviante alpha-snapshot-crd` 알파 스냅샷 CRD를 삭제하는 명령입니다. 을 참조하십시오 **"블로그입니다"** 알파 스냅샷 마이그레이션 단계를 이해합니다.
- Docker를 컨테이너 런타임으로 사용하여 Trident를 설치할 때 권한 문제가 발생하면 를 사용하여 Trident 설치를 시도하십시오 `--in cluster=false` 깃발. 이렇게 하면 설치 프로그램 포드가 사용되지 않으며 로 인해 발생하는 권한 문제가 발생하지 않습니다 `trident-installer` 사용자.
- 를 사용합니다 `uninstall parameter <Uninstalling Trident>` 실행 실패 후 정리. 기본적으로 이 스크립트는 Trident에서 만든 CRD를 제거하지 않으므로 실행 중인 구축에서도 안전하게 제거한 후 다시 설치할 수 있습니다.
- 이전 버전의 Trident로 다운그레이드하려는 경우 먼저 을 실행합니다 `tridentctl uninstall Trident` 를 제거하는 명령입니다. 원하는 를 다운로드합니다 **"Trident 버전"** 를 사용하여 를 설치합니다 `tridentctl install` 명령. 새로 생성된 PVS가 없고 기존 PVS/백엔드/스토리지 클래스가 변경되지 않은 경우에만 다운그레이드를 고려하십시오. 이제 Trident는 CRD를 사용하여 상태를 유지하므로 생성된 모든 스토리지 요소 (백엔드, 스토리지 클래스, PVS 및 볼륨 스냅샷)에는 가 있습니다 `associated CRD objects <Kubernetes CustomResourceDefinition Objects>` 이전의 설치된 Trident 버전에서 사용한 PV에 기록된 데이터 대신 * 이전 버전으로 되돌릴 때 새로 생성된 PVS를 사용할 수 없습니다. * * 다운그레이드 시 백엔드, PVS, 스토리지 클래스 및 볼륨 스냅샷(생성/업데이트/삭제)과 같은 객체에 대한 변경 사항은 Trident에 표시되지 않습니다 *. 설치된 Trident의 이전 버전에서 사용된 PV는 여전히 Trident에 표시됩니다. 이전 버전으로 돌아가면 업그레이드되지 않은 경우 이전 릴리즈를 사용하여 이미 생성된 PVS에 대한 액세스가 중단되지 않습니다.
- Trident를 완전히 제거하려면 를 실행합니다 `tridentctl obliviante crd` 명령. 그러면 모든 CRD 객체가 제거되고 CRD의 정의가 해제됩니다. Trident는 이미 프로비저닝한 PVS를 더 이상 관리하지 않습니다.



이 후 Trident를 처음부터 다시 구성해야 합니다.

- 성공적으로 설치한 후 PVC가 에 고착된 경우 `Pending` 단계, 실행 중 `kubect1 describe pvc Trident`가 이 PVC에 대한 PV를 프로비저닝하지 못한 이유에 대한 추가 정보를 제공할 수 있습니다.

연산자를 사용하여 실패한 Trident 배포 문제 해결

연산자를 사용하여 Trident를 배포하는 경우 의 상태입니다 `TridentOrchestrator` 변경 시작 `Installing` 를 선택합니다 `Installed`. 을(를) 관찰하면 `Failed` 상태, 운용자가 자체적으로 복구할 수 없는 경우, 다음 명령어를 실행해 운용자의 로그를 확인해야 한다.

```
tridentctl logs -l trident-operator
```

삼원 운영자 컨테이너의 로그를 뒤로하면 문제가 있는 위치를 가리킬 수 있습니다. 예를 들어, 이러한 문제 중 하나는 Airgapped 환경의 업스트림 등록부에서 필요한 컨테이너 이미지를 가져올 수 없는 것일 수 있습니다.

Trident를 설치하지 못한 이유를 알아보려면 를 살펴보십시오 TridentOrchestrator 상태.

```
kubectl describe torc trident-2
Name:          trident-2
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:      <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                  Trident is bound to another CR 'trident'
  Namespace:                trident-2
  Status:                    Error
  Version:
Events:
  Type      Reason  Age          From              Message
  ----      -
  Warning   Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
  is bound to another CR 'trident'
```

이 오류는 가 이미 있음을 나타냅니다 TridentOrchestrator`Trident를 설치하는 데 사용했습니다. 각 Kubernetes 클러스터에는 Trident의 인스턴스가 하나만 있을 수 있으므로 운영자는 언제든지 활성 인스턴스가 하나만 있는지 확인할 수 있습니다 `TridentOrchestrator 만들 수 있습니다.

또한 Trident Pod의 상태를 관찰하면 무언가 잘못되었음을 나타내는 경우가 많습니다.


```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
trident-csi-4p5kq 5m18s	1/2	ImagePullBackOff	0
trident-csi-6f45bfd8b6-vfrkw 5m19s	4/5	ImagePullBackOff	0
trident-csi-9q5xc 5m18s	1/2	ImagePullBackOff	0
trident-csi-9v95z 5m18s	1/2	ImagePullBackOff	0
trident-operator-766f7b8658-ldzsv 8m17s	1/1	Running	0

하나 이상의 컨테이너 이미지를 가져오지 않았기 때문에 포드를 완전히 초기화할 수 없다는 것을 분명히 알 수 있습니다.

문제를 해결하려면 을 편집해야 합니다 TridentOrchestrator 있습니다. 또는 을(를) 삭제할 수도 있습니다 `TridentOrchestrator` 수정 및 정확한 정의를 사용하여 새 정의를 만듭니다.

를 사용하여 Trident 배포가 성공하지 못한 경우 문제 해결

tridentctl

무엇이 잘못되었는지 알아내기 위해 를 사용하여 설치 프로그램을 다시 실행할 수 있습니다 -d 디버그 모드를 설정하고 문제가 무엇인지 이해하는 데 도움이 되는 인수:

```
./tridentctl install -n trident -d
```

문제를 해결한 후 다음과 같이 설치를 정리한 다음 를 실행할 수 있습니다 tridentctl install 다시 명령:

```
./tridentctl uninstall -n trident  
INFO Deleted Trident deployment.  
INFO Deleted cluster role binding.  
INFO Deleted cluster role.  
INFO Deleted service account.  
INFO Removed Trident user from security context constraint.  
INFO Trident uninstallation succeeded.
```

모범 사례 및 권장사항

구축

Astra Trident를 배포할 때 여기에 나열된 권장 사항을 사용하십시오.

전용 네임스페이스에 구축

"네임스페이스" 서로 다른 애플리케이션 간의 관리 분리를 제공하며 리소스 공유의 장벽입니다. 예를 들어, 한 네임스페이스의 PVC는 다른 네임스페이스에서 사용할 수 없습니다. Astra Trident는 Kubernetes 클러스터의 모든 네임스페이스에 PV 리소스를 제공하고, 결과적으로 권한이 상승된 서비스 계정을 활용합니다.

또한 Trident Pod에 액세스하면 사용자가 스토리지 시스템 자격 증명 및 기타 중요한 정보에 액세스할 수 있습니다. 애플리케이션 사용자 및 관리 애플리케이션에서 Trident 객체 정의 또는 POD 자체에 액세스할 수 없도록 하는 것이 중요합니다.

할당량 및 범위 제한을 사용하여 스토리지 사용량을 제어할 수 있습니다

Kubernetes에는 2가지 기능이 있으며, 이 기능을 조합하여 애플리케이션의 리소스 사용을 제한하는 강력한 메커니즘을 제공합니다. 를 클릭합니다 "[스토리지 할당량 메커니즘](#)" 관리자가 네임스페이스별로 글로벌 및 스토리지 클래스별, 용량 및 오브젝트 수 사용 제한을 구현할 수 있도록 지원 또한 를 사용합니다 "[범위 제한](#)" 요청이 프로비저닝 사용자에게 전달되기 전에 PVC 요청이 최소값 및 최대값 내에 있는지 확인합니다.

이러한 값은 네임스페이스 단위로 정의됩니다. 즉, 각 네임스페이스에는 리소스 요구 사항에 맞는 값이 정의되어 있어야 합니다. 에 대한 자세한 내용은 여기 를 참조하십시오 "[할당량을 활용하는 방법](#)".

스토리지 구성

NetApp 포트폴리오의 각 스토리지 플랫폼은 컨테이너식으로 애플리케이션에 이점을 제공하는 고유한 기능을 제공합니다.

플랫폼 개요

Trident는 ONTAP 및 요소와 함께 작동합니다. 모든 애플리케이션과 시나리오에 적합한 플랫폼이 한 개 있는 것은 아니지만, 플랫폼을 선택할 때 애플리케이션과 장치를 관리하는 팀의 요구 사항을 고려해야 합니다.

활용 중인 프로토콜을 사용하여 호스트 운영 체제의 기존 모범 사례를 따라야 합니다. 필요에 따라 특정 애플리케이션에 맞게 스토리지를 최적화할 수 있도록 백엔드, 스토리지 클래스 및 PVC 설정과 함께 사용 가능한 경우 애플리케이션 Best Practice를 통합하는 것을 고려할 수 있습니다.

ONTAP 및 Cloud Volumes ONTAP 모범 사례

Trident를 위한 ONTAP 및 Cloud Volumes ONTAP를 구성하기 위한 모범 사례에 대해 알아보십시오.

다음 권장 사항은 Trident에서 동적으로 프로비저닝되는 볼륨을 사용하는 컨테이너식 워크로드에 대한 ONTAP 구성 지침입니다. 각 항목을 고려하여 작업 환경의 적절성을 판단해야 합니다.

Trident 전용 SVM을 사용하십시오

SVM(스토리지 가상 시스템)은 ONTAP 시스템의 테넌트 간에 격리하고 관리를 제공합니다. SVM을 애플리케이션 전용으로 사용하면 권한을 위임하고 리소스 사용을 제한하는 모범 사례를 적용할 수 있습니다.

SVM 관리를 위해 몇 가지 옵션을 사용할 수 있습니다.

- 백엔드 구성에서 클러스터 관리 인터페이스를 적절한 자격 증명과 함께 제공하고 SVM 이름을 지정합니다.
- ONTAP System Manager 또는 CLI를 사용하여 SVM을 위한 전용 관리 인터페이스를 생성합니다.
- 관리 역할을 NFS 데이터 인터페이스와 공유합니다.

각 경우 인터페이스가 DNS에 있어야 하며, Trident를 구성할 때 DNS 이름을 사용해야 합니다. 이렇게 하면 네트워크 ID 보존을 사용하지 않고 SVM-DR과 같은 일부 DR 시나리오를 간편하게 수행할 수 있습니다.

SVM에 전용 또는 공유 관리 LIF를 사용하는 것이 더 이상 선호되지 않지만, 선택한 접근 방식에 맞게 네트워크 보안 정책을 조정해야 합니다. 관리 LIF는 DNS를 통해 액세스할 수 있어야 하며, 유연성을 극대화해야 합니다 "SVM-DR" Trident와 함께 사용합니다.

최대 볼륨 수를 제한합니다

ONTAP 스토리지 시스템의 최대 볼륨 수는 소프트웨어 버전과 하드웨어 플랫폼에 따라 다릅니다. 을 참조하십시오 ["NetApp Hardware Universe를 참조하십시오"](#) 정확한 제한을 결정하는 특정 플랫폼 및 ONTAP 버전. 볼륨 수가 소진되면 프로비저닝 작업이 Trident뿐 아니라 모든 스토리지 요청에 대해 실패합니다.

Trident `ontap-nas` 및 `ontap-san` 드라이버는 만들어진 각 Kubernetes PV(영구 볼륨)에 대해 FlexVolume을 프로비저닝합니다. 를 클릭합니다 `ontap-nas-economy` 드라이버는 200개의 PVS에 대해 약 1개의 FlexVolume을 생성합니다(50에서 300까지 구성 가능). 를 클릭합니다 `ontap-san-economy` 드라이버는 100개의 PVS에 대해 약 1개의 FlexVolume을 생성합니다(50에서 200까지 구성 가능). Trident가 스토리지 시스템에서 사용 가능한 모든 볼륨을 사용하지 않도록 하려면 SVM에 제한을 설정해야 합니다. 이 작업은 명령줄에서 수행할 수 있습니다.

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

의 값 `max-volumes` 환경에 따라 몇 가지 기준에 따라 다름:

- ONTAP 클러스터에 있는 기존 볼륨의 수입입니다
- 다른 애플리케이션에 대해 Trident 외부에 프로비저닝할 것으로 예상되는 볼륨 수입입니다
- Kubernetes 애플리케이션에서 사용할 것으로 예상되는 영구 볼륨의 수입입니다

를 클릭합니다 `max-volumes` value는 개별 ONTAP 노드가 아닌 ONTAP 클러스터의 모든 노드에 프로비저닝된 총 볼륨입니다. 결과적으로, ONTAP 클러스터 노드에 다른 노드보다 훨씬 더 많은 Trident 프로비저닝 볼륨이 있을 수 있는 몇 가지 조건이 발생할 수 있습니다.

예를 들어, 2노드 ONTAP 클러스터에는 최대 2000개의 FlexVolumes를 호스팅할 수 있는 기능이 있습니다. 최대 볼륨 수를 1250으로 설정하면 매우 적절합니다. 그러나, 단 인 경우 "애그리게이트" 한 노드에서 SVM에 할당하거나, 한 노드에서 할당된 애그리게이트는 용량 등으로 인해 프로비저닝할 수 없는 경우, 다른 노드는 프로비저닝된 모든 Trident 볼륨의 타겟이 됩니다. 즉, 보다 먼저 해당 노드에 대한 볼륨 제한에 도달할 수 있습니다 `max-volumes` 값에 도달하면 Trident와 해당 노드를 사용하는 기타 볼륨 작업에 모두 영향을 미칩니다. * 클러스터의 각 노드에서 애그리게이트가 동일한 수의 Trident가 사용하는 SVM에 할당되도록 하면 이러한 상황을 방지할 수 있습니다. *

Trident에서 생성한 볼륨의 최대 크기를 제한합니다

Trident에서 생성할 수 있는 볼륨의 최대 크기를 구성하려면 `limitVolumeSize` 매개 변수를 선택합니다 `backend.json` 정의.

스토리지 어레이에서 볼륨 크기를 제어하는 것 외에도 Kubernetes 기능을 활용해야 합니다.

양방향 CHAP를 사용하도록 Trident를 구성합니다

백엔드 정의에 CHAP 이니시에이터와 타겟 사용자 이름 및 암호를 지정하고 SVM에서 Trident가 CHAP를 사용하도록 설정할 수 있습니다. `useCHAP` 매개 변수 백엔드 구성에서 Trident는 CHAP를 사용하여 ONTAP 백엔드에 대한 iSCSI 연결을 인증합니다. 양방향 CHAP 지원은 Trident 20.04 이상에서 사용할 수 있습니다.

SVM QoS 정책을 생성하고 사용합니다

SVM에 적용되는 ONTAP QoS 정책을 활용하여 Trident에서 프로비저닝된 볼륨에서 사용할 수 있는 IOPS 수를 제한합니다. 그러면 에 도움이 됩니다 "**괴롭힘을 방지합니다**" 또는 Trident SVM 외부의 워크로드에 영향을 주지 않는 제어 컨테이너

몇 가지 단계로 SVM에 대한 QoS 정책을 생성할 수 있습니다. 가장 정확한 정보는 사용 중인 ONTAP 버전 설명서를 참조하십시오. 아래 예는 SVM에 사용 가능한 총 IOPS를 5000으로 제한하는 QoS 정책을 생성합니다.

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

또한 사용하는 ONTAP 버전에서 지원하는 경우에는 최소 QoS를 사용하여 컨테이너화된 워크로드에 대한 처리량을 보장하는 것을 고려할 수 있습니다. 적응형 QoS는 SVM 레벨 정책과 호환되지 않습니다.

컨테이너화된 워크로드 전용 IOPS 수는 다양한 측면에 따라 다릅니다. 그 밖의 다른 사항으로는 다음과 같은 것들이 있습니다.

- 기타 워크로드는 스토리지 어레이를 사용합니다. 스토리지 리소스를 활용하여 Kubernetes 구축과 관련되지 않은 다른 워크로드가 있는 경우, 해당 워크로드가 실수로 영향을 받지 않도록 주의해야 합니다.
- 컨테이너에서 실행 중인 예상 워크로드 IOPS 요구사항이 높은 워크로드를 컨테이너에서 실행할 경우 QoS 정책이 낮으면 잘못된 경험이 될 수 있습니다.

SVM 레벨에서 할당된 QoS 정책을 사용하면 동일한 IOPS 풀을 공유하는 SVM에 프로비저닝된 모든 볼륨이 생성된다는 점을 기억해야 합니다. 컨테이너화된 애플리케이션 중 하나 또는 그 수가 적은 경우 높은 IOPS 요구사항이 있으면 다른 컨테이너화된 워크로드에 문제가 될 수 있습니다. 이 경우 외부 자동화를 사용하여 볼륨당 QoS 정책을 할당하는 것을 고려할 수 있습니다.



ONTAP 버전이 9.8 이전인 경우 SVM *에만 QoS 정책 그룹을 할당해야 합니다.

Trident에 대한 QoS 정책 그룹을 생성합니다

QoS(서비스 품질)는 경쟁 워크로드로부터 주요 워크로드의 성능이 저하되지 않도록 보장합니다. ONTAP QoS 정책 그룹은 볼륨에 대한 QoS 옵션을 제공하고 사용자가 하나 이상의 워크로드에 대한 처리량 한도를 정의할 수 있도록 지원합니다. QoS에 대한 자세한 내용은 [를 참조하십시오 "QoS를 통해 처리량 보장"](#). 백엔드에서 또는 스토리지 풀에 QoS 정책 그룹을 지정할 수 있으며, 이러한 그룹은 해당 풀 또는 백엔드에서 생성된 각 볼륨에 적용됩니다.

ONTAP에는 기존 QoS 정책과 적응형 서비스 두 가지 QoS 정책 그룹이 있습니다. 기존 정책 그룹은 IOPS 단위로 최대 또는 최소 단위의 고정 처리량을 제공합니다. 적응형 QoS는 워크로드 크기에 따라 처리량을 자동으로 확장하므로 워크로드 크기에 따라 IOPS와 TB|GB의 비율을 유지합니다. 따라서 대규모 구축 환경에서 수백 또는 수천 개의 워크로드를 관리할 경우 상당한 이점이 있습니다.

QoS 정책 그룹을 생성할 때는 다음 사항을 고려하십시오.

- `를 설정해야 합니다 qosPolicy` 키를 누릅니다 `defaults` 백엔드 구성의 블록 다음 백엔드 구성 예를 참조하십시오.

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
- labels:
  performance: extreme
  defaults:
  adaptiveQosPolicy: extremely-adaptive-pg
- labels:
  performance: premium
  defaults:
  qosPolicy: premium-pg
```

- 각 볼륨이 정책 그룹에서 지정한 전체 처리량을 얻을 수 있도록 볼륨별로 정책 그룹을 적용해야 합니다. 공유 정책 그룹은 지원되지 않습니다.

QoS 정책 그룹에 대한 자세한 내용은 [을 참조하십시오 "ONTAP 9.8 QoS 명령"](#).

스토리지 리소스에 대한 액세스 권한을 **Kubernetes** 클러스터 구성원으로 제한합니다

Trident에서 생성한 NFS 볼륨 및 iSCSI LUN에 대한 액세스를 제한하는 것은 Kubernetes 구축을 위한 보안 환경의 중요한 구성요소입니다. 이렇게 하면 Kubernetes 클러스터의 일부가 아닌 호스트가 볼륨에 액세스하고 예기치 않게 데이터를 수정하는 것을 방지할 수 있습니다.

네임스페이스가 Kubernetes의 리소스에 대한 논리적 경계라는 것을 이해하는 것이 중요합니다. 동일한 네임스페이스의

리소스를 공유할 수 있다고 가정하지만, 특히 상호 네임스페이스 기능이 없다는 것이 중요합니다. 즉, PVS는 글로벌 객체이지만 PVC에 바인딩되면 동일한 네임스페이스에 있는 Pod에서만 액세스할 수 있습니다. * 적절한 경우 네임스페이스를 사용하여 구분을 제공하는 것이 중요합니다. *

Kubernetes 컨텍스트에서 데이터 보안과 관련하여 대부분의 조직은 컨테이너 내의 프로세스가 호스트에 마운트된 스토리지에 액세스할 수 있지만 컨테이너용 프로세스는 아닙니다. "네임스페이스" 이러한 유형의 손상을 방지하도록 설계되었습니다. 그러나 권한 있는 컨테이너에는 한 가지 예외가 있습니다.

권한 있는 컨테이너는 일반적인 것보다 훨씬 더 많은 호스트 수준 권한으로 실행되는 컨테이너입니다. 이러한 기능은 기본적으로 거부되지 않으므로 을 사용하여 기능을 사용하지 않도록 설정해야 합니다 "POD 보안 정책".

Kubernetes 및 외부 호스트 모두에서 액세스가 필요한 볼륨의 경우, Trident에서 관리하지 않고 관리자가 PV를 도입한 상태로 스토리지를 기존 방식으로 관리해야 합니다. 이렇게 하면 Kubernetes 및 외부 호스트의 연결이 모두 끊기고 볼륨을 더 이상 사용하지 않는 경우에만 스토리지 볼륨이 폐기됩니다. 또한, 맞춤형 익스포트 정책을 적용하여 Kubernetes 클러스터 노드 및 Kubernetes 클러스터 외부의 타겟 서버에서 액세스할 수 있습니다.

전용 인프라 노드(예: OpenShift) 또는 사용자 애플리케이션을 예약할 수 없는 다른 노드를 구축하는 경우, 별도의 익스포트 정책을 사용하여 스토리지 리소스에 대한 액세스를 더욱 제한해야 합니다. 여기에는 해당 인프라 노드에 배포된 서비스(예: OpenShift Metrics 및 Logging 서비스)에 대한 익스포트 정책과 비인프라 노드에 배포되는 표준 애플리케이션이 포함됩니다.

전용 익스포트 정책을 사용하십시오

Kubernetes 클러스터에 있는 노드에만 액세스할 수 있도록 각 백엔드에 대한 익스포트 정책이 있어야 합니다. Trident는 20.04 릴리스부터 내보내기 정책을 자동으로 생성하고 관리할 수 있습니다. 이러한 방법으로 Trident는 Kubernetes 클러스터의 노드에 프로비저닝되는 볼륨에 대한 액세스를 제한하고 노드 추가/삭제를 단순화합니다.

또는 수동으로 익스포트 정책을 생성하여 각 노드 액세스 요청을 처리하는 하나 이상의 익스포트 규칙으로 채울 수도 있습니다.

- 를 사용합니다 `vserver export-policy create ONTAP CLI` 명령을 사용하여 익스포트 정책을 생성합니다.
- 를 사용하여 익스포트 정책에 규칙을 추가합니다 `vserver export-policy rule create ONTAP CLI` 명령

이러한 명령을 실행하면 데이터에 액세스할 수 있는 Kubernetes 노드를 제한할 수 있습니다.

사용 안 함 `showmount` 애플리케이션 **SVM**을 위해

를 클릭합니다 `showmount` 기능을 사용하면 NFS 클라이언트가 SVM에서 사용 가능한 NFS 익스포트 목록을 쿼리할 수 있습니다. Kubernetes 클러스터에 구축된 POD에서 를 실행할 수 있습니다 `showmount -e` 데이터 LIF에 대한 명령을 실행하면 액세스 권한이 없는 마운트를 비롯하여 사용 가능한 마운트의 목록이 표시됩니다. 이는 그 자체로 보안 문제가 아니라, 권한이 없는 사용자가 NFS 내보내기에 연결하는 데 도움이 될 수 있는 불필요한 정보를 제공합니다.

를 비활성화해야 합니다 `showmount SVM` 레벨의 ONTAP CLI 명령을 사용하여 다음을 수행합니다.

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire 모범 사례

Trident를 위한 SolidFire 스토리지를 구성하기 위한 모범 사례에 대해 알아보십시오.

SolidFire 계정을 만듭니다

각 SolidFire 계정은 고유한 볼륨 소유자를 나타내며 자체 CHAP(Challenge-Handshake 인증 프로토콜) 자격 증명을 받습니다. 계정 이름 및 상대 CHAP 자격 증명을 사용하거나 볼륨 액세스 그룹을 통해 계정에 할당된 볼륨에 액세스할 수 있습니다. 계정에는 최대 2천 개의 볼륨이 할당될 수 있지만 볼륨은 하나의 계정에만 속할 수 있습니다.

QoS 정책을 생성합니다

여러 볼륨에 적용할 수 있는 표준화된 서비스 품질 설정을 만들어 저장하려면 SolidFire 서비스 품질(QoS) 정책을 사용하십시오.

볼륨별로 QoS 매개 변수를 설정할 수 있습니다. QoS를 정의하는 세 가지 구성 가능한 매개 변수, 즉 Min IOPS, Max IOPS, Burst IOPS를 설정하여 각 볼륨의 성능을 보장할 수 있습니다.

4KB 블록 크기에 대해 가능한 최소, 최대 및 버스트 IOPS 값입니다.

IOPS 매개 변수입니다	정의	최소 값	기본값	최대 가치(4KB)
최소 IOPS	볼륨에 대한 보장된 성능 수준.	50	50	15000
최대 IOPS	성능은 이 제한을 초과하지 않습니다.	50	15000	200,000
버스트 IOPS	짧은 버스트 시나리오에서 허용되는 최대 IOPS입니다.	50	15000	200,000



최대 IOPS와 버스트 IOPS는 최대 200,000으로 설정할 수 있지만, 실제 볼륨의 최대 성능은 클러스터 사용량 및 노드당 성능에 의해 제한됩니다.

블록 크기와 대역폭은 IOPS 수에 직접적인 영향을 미칩니다. 블록 크기가 증가함에 따라 시스템에서 더 큰 블록 크기를 처리하는 데 필요한 수준까지 대역폭을 높일 수 있습니다. 대역폭이 증가할수록 시스템에서 달성할 수 있는 IOPS의 수가 감소합니다. 을 참조하십시오 ["SolidFire 서비스 품질"](#) QoS 및 성능에 대한 자세한 내용은 를 참조하십시오.

SolidFire 인증

요소는 CHAP 및 vag(볼륨 액세스 그룹)의 두 가지 인증 방법을 지원합니다. CHAP는 CHAP 프로토콜을 사용하여 호스트를 백엔드에 인증합니다. 볼륨 액세스 그룹은 프로비전되는 볼륨에 대한 액세스를 제어합니다. NetApp은 CHAP를 사용하여 인증을 수행하는 것이 더 간단하고 확장 제한이 없기 때문에 CHAP를 사용하는 것이 좋습니다.



CSI 프로비저닝이 강화된 Trident는 CHAP 인증 사용을 지원합니다. VAG는 일반적인 비 CSI 작동 모드에서만 사용해야 합니다.

CHAP 인증(이니시에이터가 대상 볼륨 사용자인지 확인)은 계정 기반 액세스 제어에서만 지원됩니다. CHAP를 인증에 사용하는 경우 단방향 CHAP 및 양방향 CHAP의 두 가지 옵션을 사용할 수 있습니다. 단방향 CHAP는 SolidFire 계정 이름 및 이니시에이터 암호를 사용하여 볼륨 액세스를 인증합니다. 양방향 CHAP 옵션은 볼륨이 계정 이름과 이니시에이터 암호를 통해 호스트를 인증한 다음 호스트가 계정 이름과 타겟 암호를 통해 볼륨을 인증하기 때문에 볼륨을 인증하는 가장 안전한 방법을 제공합니다.

그러나 CHAP를 설정할 수 없고 VAG가 필요한 경우 액세스 그룹을 생성하고 호스트 이니시에이터 및 볼륨을 액세스 그룹에 추가합니다. 액세스 그룹에 추가하는 각 IQN은 CHAP 인증을 사용하거나 사용하지 않고 그룹의 각 볼륨에 액세스할 수 있습니다. iSCSI 이니시에이터가 CHAP 인증을 사용하도록 구성된 경우 계정 기반 액세스 제어가 사용됩니다. iSCSI 초기자가 CHAP 인증을 사용하도록 구성되지 않은 경우 볼륨 액세스 그룹 액세스 제어가 사용됩니다.

자세한 정보는 어디서 찾을 수 있습니까?

다음은 몇 가지 모범 사례 문서입니다. 를 검색합니다 ["NetApp 라이브러리"](#) 최신 버전의 경우.

- [ONTAP *](#)
- ["NFS Best Practice and Implementation Guide를 참조하십시오"](#)
- ["SAN 관리 가이드를 참조하십시오"](#) (iSCSI의 경우)
- ["RHEL용 iSCSI Express 구성"](#)

Element 소프트웨어 *

- ["Linux용 SolidFire 구성"](#)
- [NetApp HCI *](#)
- ["NetApp HCI 구축 사전 요구 사항"](#)
- ["NetApp 배포 엔진에 액세스합니다"](#)
- [응용 프로그램 모범 사례 정보 *](#)
- ["ONTAP 기반 MySQL의 모범 사례"](#)
- ["SolidFire 기반 MySQL의 모범 사례"](#)
- ["NetApp SolidFire 및 Cassandra"](#)
- ["SolidFire에 대한 Oracle 모범 사례"](#)
- ["SolidFire에 대한 PostgreSQL Best Practice"](#)

모든 애플리케이션에 구체적인 지침이 있는 것은 아니며 NetApp 팀과 함께 을 사용하는 것이 중요합니다 ["NetApp 라이브러리"](#) 최신 설명서를 참조하십시오.

Astra Trident 통합

Astra Trident를 통합하려면 다음과 같은 설계 및 아키텍처 요소를 통합해야 합니다. 드라이버 선택 및 배포, 스토리지 클래스 설계, 가상 풀 설계, PVC(Persistent Volume Claim)가 Astra Trident를 사용한 스토리지 프로비저닝, 볼륨 운영, OpenShift 서비스 구축에 미치는 영향

운전자 선택 및 전개

스토리지 시스템에 사용할 백엔드 드라이버를 선택하고 구축합니다.

ONTAP 백엔드 드라이버

ONTAP 백엔드 드라이버는 사용된 프로토콜과 스토리지 시스템에서 볼륨을 프로비저닝하는 방법에 따라 다릅니다. 따라서 배포할 드라이버를 결정할 때는 신중하게 고려해야 합니다.

상위 레벨에서는 애플리케이션에 공유 스토리지가 필요한 구성 요소(동일한 PVC에 액세스하는 여러 Pod)가 있는 경우 NAS 기반 드라이버가 기본 선택이고 블록 기반 iSCSI 드라이버는 비공유 스토리지의 요구를 충족합니다. 애플리케이션의 요구사항 및 스토리지 및 인프라 팀의 편안함 수준을 기준으로 프로토콜을 선택합니다. 일반적으로 대부분의 애플리케이션에서 두 서버 간에 차이가 거의 없기 때문에 공유 스토리지(둘 이상의 POD에 동시 액세스가 필요한 경우)가 필요한지 여부에 따라 결정하는 경우가 많습니다.

사용 가능한 ONTAP 백엔드 드라이버는 다음과 같습니다.

- `ontap-nas`: 각 PV는 완전한 ONTAP FlexVolume입니다.
- `ontap-nas-economy`: 각 PV 프로비저닝은 `qtree`이며, FlexVolume당 구성 가능한 Qtree 수가 있습니다 (기본값은 200).
- `ontap-nas-flexgroup`: 각 PV는 전체 ONTAP FlexGroup로 프로비저닝되고 SVM에 할당된 모든 애그리게이트가 사용됩니다.
- `ontap-san`: 각 PV는 자체 FlexVolume 내에 있는 LUN입니다.
- `ontap-san-economy`: 각 PV는 FlexVolume당 구성 가능한 LUN 수가 있는 LUN입니다(기본값은 100).

세 개의 NAS 드라이버 중 하나를 선택할 경우 해당 기능에 약간의 영향을 줍니다. 이 기능은 응용 프로그램에서 사용할 수 있습니다.

아래 표에서 모든 기능이 Astra Trident를 통해 표시되는 것은 아닙니다. 용량 할당 후 기능을 적용하려면 스토리지 관리자가 일부 기능을 적용해야 합니다. 위 첨자 각주는 기능 및 드라이버별 기능을 구별합니다.

ONTAP NAS 드라이버	스냅샷 수	복제	동적 익스포트 정책	다중 연결	QoS를 참조하십시오	크기 조정	복제
<code>ontap-nas</code>	예	예	Yesfootnote: 5[]	예	Yesfootnote: 1[]	예	Yesfootnote: 1[]
<code>ontap-nas-economy</code>	Yesfootnote: 3[]	Yesfootnote: 3[]	Yesfootnote: 5[]	예	Yesfootnote: 3[]	예	Yesfootnote: 3[]
<code>ontap-nas-flexgroup</code>	Yesfootnote: 1[]	아니요	Yesfootnote: 5[]	예	Yesfootnote: 1[]	예	Yesfootnote: 1[]

Astra Trident는 ONTAP용 SAN 드라이버 2개를 제공하며 해당 기능은 아래에 나와 있습니다.

ONTAP SAN 드라이버	스냅샷 수	복제	다중 연결	양방향 CHAP	QoS를 참조하십시오	크기 조정	복제
<code>ontap-san</code>	예	예	Yesfootnote: 4[]	예	Yesfootnote: 1[]	예	Yesfootnote: 1[]

ONTAP SAN 드라이버	스냅샷 수	복제	다중 연결	양방향 CHAP	QoS를 참조하십시오	크기 조정	복제
ontap-san-economy	예	예	Yesfootnote: 4[]	예	Yesfootnote: 3[]	예	Yesfootnote: 3[]

위 표의 각주: Yes [1]: Astra Trident에서 관리하지 않음 Yes [2]: Astra Trident에서 관리하지만 PV 세분화되지는 않음 Yes [3]: Astra Trident에서 관리하지 않음, PV 세분화됨 Yes [4]: 원시 블록 볼륨에서 지원됨 Yes [5]: CSI Trident에서 지원

PV 세분화되지 않은 기능은 전체 FlexVolume에 적용되고 모든 PVS(즉, 공유 FlexVol의 qtree 또는 LUN)는 공통 스케줄을 공유합니다.

위의 표에서 볼 수 있듯이, 이 기능 중 대부분은 `ontap-nas` 및 `ontap-nas-economy` 동일합니다. 그러나, 왜냐하면 `ontap-nas-economy` 드라이버는 PV 단위로 일정을 제어하는 기능을 제한하므로 특히 재해 복구 및 백업 계획에 영향을 줄 수 있습니다. ONTAP 스토리지에서 PVC 클론 기능을 활용하고자 하는 개발 팀의 경우 이를 사용할 때만 가능합니다 `ontap-nas`, `ontap-san` 또는 `ontap-san-economy` 드라이버.



를 클릭합니다 `solidfire-san` 운전자가 PVC를 클로닝할 수도 있습니다.

Cloud Volumes ONTAP 백엔드 드라이버

Cloud Volumes ONTAP은 NAS 및 SAN 프로토콜(NFS, SMB/CIFS 및 iSCSI)을 지원하는 파일 공유 및 블록 레벨 스토리지 등 다양한 활용 사례에 맞게 엔터프라이즈급 스토리지 기능과 함께 데이터 제어를 제공합니다. Cloud Volume ONTAP와 호환되는 드라이버는 `ontap-nas`, `ontap-nas-economy`, `ontap-san` 및 `ontap-san-economy`. Cloud Volume ONTAP for Azure, Cloud Volume ONTAP for GCP에 적용할 수 있습니다.

ONTAP 백엔드 드라이버용 Amazon FSx

ONTAP용 Amazon FSx를 사용하면 고객이 익숙한 NetApp 기능, 성능 및 관리 기능을 활용하는 동시에, AWS에 데이터를 저장하는 간편성, 민첩성, 보안, 확장성을 활용할 수 있습니다. ONTAP용 FSx는 ONTAP의 다양한 파일 시스템 기능과 관리 API를 지원합니다. Cloud Volume ONTAP와 호환되는 드라이버는 `ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san` 및 `ontap-san-economy`.

NetApp HCI/SolidFire 백엔드 드라이버

를 클릭합니다 `solidfire-san` NetApp HCI/SolidFire 플랫폼과 함께 사용되는 드라이버는 관리자가 QoS 제한을 기반으로 Trident에 대한 Element 백엔드를 구성하는 데 도움이 됩니다. Trident에서 프로비저닝한 볼륨에 대한 특정 QoS 제한을 설정하도록 백엔드를 설계하려면 이를 사용합니다 `type` 백엔드 파일의 매개 변수입니다. 또한 관리자는 이를 사용하여 스토리지에서 생성할 수 있는 볼륨 크기를 제한할 수 있습니다 `limitVolumeSize` 매개 변수. 현재 볼륨 크기 조정 및 볼륨 복제와 같은 Element 스토리지 기능은 여기서 지원되지 않습니다 `solidfire-san` 드라이버. 이러한 작업은 Element 소프트웨어 웹 UI를 통해 수동으로 수행해야 합니다.

SolidFire 드라이버	스냅샷 수	복제	다중 연결	CHAP	QoS를 참조하십시오	크기 조정	복제
solidfire-san	예	예	Yes [2]	예	예	예	Yesfootnote: 1[]

각주: Yesfootnote: 1 []: Astra Trident Yesfootnote: 2 []: 원시 블록 볼륨에서 지원됩니다

Azure NetApp Files 백엔드 드라이버

Astra Trident가 을 사용합니다 azure-netapp-files 를 관리할 드라이버 "Azure NetApp Files" 서비스.

이 드라이버 및 구성 방법에 대한 자세한 내용은 에서 찾을 수 있습니다 "Azure NetApp Files를 위한 Astra Trident 백엔드 구성입니다".

Azure NetApp Files 드라이버	스냅샷 수	복제	다중 연결	QoS를 참조하십시오	비즈니스	복제
azure-netapp-files	예	예	예	예	예	Yesfootnote: 1[]

각주: Yesfootnote: 1 []: Astra Trident에서 관리하지 않습니다

Cloud Volumes Service on Google Cloud 백엔드 드라이버

Astra Trident가 을 사용합니다 gcp-cvs Google Cloud에서 Cloud Volumes Service와 연결할 드라이버.

를 클릭합니다 gcp-cvs 드라이버는 가상 풀을 사용하여 백엔드를 추상화하고 Astra Trident가 볼륨 배치를 결정할 수 있도록 합니다. 관리자는 에서 가상 풀을 정의합니다 backend.json 파일. 스토리지 클래스는 선택기를 사용하여 레이블별로 가상 풀을 식별합니다.

- 백엔드에 가상 풀이 정의되어 있는 경우, Astra Trident는 Google Cloud 스토리지 풀에서 해당 가상 풀이 제한되는 볼륨을 생성하려고 시도합니다.
- 백엔드에 가상 풀이 정의되지 않은 경우 Astra Trident는 해당 지역의 사용 가능한 스토리지 풀에서 Google Cloud 스토리지 풀을 선택합니다.

Astra Trident에서 Google Cloud 백엔드를 구성하려면 을 지정해야 합니다 projectNumber, apiRegion, 및 apiKey 백엔드 파일 Google Cloud 콘솔에서 프로젝트 번호를 찾을 수 있습니다. API 키는 Google Cloud에서 Cloud Volumes Service에 대한 API 액세스를 설정할 때 생성한 서비스 계정 개인 키 파일에서 가져옵니다.

Cloud Volumes Service on Google Cloud 서비스 유형 및 서비스 수준에 대한 자세한 내용은 를 참조하십시오 "CVS for GCP에 대한 Astra Trident 지원에 대해 알아보십시오".

Google Cloud용 Cloud Volumes Service 드라이버	스냅샷 수	복제	다중 연결	QoS를 참조하십시오	비즈니스	복제
gcp-cvs	예	예	예	예	예	CVS에서 사용 가능 - 성능 서비스 유형만 해당



복제 참고 사항

- Astra Trident에서 복제를 관리하지 않습니다.
- 클론이 소스 볼륨과 동일한 스토리지 풀에 생성됩니다.

스토리지 클래스 설계

Kubernetes Storage Class 객체를 생성하려면 개별 스토리지 클래스를 구성 및 적용해야 합니다. 이 섹션에서는 애플리케이션에 대한 스토리지 클래스를 설계하는 방법에 대해 설명합니다.

특정 백엔드 활용도

특정 스토리지 클래스 객체 내에서 필터링을 사용하여 해당 스토리지 클래스에 사용할 스토리지 풀 또는 풀 세트를 결정할 수 있습니다. Storage Class(저장소 클래스)에서 세 가지 필터 세트를 설정할 수 있습니다. `storagePools`, `additionalStoragePools`, 및/또는 `excludeStoragePools`.

를 클릭합니다 `storagePools` 매개 변수는 지정된 속성과 일치하는 풀 세트로 스토리지를 제한하는 데 도움이 됩니다. 를 클릭합니다 `additionalStoragePools` 매개 변수는 Astra Trident가 프로비저닝에 사용할 풀 세트를 속성 및 에서 선택한 풀 세트와 확장하는 데 사용됩니다 `storagePools` 매개 변수. 매개 변수만 사용하거나 둘 모두를 함께 사용하여 적절한 스토리지 풀 세트가 선택되었는지 확인할 수 있습니다.

를 클릭합니다 `excludeStoragePools` 매개 변수는 속성과 일치하는 나열된 풀 세트를 특별히 제외하는 데 사용됩니다.

QoS 정책을 에뮬레이트합니다

서비스 품질 정책을 에뮬레이트하기 위해 스토리지 클래스를 설계하려면 를 사용하여 스토리지 클래스를 생성합니다 `media` 속성 `hdd` 또는 `ssd`. 을 기반으로 합니다 `media` 스토리지 클래스에 설명된 특성인 Trident는 제공하는 적절한 백엔드를 선택합니다 `hdd` 또는 `ssd Aggregate`는 미디어 속성과 일치시킨 다음, 볼륨 프로비저닝을 특정 애그리게이트로 전달합니다. 따라서 가지고 있는 스토리지 클래스 Premium을 생성할 수 있습니다 `media` 속성을 로 설정합니다 `ssd 프리미엄 QoS` 정책으로 분류될 수 있습니다. 표준 QoS 정책으로 분류될 수 있는 미디어 속성을 'HDD'로 설정하는 또 다른 스토리지 클래스 표준을 생성할 수 있습니다. 또한 스토리지 클래스에서 ""IOPS"" 속성을 사용하여 QoS 정책으로 정의할 수 있는 Element 어플라이언스로 프로비저닝을 리디렉션할 수도 있습니다.

특정 기능을 기반으로 백엔드를 활용합니다

스토리지 클래스는 씬 및 일반 프로비저닝, 스냅샷, 클론 및 암호화와 같은 기능이 설정된 특정 백엔드에서 볼륨 프로비저닝을 수행하도록 설계되었습니다. 사용할 스토리지를 지정하려면 필요한 기능이 설정된 적절한 백엔드를 지정하는 스토리지 클래스를 생성합니다.

가상 풀

모든 Astra Trident 백엔드에 가상 풀을 사용할 수 있습니다. Astra Trident가 제공하는 모든 드라이버를 사용하여 백엔드에 대한 가상 풀을 정의할 수 있습니다.

가상 풀을 사용하면 관리자가 저장소 클래스를 통해 참조할 수 있는 백엔드에 대한 추상화 수준을 생성하여 백엔드에 볼륨을 보다 유연하고 효율적으로 배치할 수 있습니다. 동일한 서비스 클래스로 다른 백엔드를 정의할 수 있습니다. 또한 동일한 백엔드에서 여러 스토리지 풀을 생성할 수 있지만 특성이 다릅니다. 특정 레이블이 있는 선택기로 스토리지 클래스를 구성한 경우 Astra Trident는 볼륨을 배치할 모든 선택기 레이블과 일치하는 백엔드를 선택합니다. 스토리지 클래스 선택기 레이블이 여러 스토리지 풀과 일치하면 Astra Trident가 볼륨 용량을 할당할 스토리지 풀 중 하나를 선택합니다.

가상 풀 설계

백엔드를 생성하는 동안 일반적으로 매개 변수 집합을 지정할 수 있습니다. 관리자가 동일한 스토리지 자격 증명을 사용하여 다른 매개 변수 집합을 가진 다른 백엔드를 생성할 수 없었습니다. 가상 풀이 도입됨에 따라 이 문제가 완화되었습니다. 가상 풀은 백엔드 및 Kubernetes 스토리지 클래스 간에 도입된 레벨 추상화입니다. 따라서 관리자는

Kubernetes 스토리지 클래스를 통해 백엔드에 독립적인 방식으로 Selector로 참조할 수 있는 레이블과 함께 매개 변수를 정의할 수 있습니다. Astra Trident를 사용하여 지원되는 모든 NetApp 백엔드에 가상 풀을 정의할 수 있습니다. 해당 목록에는 SolidFire/NetApp HCI, ONTAP, Cloud Volumes Service on GCP 및 Azure NetApp Files가 포함됩니다.



가상 풀을 정의할 때는 백엔드 정의에서 기존 가상 풀의 순서를 재정렬하지 않는 것이 좋습니다. 또한 기존 가상 풀의 속성을 편집/수정하고 대신 새 가상 풀을 정의하는 것이 좋습니다.

다양한 서비스 수준/QoS 에뮬레이션

서비스 클래스를 에뮬레이트하기 위한 가상 풀을 설계할 수 있습니다. Azure NetApp Files용 Cloud Volume Service에 대한 가상 풀 구현을 사용하여 다양한 서비스 클래스를 설정하는 방법을 살펴보겠습니다. 다양한 성능 수준을 나타내는 여러 레이블을 사용하여 ANF 백엔드를 구성합니다. 설정 `servicelevel` 적절한 성과 수준에 맞게 중횡비를 지정하고 각 레이블 아래에 다른 필요한 요소를 추가합니다. 이제 다른 가상 풀에 매핑할 다른 Kubernetes 스토리지 클래스를 생성합니다. 를 사용합니다 `parameters.selector` 필드에서 각 StorageClass는 볼륨을 호스팅하는 데 사용할 수 있는 가상 풀을 호출합니다.

특정 측면 지정

특정 측면의 여러 가상 풀을 단일 스토리지 백엔드에서 설계할 수 있습니다. 이를 위해 백엔드에 여러 레이블을 구성하고 각 레이블 아래에 필요한 측면을 설정합니다. 이제 를 사용하여 다양한 Kubernetes Storage 클래스를 생성할 수 있습니다 `parameters.selector` 다른 가상 풀에 매핑될 필드입니다. 백엔드에서 프로비저닝되는 볼륨에는 선택한 가상 풀에 정의된 측면이 있습니다.

스토리지 프로비저닝에 영향을 미치는 PVC 특성

요청된 스토리지 클래스 이외의 일부 매개 변수는 PVC 생성 시 Astra Trident 프로비저닝 결정 프로세스에 영향을 줄 수 있습니다.

액세스 모드

PVC를 통한 저장 요청 시 필수 필드 중 하나가 액세스 모드입니다. 원하는 모드는 스토리지 요청을 호스팅하기 위해 선택한 백엔드에 영향을 줄 수 있습니다.

Astra Trident는 다음 매트릭스에 따라 지정된 액세스 방법과 사용된 스토리지 프로토콜을 일치시키려고 시도합니다. 이는 기본 스토리지 플랫폼과 무관합니다.

	ReadWriteOnce 를 참조하십시오	ReadOnlyMany 를 참조하십시오	ReadWriteMany 를 참조하십시오
iSCSI	예	예	예(원시 블록)
NFS 를 참조하십시오	예	예	예

NFS 백엔드가 구성되지 않은 상태로 Trident 배포에 제출된 ReadWriteMany PVC에 대한 요청은 볼륨이 프로비저닝되지 않습니다. 이러한 이유로 요청자는 자신의 응용 프로그램에 적합한 액세스 모드를 사용해야 합니다.

볼륨 작업입니다

영구 볼륨 수정

영구 볼륨은 Kubernetes에서 두 가지 예외, 영구적 객체입니다. 생성된 후에는 부가세 반환 청구액 정책 및 크기를 수정할 수 있습니다. 그러나 이렇게 해서 Kubernetes 외부에서 볼륨의 일부 측면이 수정되지 않도록 할 수는 없습니다. 특정 애플리케이션에 맞게 볼륨을 사용자 지정하거나, 실수로 용량이 소비되지 않도록 하거나, 어떠한 이유로든 볼륨을 다른 스토리지 컨트롤러로 이동하는 것이 좋을 수 있습니다.



현재 Kubernetes 트리 프로비저닝 시 NFS 또는 iSCSI PVS의 볼륨 크기 조정 작업은 지원되지 않습니다. Astra Trident는 NFS 및 iSCSI 볼륨 확장을 지원합니다.

PV의 접속 세부 정보는 생성 후 수정할 수 없습니다.

주문형 볼륨 스냅샷을 생성합니다

Astra Trident는 CSI 프레임워크를 사용하여 필요 시 볼륨 스냅샷 생성 및 스냅샷에서 PVC 생성을 지원합니다. 스냅샷은 편리한 데이터 시점 복사본을 유지 관리하는 방법을 제공하며 Kubernetes의 소스 PV와 독립적인 라이프사이클을 갖고 있습니다. 이러한 스냅샷을 사용하여 PVC를 복제할 수 있습니다.

스냅샷으로부터 볼륨을 생성합니다

Astra Trident는 볼륨 스냅샷으로부터 PersistentVolumes 생성을 지원합니다. 이를 수행하려면 PersistentVolumeClaim을 생성하고 을 언급하기만 하면 됩니다 datasource 볼륨을 생성해야 하는 필수 스냅샷입니다. Astra Trident는 스냅샷에 데이터가 있는 볼륨을 생성하여 이 PVC를 처리합니다. 이 기능을 사용하면 지역 간에 데이터를 복제하거나 테스트 환경을 생성하거나 손상되거나 손상된 운영 볼륨을 전체적으로 교체하거나 특정 파일 및 디렉토리를 검색하여 연결된 다른 볼륨으로 전송할 수 있습니다.

클러스터에서 볼륨 이동

스토리지 관리자는 ONTAP 클러스터의 Aggregate와 컨트롤러 간에 볼륨을 스토리지 소비자로 중단 없이 이동할 수 있습니다. 대상 애그리게이트는 Astra Trident가 사용하는 SVM이 액세스할 수 있는 경우, 이 작업은 Astra Trident 또는 Kubernetes 클러스터에 영향을 주지 않습니다. 여기서 중요한 점은 애그리게이트를 SVM에 새로 추가한 경우, Astra Trident에 다시 추가하여 백엔드를 새로 고쳐야 한다는 것입니다. 그러면 Astra Trident가 SVM의 인벤토리를 다시 만들어 새 애그리게이트를 인식할 수 있습니다.

그러나 Astra Trident는 백엔드에서 볼륨을 이동하는 기능을 자동으로 지원하지 않습니다. 여기에는 동일한 클러스터, 클러스터 간 또는 다른 스토리지 플랫폼(스토리지 시스템이 Astra Trident에 연결된 SVM인 경우에도 해당 스토리지 플랫폼)에 있는 SVM이 포함됩니다.

볼륨이 다른 위치에 복사되면 볼륨 가져오기 기능을 사용하여 현재 볼륨을 Astra Trident로 가져올 수 있습니다.

볼륨 확장

Astra Trident는 NFS 및 iSCSI PVS 크기를 조정할 수 있도록 지원합니다. 따라서 사용자는 Kubernetes 계층을 통해 직접 볼륨의 크기를 조정할 수 있습니다. ONTAP, SolidFire/NetApp HCI 및 Cloud Volumes Service 백엔드를 포함한 모든 주요 NetApp 스토리지 플랫폼에서 볼륨 확장이 가능합니다. 나중에 가능한 확장을 허용하려면 를 설정합니다 allowVolumeExpansion 를 선택합니다 true 볼륨과 연결된 StorageClass에서 영구 볼륨의 크기를 조정해야 할 때마다 를 편집합니다 spec.resources.requests.storage 영구 볼륨 클레임의 주석을 필요한 볼륨 크기로 설정합니다. Trident는 스토리지 클러스터의 볼륨 크기를 자동으로 조정합니다.

기존 볼륨을 Kubernetes로 임포트

볼륨 가져오기를 사용하면 기존 스토리지 볼륨을 Kubernetes 환경으로 가져올 수 있습니다. 이 기능은 현재 에서

지원됩니다 `ontap-nas`, `ontap-nas-flexgroup`, `solidfire-san`, `azure-netapp-files`, 및 `gcp-cvs` 드라이버. 이 기능은 기존 애플리케이션을 Kubernetes로 포팅하거나 재해 복구 시나리오에서 유용합니다.

ONTAP 및 를 사용하는 경우 `solidfire-san` 드라이버, 명령을 사용합니다 `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml` Astra Trident에서 관리할 기존 볼륨을 Kubernetes로 가져오려면 볼륨 가져오기 명령에 사용되는 PVC YAML 또는 JSON 파일은 Astra Trident를 프로비저닝자로 식별하는 스토리지 클래스를 가리킵니다. NetApp HCI/SolidFire 백엔드를 사용할 경우 볼륨 이름이 고유한지 확인합니다. 볼륨 이름이 중복되면 볼륨을 고유한 이름으로 복제하여 볼륨 가져오기 기능에서 볼륨 이름을 구분할 수 있도록 합니다.

를 누릅니다 `azure-netapp-files` 또는 `gcp-cvs` 드라이버가 사용되는 경우 명령을 사용합니다 `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` Astra Trident에서 관리할 Kubernetes로 볼륨을 가져오려면 이렇게 하면 고유한 볼륨 참조가 보장됩니다.

위 명령을 실행하면 Astra Trident가 백엔드에서 볼륨을 찾고 해당 크기를 읽습니다. 구성된 PVC의 볼륨 크기를 자동으로 추가(필요한 경우 덮어쓰기)합니다. 그런 다음 Astra Trident가 새로운 PV를 생성하고 Kubernetes가 PVC를 PV에 결합합니다.

특정 가져온 PVC가 필요한 컨테이너를 배포한 경우 PVC/PV 쌍이 볼륨 가져오기 프로세스를 통해 바인딩될 때까지 보류 상태로 유지됩니다. PVC/PV 쌍이 바인딩되면 다른 문제가 없는 한 컨테이너가 나타나야 합니다.

OpenShift 서비스를 배포합니다

OpenShift 부가 가치 클러스터 서비스는 클러스터 관리자와 호스팅 중인 애플리케이션에 중요한 기능을 제공합니다. 이러한 서비스가 사용되는 스토리지는 노드 로컬 리소스를 사용하여 프로비저닝할 수 있지만, 이로 인해 서비스의 용량, 성능, 복구 가능성 및 지속 가능성이 제한되기도 합니다. 엔터프라이즈 스토리지 어레이를 활용하여 이러한 서비스에 필요한 용량을 제공하면 서비스를 대폭 향상시킬 수 있습니다. 그러나 모든 애플리케이션과 마찬가지로 OpenShift와 스토리지 관리자는 긴밀하게 협력하여 각 애플리케이션에 가장 적합한 옵션을 결정해야 합니다. Red Hat 문서는 요구 사항을 결정하고 사이징 및 성능 요구 사항을 충족할 수 있도록 적극 활용해야 합니다.

레지스트리 서비스

레지스트리의 스토리지 배포 및 관리는 에 설명되어 있습니다 "[NetApp.IO를 참조하십시오](#)" 에 있습니다 "[블로그](#)".

로깅 서비스

다른 OpenShift 서비스와 마찬가지로 로깅 서비스는 Ansible을 사용하여 인벤토리 파일에서 제공하는 구성 매개 변수로 배포됩니다 호스트가 플레이북에 제공됩니다. OpenShift를 설치한 후 초기 OpenShift 설치 중에 로깅을 배포하고 로깅을 배포하는 두 가지 설치 방법이 제공됩니다.



Red Hat OpenShift 버전 3.9를 기준으로 공식 문서는 데이터 손상 관련 우려 때문에 로깅 서비스에 NFS를 사용할 것을 권장합니다. 이는 제품에 대한 Red Hat 테스트를 기반으로 합니다. ONTAP의 NFS 서버에는 이러한 문제가 없으며 로깅 구축을 쉽게 되돌릴 수 있습니다. 궁극적으로, 로깅 서비스를 위한 프로토콜을 선택할 수 있습니다. 두 가지 모두 NetApp 플랫폼을 사용할 때 효과가 있으며 원할 경우 NFS를 피할 이유가 없습니다.

로깅 서비스에서 NFS를 사용하기로 결정한 경우 Ansible 변수를 설정해야 합니다

`openshift_enable_unsupported_configurations` 를 선택합니다 `true` 설치 프로그램이 실패하는 것을 방지합니다.

시작하십시오

로깅 서비스는 필요에 따라 두 애플리케이션 및 OpenShift 클러스터 자체의 핵심 운영에 구축할 수 있습니다. 변수를 지정하여 작업 로깅을 배포하도록 선택하는 경우 `openshift_logging_use_ops` 현재 `true` 서비스 인스턴스가 두 개 생성됩니다. 작업에 대한 로깅 인스턴스를 제어하는 변수에는 "ops"가 포함되어 있지만 응용 프로그램의 인스턴스는 그렇지 않습니다.

기본 서비스에서 올바른 스토리지를 활용할 수 있도록 구축 방법에 따라 Ansible 변수를 구성하는 것이 중요합니다. 각 배포 방법에 대한 옵션을 살펴보겠습니다.



아래 표에는 로깅 서비스와 관련된 스토리지 구성과 관련된 변수만 포함되어 있습니다. 에서 다른 옵션을 찾을 수 있습니다 ["RedHat OpenShift 로깅 설명서"](#) 배포 내용에 따라 검토, 구성 및 사용해야 합니다.

아래 표의 변수는 제공된 세부 정보를 사용하여 로깅 서비스에 대한 PV 및 PVC를 생성하는 Ansible 플레이북을 만듭니다. 이 방법은 OpenShift 설치 후 구성 요소 설치 플레이북을 사용하는 것보다 훨씬 덜 유연하지만, 기존 볼륨을 사용할 수 있는 경우 옵션으로 제공됩니다.

변수	세부 정보
<code>openshift_logging_storage_kind</code>	를 로 설정합니다 <code>nfs</code> 설치 프로그램이 로깅 서비스에 대한 NFS PV를 생성하도록 합니다.
<code>openshift_logging_storage_host</code>	NFS 호스트의 호스트 이름 또는 IP 주소입니다. 이 경우 가상 머신의 데이터 LIF로 설정해야 합니다.
<code>openshift_logging_storage_nfs_directory</code>	NFS 내보내기의 마운트 경로입니다. 예를 들어, 볼륨이 과 같이 분기되어 있는 경우 <code>/openshift_logging</code> , 이 변수에 해당 경로를 사용합니다.
<code>openshift_logging_storage_volume_name</code>	이름(예 <code>pv_ose_logs</code> , 생성할 PV의).
<code>openshift_logging_storage_volume_size</code>	예를 들어, NFS 내보내기의 크기입니다 <code>100Gi</code> .

OpenShift 클러스터가 이미 실행 중이고 Trident가 배포 및 구성된 경우 설치 관리자는 동적 프로비저닝을 사용하여 볼륨을 생성할 수 있습니다. 다음 변수를 구성해야 합니다.

변수	세부 정보
<code>openshift_logging_es_pvc_dynamic</code>	동적으로 프로비저닝된 볼륨을 사용하려면 <code>true</code> 로 설정합니다.
<code>openshift_logging_es_pvc_storage_class_name</code>	PVC에 사용될 스토리지 클래스의 이름입니다.
<code>openshift_logging_es_pvc_size</code>	PVC에서 요청된 체적의 크기입니다.
<code>openshift_logging_es_pvc_prefix</code>	로깅 서비스에서 사용하는 PVC의 접두사입니다.
<code>openshift_logging_es_ops_pvc_dynamic</code>	를 로 설정합니다 <code>true</code> 작업 로깅 인스턴스에 동적으로 프로비저닝된 볼륨을 사용하려면
<code>openshift_logging_es_ops_pvc_storage_class_name</code>	작업 로깅 인스턴스에 대한 스토리지 클래스의 이름입니다.
<code>openshift_logging_es_ops_pvc_size</code>	작업 인스턴스에 대한 볼륨 요청의 크기입니다.
<code>openshift_logging_es_ops_pvc_prefix</code>	<code>ops instance PVCs(ops 인스턴스 PVC)</code> 의 접두사입니다.

로깅 스택을 배포합니다

초기 OpenShift 설치 프로세스의 일부로 로깅을 배포하는 경우 표준 배포 프로세스만 따르면 됩니다. Ansible이 완료되는 즉시 서비스를 이용할 수 있도록 필요한 서비스와 OpenShift 개체를 구성 및 배포합니다.

하지만 초기 설치 후에 구축할 경우 구성 요소 플레이북을 Ansible에서 사용해야 합니다. 이 프로세스는 다른 버전의 OpenShift에서 약간 변경될 수 있으므로 반드시 읽고 따라야 합니다 "[RedHat OpenShift Container Platform 3.11 설명서](#)" 를 참조하십시오.

메트릭 서비스

메트릭 서비스는 관리자에게 OpenShift 클러스터의 상태, 리소스 활용도 및 가용성에 대한 중요한 정보를 제공합니다. 또한 POD 자동 확장 기능도 필요하며, 많은 조직에서 비용 청구 및/또는 애플리케이션 표시를 위해 메트릭 서비스의 데이터를 사용합니다.

로깅 서비스 및 OpenShift와 마찬가지로 Ansible을 사용하여 메트릭 서비스를 배포합니다. 또한 로깅 서비스와 마찬가지로 메트릭 서비스는 클러스터의 초기 설정 중에 또는 구성 요소 설치 방법을 사용하여 작동 후에 배포될 수 있습니다. 다음 표에는 메트릭 서비스에 대한 영구 스토리지를 구성할 때 중요한 변수가 나와 있습니다.



아래 표에는 메트릭 서비스와 관련된 스토리지 구성과 관련된 변수만 포함되어 있습니다. 문서에 나와 있는 다른 많은 옵션은 배포 내용에 따라 검토, 구성 및 사용해야 합니다.

변수	세부 정보
openshift_metrics_storage_kind	를 로 설정합니다 nfs 설치 프로그램이 로깅 서비스에 대한 NFS PV를 생성하도록 합니다.
openshift_metrics_storage_host	NFS 호스트의 호스트 이름 또는 IP 주소입니다. SVM을 위한 데이터 LIF로 설정해야 합니다.
openshift_metrics_storage_nfs_directory	NFS 내보내기의 마운트 경로입니다. 예를 들어, 볼륨이 과 같이 분기되어 있는 경우 /openshift_metrics, 이 변수에 해당 경로를 사용합니다.
openshift_metrics_storage_volume_name	이름(예 pv_ose_metrics, 생성할 PV의).
openshift_metrics_storage_volume_size	예를 들어, NFS 내보내기의 크기입니다 100Gi.

OpenShift 클러스터가 이미 실행 중이고 Trident가 배포 및 구성된 경우 설치 관리자는 동적 프로비저닝을 사용하여 볼륨을 생성할 수 있습니다. 다음 변수를 구성해야 합니다.

변수	세부 정보
openshift_metrics_cassandra_pvc_prefix	지표 PVC에 사용할 접두사입니다.
openshift_metrics_cassandra_pvc_size	요청할 볼륨의 크기입니다.
openshift_metrics_cassandra_storage_type	메트릭에 사용할 스토리지 유형으로, 적절한 스토리지 클래스로 PVC를 생성하려면 Ansible에서 이를 동적으로 설정해야 합니다.
openshift_metrics_cassandra_pvc_storage_class_name	사용할 스토리지 클래스의 이름입니다.

메트릭 서비스를 구축합니다

호스트/인벤토리 파일에 정의된 적절한 Ansible 변수를 사용하여 서비스를 구축하십시오. OpenShift 설치 시 배포하는 경우 PV가 자동으로 생성되고 사용됩니다. OpenShift를 설치한 후 구성 요소 플레이북을 사용하여 배포하는 경우, Ansible이 필요한 PVC를 만들고 Astra Trident가 PVC를 위한 스토리지를 프로비저닝하면 서비스를 배포합니다.

위의 변수와 배포 프로세스는 각 OpenShift 버전에 따라 변경될 수 있습니다. 검토 후 준수해야 합니다 ["RedHat의 OpenShift 배포 가이드"](#) 사용자 환경에 맞게 구성되도록 사용자의 버전에 대해.

데이터 보호

NetApp 스토리지 플랫폼이 제공하는 데이터 보호 및 복구 옵션에 대해 알아보십시오. Astra Trident는 이러한 기능 중 일부를 활용할 수 있는 볼륨을 프로비저닝할 수 있습니다. 지속성 요구사항이 있는 각 애플리케이션에 대한 데이터 보호 및 복구 전략이 있어야 합니다.

를 백업합니다 etcd 클러스터 데이터

Astra Trident는 Kubernetes 클러스터에 메타데이터를 저장합니다 etcd 데이터베이스: 를 주기적으로 백업합니다 etcd 클러스터 데이터는 재해 시나리오에서 Kubernetes 클러스터를 복구하는 데 중요합니다.

단계

1. 를 클릭합니다 etcdctl snapshot save 명령을 사용하면 의 시점 스냅샷을 생성할 수 있습니다 etcd 클러스터:

```
sudo docker run --rm -v /backup:/backup \
  --network host \
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \
  --env ETCDCTL_API=3 \
  registry.k8s.io/etcd-amd64:3.2.18 \
  etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  snapshot save /backup/etcd-snapshot.db
```

이 명령은 etcd 컨테이너를 스펀업하여 etcd 스냅샷을 생성하고 에 저장합니다 /backup 디렉토리.

2. 재해가 발생할 경우 etcd 스냅샷을 사용하여 Kubernetes 클러스터를 사용할 수 있습니다. 를 사용합니다 etcdctl snapshot restore 에 대해 생성된 특정 스냅샷을 복원하는 명령입니다 /var/lib/etcd 폴더. 복원 후, 이(가) 있는지 확인합니다 /var/lib/etcd 폴더가 로 채워졌습니다 member 폴더. 다음은 의 예입니다 etcdctl snapshot restore 명령:

```
etcdctl snapshot restore '/backup/etcd-snapshot-latest.db' ; mv
/default.etcd/member/ /var/lib/etcd/
```

3. Kubernetes 클러스터를 초기화하기 전에 필요한 인증서를 모두 복사합니다.
4. 로 클러스터를 생성합니다 `--ignore-preflight-errors=DirAvailable-var-lib-etcd` 깃발.
5. 클러스터가 완료되면 `kubbe-system` 포드가 시작되었는지 확인합니다.
6. 를 사용합니다 `kubect1 get crd Trident`에 의해 생성된 사용자 지정 리소스가 있는지 확인하고 Trident 객체를 검색하여 모든 데이터를 사용할 수 있는지 확인하는 명령입니다.

ONTAP 스냅샷을 사용하여 날짜를 복구합니다

스냅샷은 애플리케이션 데이터에 대한 시점 복구 옵션을 제공하여 중요한 역할을 합니다. 그러나 스냅샷은 자체적으로 백업되는 것이 아니며, 스토리지 시스템 장애나 기타 재난으로부터 데이터를 보호하지 않습니다. 그러나 대부분의 경우 데이터를 쉽고 빠르고 쉽게 복구할 수 있습니다. ONTAP 스냅샷 기술을 사용하여 볼륨을 백업하는 방법과 복원하는 방법에 대해 알아보십시오.

- 백엔드에 스냅샷 정책이 정의되지 않은 경우 기본적으로 을 사용합니다 `none` 정책. 이로 인해 ONTAP에서 자동 스냅샷이 생성되지 않습니다. 그러나 스토리지 관리자는 ONTAP 관리 인터페이스를 통해 수동 스냅샷을 생성하거나 스냅샷 정책을 변경할 수 있습니다. Trident 작업에는 영향을 주지 않습니다.
- 스냅샷 디렉토리는 기본적으로 숨겨져 있습니다. 이렇게 하면 를 사용하여 프로비저닝된 볼륨의 호환성을 극대화할 수 있습니다 `ontap-nas` 및 `ontap-nas-economy` 드라이버. 를 활성화합니다 `.snapshot` 디렉토리를 선택합니다 `ontap-nas` 및 `ontap-nas-economy` 응용 프로그램이 스냅샷에서 직접 데이터를 복구할 수 있도록 하는 드라이버입니다.
- 를 사용하여 이전 스냅샷에 기록된 상태로 볼륨을 복원합니다 `volume snapshot restore` ONTAP CLI 명령 스냅샷 복사본을 복구할 때 복구 작업은 기존 볼륨 구성을 덮어씁니다. 스냅샷 복사본이 생성된 후 볼륨의 데이터에 대한 모든 변경 사항은 손실됩니다.

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

ONTAP를 사용하여 데이터 복제

스토리지 시스템 장애로 인한 데이터 손실로부터 데이터를 복제하는 것은 중요한 역할을 할 수 있습니다.



ONTAP 복제 기술에 대한 자세한 내용은 를 참조하십시오 "[ONTAP 설명서](#)".

SnapMirror SVM(Storage Virtual Machines) 복제

을 사용할 수 있습니다 "[SnapMirror를 참조하십시오](#)" 전체 SVM을 복제하며, 여기에는 구성 설정 및 볼륨이 포함됩니다. 재해가 발생할 경우 SnapMirror 대상 SVM을 활성화하여 데이터 제공을 시작할 수 있습니다. 시스템이 복원되면 기본 시스템으로 다시 전환할 수 있습니다.

Astra Trident는 복제 관계 자체를 구성할 수 없기 때문에 스토리지 관리자는 ONTAP의 SnapMirror SVM 복제 기능을 사용하여 DR(재해 복구) 대상에 볼륨을 자동으로 복제할 수 있습니다.

SnapMirror SVM 복제 기능을 사용할 계획이거나 현재 기능을 사용 중인 경우 다음을 고려하십시오.

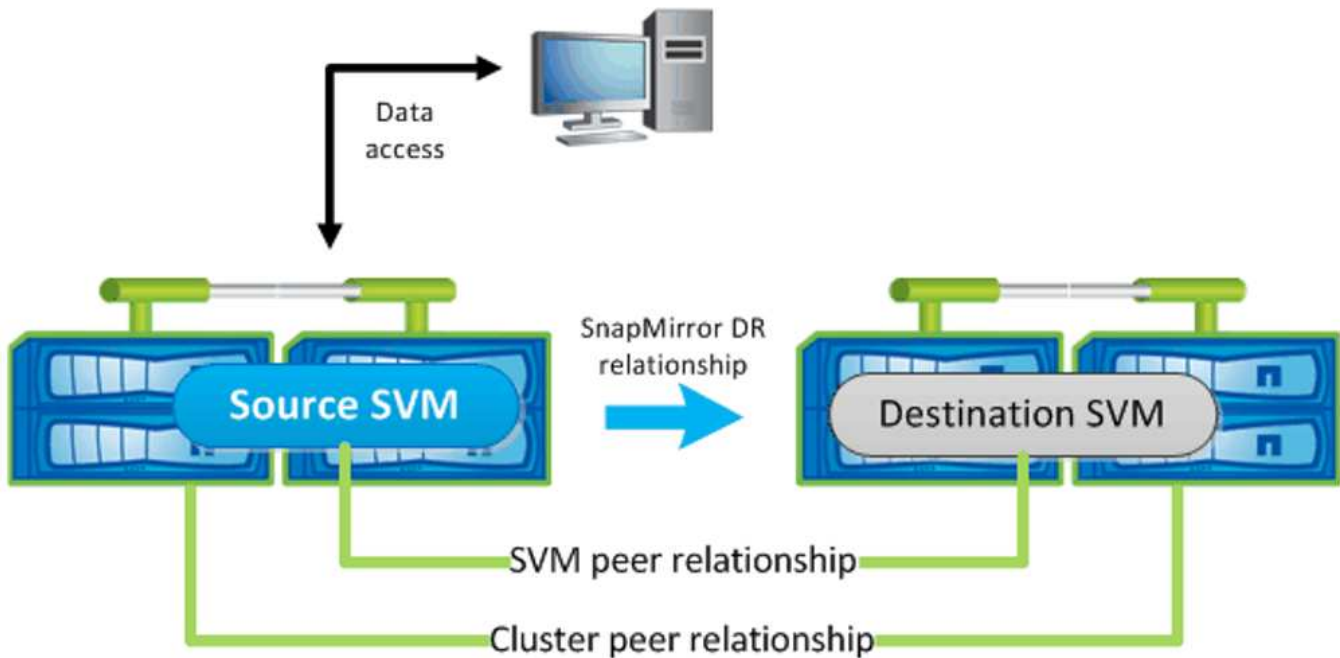
- SVM-DR이 활성화된 각 SVM에 대해 별개의 백엔드를 생성해야 합니다.
- 필요한 경우를 제외하고 복제된 백엔드를 선택하지 않도록 스토리지 클래스를 구성해야 합니다. 이는 SVM-DR을

지원하는 백엔드에 복제 관계를 프로비저닝하지 않아도 되는 볼륨이 생기지 않도록 하는 데 중요합니다.

- 애플리케이션 관리자는 데이터 복제와 관련된 추가 비용 및 복잡성을 이해하고 데이터 복제를 활용하기 전에 복구 계획을 결정해야 합니다.
- SnapMirror 대상 SVM을 활성화하기 전에 예약된 SnapMirror 전송을 모두 중지하고, 진행 중인 SnapMirror 전송을 모두 중단하고, 복제 관계를 중지하고, 소스 SVM을 중지한 다음, SnapMirror 대상 SVM을 시작하십시오.
- Astra Trident는 SVM 장애를 자동으로 감지하지 않습니다. 따라서 장애가 발생하면 관리자가 이를 실행해야 합니다 `tridentctl backend update` 새 백엔드에 대한 Trident의 장애 조치를 트리거하는 명령입니다.

다음은 SVM 설정 단계에 대한 개요입니다.

- 소스 클러스터와 타겟 클러스터 및 SVM 간 피어링을 설정합니다.
- `rl` 사용하여 대상 SVM을 생성합니다 `-subtype dp-destination` 옵션을 선택합니다.
- 필요한 간격으로 복제가 수행되도록 복제 작업 스케줄을 생성합니다.
- `rl` 사용하여 대상 SVM에서 소스 SVM으로 SnapMirror 복제를 생성합니다 `-identity-preserve true` 소스 SVM 구성과 소스 SVM 인터페이스가 타겟으로 복제되도록 보장하는 옵션 대상 SVM에서 SnapMirror SVM 복제 관계를 초기화합니다.



Trident를 위한 재해 복구 워크플로우

Astra Trident 19.07 이상 Kubernetes CRD를 사용하여 자체 상태를 저장 및 관리합니다. Kubernetes 클러스터를 사용합니다 `etcd` 메타데이터를 저장합니다. 이 경우에는 Kubernetes를 사용하는 것으로 가정합니다 `etcd` 데이터 파일 및 인증서는 NetApp FlexVolume에 저장됩니다. 이 FlexVolume은 SVM에 상주하며 보조 사이트의 대상 SVM과 SnapMirror SVM-DR 관계가 있습니다.

다음 단계에서는 재해 발생 시 Astra Trident를 사용하여 단일 마스터 Kubernetes 클러스터를 복구하는 방법을 설명합니다.

1. 소스 SVM에 장애가 발생하면 SnapMirror 타겟 SVM을 활성화합니다. 이렇게 하려면 예약된 SnapMirror 전송을 중지하고, 지속적인 SnapMirror 전송을 중단하고, 복제 관계를 중단하고, 소스 SVM을 중지하고, 타겟 SVM을

시작해야 합니다.

- 대상 SVM에서 Kubernetes가 포함된 볼륨을 마운트합니다 `etcd` 마스터 노드로 설정할 호스트에 데이터 파일 및 인증서
- 에서 Kubernetes 클러스터와 관련된 모든 필수 인증서를 복사합니다 `/etc/kubernetes/pki` 그리고 `etcd member` 파일 위치 `/var/lib/etcd`.
- 를 사용하여 Kubernetes 클러스터를 생성합니다 `kubeadm init` 명령과 함께 `--ignore-preflight-errors=DirAvailable-var-lib-etcd` 깃발. Kubernetes 노드에 사용되는 호스트 이름은 소스 Kubernetes 클러스터와 동일해야 합니다.
- 를 실행합니다 `kubectl get crd` 모든 Trident 사용자 지정 리소스가 표시되는지 확인하고 Trident 객체를 검색하여 모든 데이터를 사용할 수 있는지 확인하는 명령입니다.
- 를 실행하여 새 대상 SVM 이름을 반영하도록 필수 백엔드를 업데이트합니다 `./tridentctl update backend <backend-name> -f <backend-json-file> -n <namespace>` 명령.



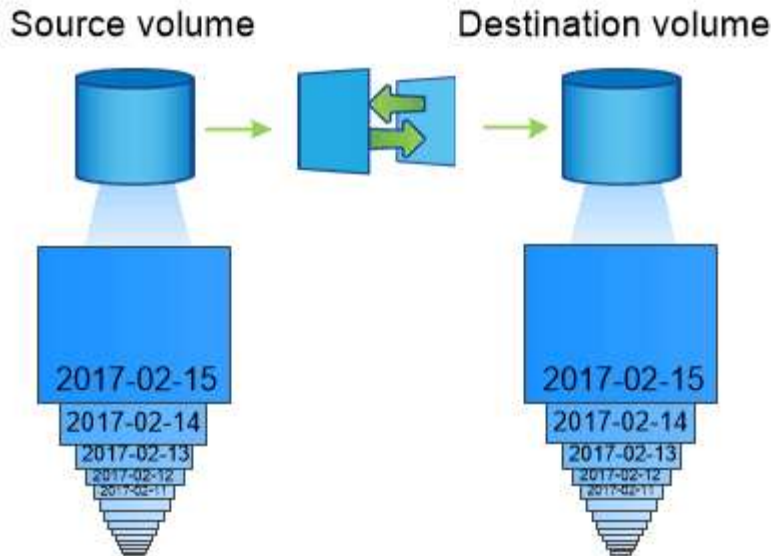
애플리케이션의 영구 볼륨의 경우, 대상 SVM이 활성화될 때 Trident가 프로비저닝한 모든 볼륨이 데이터 제공을 시작합니다. 위에서 설명한 단계를 사용하여 대상 측에 Kubernetes 클러스터를 설정한 후에는 모든 구축과 포드가 시작되고 패키지 애플리케이션은 문제 없이 실행되어야 합니다.

SnapMirror 볼륨 복제

ONTAP SnapMirror 볼륨 복제는 재해 복구 기능으로, 볼륨 레벨의 운영 스토리지에서 대상 스토리지로 페일오버할 수 있도록 지원합니다. SnapMirror는 스냅샷을 동기화하여 보조 스토리지에 운영 스토리지의 볼륨 복제본 또는 미러를 생성합니다.

다음은 ONTAP SnapMirror 볼륨 복제 설정 단계에 대한 개요입니다.

- 볼륨이 상주하는 클러스터와 볼륨의 데이터를 제공하는 SVM 간에 피어링을 설정합니다.
- 관계의 동작을 제어하고 해당 관계에 대한 구성 특성을 지정하는 SnapMirror 정책을 생성합니다.
- 를 사용하여 타겟 볼륨과 소스 볼륨 사이에 SnapMirror 관계를 생성합니다[`snapmirror create ^`] 명령을 사용하여 적절한 SnapMirror 정책을 할당합니다.
- SnapMirror 관계가 생성된 후 소스 볼륨에서 타겟 볼륨으로의 기본 전송이 완료되도록 관계를 초기화합니다.



Trident를 위한 SnapMirror 볼륨 재해 복구 워크플로우

다음 단계에서는 Astra Trident를 사용하여 단일 마스터 Kubernetes 클러스터를 복구하는 방법을 설명합니다.

1. 재해가 발생할 경우 예약된 SnapMirror 전송을 모두 중지하고 진행 중인 SnapMirror 전송을 모두 중단하십시오. 대상 볼륨이 읽기/쓰기가 되도록 대상 볼륨과 소스 볼륨 간의 복제 관계를 중단하십시오.
2. 대상 SVM에서 Kubernetes가 포함된 볼륨을 마운트합니다 etcd 호스트에 데이터 파일 및 인증서를 제공하며, 마스터 노드로 설정됩니다.
3. 에서 Kubernetes 클러스터와 관련된 모든 필수 인증서를 복사합니다 /etc/kubernetes/pki 그리고 etcd member 파일 위치 /var/lib/etcd.
4. 을 실행하여 Kubernetes 클러스터를 생성합니다 kubeadm init 명령과 함께 --ignore-preflight-errors=DirAvailable-var-lib-etcd 깃발. 호스트 이름은 소스 Kubernetes 클러스터와 같아야 합니다.
5. 를 실행합니다 kubectl get crd 모든 Trident 사용자 지정 리소스가 표시되는지 확인하고 Trident 개체를 검색하여 모든 데이터를 사용할 수 있는지 확인하는 명령입니다.
6. 이전 백엔드를 정리하고 Trident에 새 백엔드를 만듭니다. 새 관리 LIF, 새 SVM 이름 및 대상 SVM의 암호를 지정합니다.

애플리케이션의 영구 볼륨에 대한 재해 복구 워크플로우

다음 단계에서는 재해 발생 시 컨테이너화된 워크로드에 SnapMirror 대상 볼륨을 제공하는 방법을 설명합니다.

1. 예약된 모든 SnapMirror 전송을 중지하고 진행 중인 모든 SnapMirror 전송을 중단합니다. 대상 볼륨이 읽기/쓰기가 되도록 대상 볼륨과 소스 볼륨 간의 복제 관계를 중단하십시오. 소스 SVM의 볼륨에 연결된 PVC를 사용하는 구축을 정리합니다.
2. 위에서 설명한 단계를 사용하여 대상 측에 Kubernetes 클러스터를 설정한 후 Kubernetes 클러스터에서 배포, PVC 및 PV를 정리합니다.
3. Trident에서 새로운 관리 및 데이터 LIF, 새 SVM 이름 및 대상 SVM의 암호를 지정하여 새 백엔드를 생성합니다.
4. Trident 가져오기 기능을 사용하여 새 PVC에 바인딩된 PV로 필요한 볼륨을 가져옵니다.
5. 새로 생성된 PVC와 함께 애플리케이션 배포를 재배포합니다.

Element 스냅샷을 사용하여 데이터 복구

볼륨에 대한 스냅샷 스케줄을 설정하고 필요한 간격으로 스냅샷을 생성하도록 하여 Element 볼륨의 데이터를 백업합니다. Element UI 또는 API를 사용하여 스냅샷 스케줄을 설정해야 합니다. 현재 를 통해 스냅샷 스케줄을 볼륨으로 설정할 수 없습니다 `solidfire-san` 드라이버.

데이터가 손상된 경우 Element UI 또는 API를 사용하여 특정 스냅샷을 선택하고 볼륨을 스냅샷으로 수동으로 롤백할 수 있습니다. 이렇게 하면 스냅샷이 생성된 이후 볼륨에 대한 모든 변경 사항이 복구됩니다.

보안

보안

여기에 나열된 권장 사항을 사용하여 Astra Trident 설치가 안전한지 확인합니다.

자체 네임스페이스에서 **Astra Trident**를 실행합니다

애플리케이션, 애플리케이션 관리자, 사용자 및 관리 애플리케이션에서 Astra Trident 객체 정의 또는 Pod에 액세스하여 안정적인 스토리지를 보장하고 잠재적인 악성 활동을 차단하는 것이 중요합니다.

다른 애플리케이션과 사용자를 Astra Trident에서 분리하려면 항상 고유한 Kubernetes 네임스페이스에 Astra Trident를 설치하십시오 (`trident`)를 클릭합니다. Astra Trident를 자체 네임스페이스에 두면 Kubernetes 관리 담당자만 Astra Trident POD와 이름이 같은 CRD 객체에 저장된 아티팩트(예: 백엔드 및 CHAP 암호)에 액세스할 수 있습니다. 관리자만이 Astra Trident 네임스페이스에 액세스하고 에 액세스할 수 있도록 허용해야 합니다 `tridentctl` 응용 프로그램.

ONTAP SAN 백엔드에 CHAP 인증을 사용합니다

Astra Trident는 ONTAP SAN 워크로드에 대한 CHAP 기반 인증을 지원합니다(사용 `ontap-san` 및 `ontap-san-economy` 드라이버). 호스트와 스토리지 백엔드 간의 인증을 위해 Astra Trident와 양방향 CHAP를 사용하는 것이 좋습니다.

SAN 스토리지 드라이버를 사용하는 ONTAP 백엔드의 경우 Astra Trident는 양방향 CHAP를 설정하고 를 통해 CHAP 사용자 이름 및 암호를 관리할 수 있습니다 `tridentctl`. 을 참조하십시오 "[여기](#)" Astra Trident가 ONTAP 백엔드에서 CHAP를 구성하는 방법을 이해합니다.



ONTAP 백엔드에 대한 CHAP 지원은 Trident 20.04 이상에서 사용할 수 있습니다.

NetApp HCI 및 SolidFire 백엔드에서 CHAP 인증을 사용합니다

양방향 CHAP를 구축하여 호스트와 NetApp HCI 및 SolidFire 백엔드 간의 인증을 보장하는 것이 좋습니다. Astra Trident는 테넌트당 2개의 CHAP 암호를 포함하는 비밀 객체를 사용합니다. Trident를 CSI 프로비저닝자로 설치하면 CHAP 암호를 관리하고 에 저장합니다 `tridentvolume` 해당 PV에 대한 CR 개체입니다. PV를 생성할 때 CSI Astra Trident는 CHAP 암호를 사용하여 iSCSI 세션을 시작하고 CHAP를 통해 NetApp HCI 및 SolidFire 시스템과 통신합니다.



CSI Trident에서 생성한 볼륨은 볼륨 액세스 그룹과 연결되지 않습니다.

CSI가 아닌 프론트엔드에서는 작업자 노드의 디바이스로 볼륨을 연결하는 작업을 Kubernetes에서 처리합니다. 볼륨 생성 후 Astra Trident는 NetApp HCI/SolidFire 시스템에 API 호출을 통해 해당 테넌트의 암호가 아직 없는 경우

비밀을 검색합니다. 그런 다음 Astra Trident가 Kubernetes에 비밀을 전달합니다. 각 노드에 위치한 kubelet은 Kubernetes API를 통해 기밀에 액세스하고 이를 사용하여 볼륨에 액세스하는 각 노드와 볼륨이 있는 NetApp HCI/SolidFire 시스템 간에 CHAP를 실행/사용하도록 설정합니다.

NVE와 NAE가 포함된 Astra Trident를 사용하십시오

NetApp ONTAP는 유해 데이터 암호화를 제공하여 디스크를 도난, 반환 또는 용도 변경할 때 중요한 데이터를 보호합니다. 자세한 내용은 을 참조하십시오 ["NetApp 볼륨 암호화 구성 개요"](#).

- 백엔드에서 NAE가 활성화된 경우 Astra Trident에 프로비저닝된 모든 볼륨은 NAE가 활성화됩니다.
- NAE가 백엔드에서 활성화되지 않은 경우, NVE 암호화 플래그를 로 설정하지 않으면 Astra Trident에 프로비저닝된 모든 볼륨은 NVE를 사용할 수 있습니다 `false` 백엔드 구성

NAE 지원 백엔드의 Astra Trident에 생성된 볼륨은 NVE 또는 NAE 암호화여야 합니다.



- NVE 암호화 플래그를 로 설정할 수 있습니다 `true` Trident 백엔드 구성에서 NAE 암호화를 재정의하고 볼륨별로 특정 암호화 키를 사용합니다.
- NVE 암호화 플래그를 로 설정합니다 `false` NAE 지원 백엔드에서 NAE 지원 볼륨을 생성합니다. NVE 암호화 플래그를 로 설정하여 NAE 암호화를 비활성화할 수 없습니다 `false`.

- NVE 암호화 플래그를 명시적으로 로 설정하여 Astra Trident에서 NVE 볼륨을 수동으로 생성할 수 있습니다 `true`.

백엔드 구성 옵션에 대한 자세한 내용은 다음을 참조하십시오.

- ["ONTAP SAN 구성 옵션"](#)
- ["ONTAP NAS 구성 옵션"](#)

Linux 통합 키 설정(LUKS)

LUKS(Linux 통합 키 설정)를 활성화하여 Astra Trident에서 ONTAP SAN 및 ONTAP SAN 경제 볼륨을 암호화할 수 있습니다. Astra Trident는 LUKS 암호화 볼륨에 대한 암호 순환 및 볼륨 확장을 지원합니다.

Astra Trident에서 LUKS 암호화 볼륨은 에서 권장하는 대로 AES-XTS-ai64 cypher 및 모드를 사용합니다 ["NIST"](#).

시작하기 전에

- 작업자 노드에는 Cryptsetup 2.1 이상(3.0 이하)이 설치되어 있어야 합니다. 자세한 내용은 를 참조하십시오 ["Gitlab: cryptsetup"](#).
- 성능상의 이유로 작업자 노드는 AES-NI(Advanced Encryption Standard New Instructions)를 지원하는 것이 좋습니다. AES-NI 지원을 확인하려면 다음 명령을 실행합니다.

```
grep "aes" /proc/cpuinfo
```

아무 것도 반환되지 않으면 프로세서는 AES-NI를 지원하지 않습니다. AES-NI에 대한 자세한 내용은 다음 웹 사이트를 참조하십시오. ["인텔: AES-NI\(Advanced Encryption Standard Instructions\)"](#).

LUKS 암호화를 사용합니다

ONTAP SAN 및 ONTAP SAN 이코노미 볼륨에 대해 Linux 통합 키 설정(LUKS)을 사용하여 볼륨별 호스트 측 암호화를 활성화할 수 있습니다.

단계

1. 백엔드 구성에서 LUKS 암호화 속성을 정의합니다. ONTAP SAN의 백엔드 구성 옵션에 대한 자세한 내용은 을 참조하십시오 ["ONTAP SAN 구성 옵션"](#).

```
"storage": [  
  {  
    "labels":{"luks": "true"},  
    "zone":"us_east_1a",  
    "defaults": {  
      "luksEncryption": "true"  
    }  
  },  
  {  
    "labels":{"luks": "false"},  
    "zone":"us_east_1a",  
    "defaults": {  
      "luksEncryption": "false"  
    }  
  },  
]
```

2. 사용 `parameters.selector` LUKS 암호화를 사용하여 스토리지 풀을 정의합니다. 예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1  
kind: StorageClass  
metadata:  
  name: luks  
provisioner: netapp.io/trident  
parameters:  
  selector: "luks=true"  
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}  
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKS 암호를 포함하는 암호를 생성합니다. 예를 들면 다음과 같습니다.

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

제한 사항

LUKS - 암호화된 볼륨은 ONTAP 중복 제거 및 압축을 활용할 수 없습니다.

LUKS 암호를 회전합니다

LUKS 암호를 회전하고 회전을 확인할 수 있습니다.



볼륨, 스냅샷 또는 비밀이 더 이상 참조하지 않음을 확인할 때까지 암호문을 잊지 마십시오. 참조된 암호가 손실된 경우 볼륨을 마운트할 수 없으며 데이터가 암호화된 상태로 유지되고 액세스할 수 없게 됩니다.

이 작업에 대해

LUKS 암호 회전은 새 LUKS 암호를 지정한 후 볼륨을 마운트하는 POD가 생성될 때 발생합니다. 새 POD를 생성할 때 Astra Trident는 볼륨의 LUKS 암호를 비밀의 활성 패스프레이즈(passphrase)와 비교합니다.

- 볼륨의 암호가 비밀의 활성 암호와 일치하지 않으면 회전이 발생합니다.
- 볼륨의 암호가 비밀의 활성 암호와 일치하면 가 됩니다 `previous-luks-passphrase` 매개 변수는 무시됩니다.

단계

1. 를 추가합니다 `node-publish-secret-name` 및 `node-publish-secret-namespace` `StorageClass` 매개 변수입니다. 예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. 볼륨 또는 스냅샷에서 기존 암호를 식별합니다.

볼륨

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames: ["A"]
```

스냅샷

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames: ["A"]
```

3. 볼륨에 대한 LUKS 암호를 업데이트하여 새 암호 및 이전 암호 문구를 지정합니다. 확인합니다 `previous-luks-passphrase-name` 및 `previous-luks-passphrase` 이전 패스프레이즈를 일치시킵니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. 볼륨을 마운트하는 새 포드를 생성합니다. 이 작업은 회전을 시작하는 데 필요합니다.

5. 패스프레이즈가 회전되었는지 확인합니다.

볼륨

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames: ["B"]
```

스냅샷

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames: ["B"]
```

결과

볼륨과 스냅샷에 새 암호문만 반환되면 암호가 회전되었습니다.



예를 들어, 두 개의 암호 구문이 반환되는 경우 `luksPassphraseNames: ["B", "A"]`, 회전이 완료되지 않았습니다. 새 포드를 트리거하여 회전을 완료할 수 있습니다.

볼륨 확장을 설정합니다

LUKS 암호화 볼륨에서 볼륨 확장을 활성화할 수 있습니다.

단계

1. 를 활성화합니다 `CSINodeExpandSecret` 기능 게이트(베타 1.25+). 을 참조하십시오 ["Kubernetes 1.25: CSI 볼륨의 노드 기반 확장에 비밀을 사용합니다"](#) 를 참조하십시오.
2. 를 추가합니다 `node-expand-secret-name` 및 `node-expand-secret-namespace` `StorageClass` 매개 변수입니다. 예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks- $\{pvc.name\}$ 
  csi.storage.k8s.io/node-stage-secret-namespace:  $\{pvc.namespace\}$ 
  csi.storage.k8s.io/node-expand-secret-name: luks- $\{pvc.name\}$ 
  csi.storage.k8s.io/node-expand-secret-namespace:  $\{pvc.namespace\}$ 
allowVolumeExpansion: true
```

결과

온라인 저장소 확장을 시작할 때 kubelet은 적절한 자격 증명을 드라이버에 전달합니다.

참조하십시오

Astra Trident 포트

Astra Trident가 통신에 사용하는 포트에 대해 자세히 알아보십시오.

Astra Trident 포트

Astra Trident는 다음 포트를 통해 통신합니다.

포트	목적
8443	백채널 HTTPS
8001입니다	Prometheus 메트릭 엔드포인트
8000	Trident REST 서버
17546	Trident 디포드에 사용되는 활성화/준비 프로브 포트



을 사용하여 설치하는 동안 활성화/준비 프로브 포트를 변경할 수 있습니다 `--probe-port` 깃발. 이 포트가 작업자 노드의 다른 프로세스에서 사용되지 않도록 하는 것이 중요합니다.

Astra Trident REST API

있습니다 **"tridentctl 명령 및 옵션"** Astra Trident REST API와 상호 작용하는 가장 쉬운 방법은 REST 엔드포인트를 직접 사용하는 것입니다.

REST API 사용 시기

REST API는 Kubernetes가 아닌 구축 환경에서 Astra Trident를 독립 실행형 바이너리로 사용하는 고급 설치에 유용합니다.

더 나은 보안을 위해 Astra Trident를 사용해 보십시오 REST API 는 POD 내에서 실행할 때 기본적으로 localhost로 제한됩니다. 이 동작을 변경하려면 Astra Trident를 설정해야 합니다 `-address` 인수가 해당 POD 구성의 인수입니다.

REST API 사용

API는 다음과 같이 작동합니다.

GET

- GET `<trident-address>/trident/v1/<object-type>`: 해당 형식의 모든 개체를 나열합니다.
- GET `<trident-address>/trident/v1/<object-type>/<object-name>`(Control 에서 상속됨): 명명된 개체의 세부 정보를 가져옵니다.

POST

POST <trident-address>/trident/v1/<object-type>: 지정된 형식의 개체를 만듭니다.

- 생성할 개체의 JSON 구성이 필요합니다. 각 개체 유형의 사양은 링크 [tridentctl.html](#) 을 참조하십시오 [tridentctl 명령 및 옵션].
- 개체가 이미 있는 경우 동작이 달라집니다. backends는 기존 개체를 업데이트하지만 다른 모든 개체 형식은 작업에 실패합니다.

DELETE

DELETE <trident-address>/trident/v1/<object-type>/<object-name>: 명명된 리소스를 삭제합니다.



백엔드 또는 스토리지 클래스와 연결된 볼륨은 계속 존재하므로 별도로 삭제해야 합니다. 자세한 내용은 링크: [tridentctl.html](#) 을 참조하십시오 [tridentctl 명령 및 옵션].

이러한 API의 호출 방법에 대한 예제를 보려면 디버그를 전달합니다 (-d) 깃발. 자세한 내용은 링크: [tridentctl.html](#) 을 참조하십시오 [tridentctl 명령 및 옵션].

명령줄 옵션

Astra Trident는 Trident Orchestrator를 위한 여러 명령줄 옵션을 제공합니다. 이러한 옵션을 사용하여 배포를 수정할 수 있습니다.

로깅

- -debug: 디버깅 출력을 활성화합니다.
- -loglevel <level>: 로깅 수준을 설정합니다(debug, info, warn, error, fatal). 기본적으로 info(정보) 가 사용됩니다.

쿠버네티스

- -k8s_pod: 이 옵션 또는 을 사용합니다 -k8s_api_server Kubernetes 지원을 사용하도록 설정하십시오. 이 설정을 사용하면 Trident에서 포함된 POD의 Kubernetes 서비스 계정 자격 증명을 사용하여 API 서버에 연락합니다. Trident가 서비스 계정이 활성화된 Kubernetes 클러스터에서 POD로 실행되는 경우에만 작동합니다.
- -k8s_api_server <insecure-address:insecure-port>: 이 옵션 또는 을 사용합니다 -k8s_pod Kubernetes 지원을 사용하도록 설정하십시오. Trident가 지정된 경우 제공된 비보안 주소 및 포트를 사용하여 Kubernetes API 서버에 연결합니다. 따라서 Trident를 POD 외부에 배포할 수 있지만 API 서버에 대한 비보안 연결만 지원합니다. 를 사용하여 포드에 Trident를 배포하여 안전하게 연결합니다 -k8s_pod 옵션을 선택합니다.
- -k8s_config_path <file>: 필수. KubeConfig 파일에 대한 이 경로를 지정해야 합니다.

Docker 를 참조하십시오

- -volume_driver <name>: Docker 플러그인을 등록할 때 사용되는 드라이버 이름입니다. 기본값은 입니다 netapp.
- -driver_port <port-number>: UNIX 도메인 소켓이 아닌 이 포트에서 수신 대기하십시오.

- `-config <file>` 필수: 백엔드 구성 파일에 대한 이 경로를 지정해야 합니다.

휴식

- `-address <ip-or-host>`: Trident의 REST 서버가 수신할 주소를 지정합니다. 기본값은 localhost입니다. localhost에서 듣거나 Kubernetes Pod에서 실행 중인 경우, REST 인터페이스는 Pod 외부에서 직접 액세스할 수 없습니다. 사용 `-address ""` REST 인터페이스를 POD IP 주소에서 액세스할 수 있도록 합니다.



Trident REST 인터페이스는 127.0.0.1(IPv4의 경우) 또는 `:::1`(IPv6의 경우)에서만 수신 및 서비스하도록 구성할 수 있습니다.

- `-port <port-number>`: Trident의 REST 서버가 수신할 포트를 지정합니다. 기본값은 8000입니다.
- `-rest`: REST 인터페이스를 활성화합니다. 기본값은 true 입니다.

Kubernetes와 통합된 NetApp 제품

NetApp 스토리지 제품 포트폴리오는 Kubernetes 클러스터의 다양한 측면과 통합되어 고급 데이터 관리 기능을 제공하며, Kubernetes 구축의 기능, 기능, 성능 및 가용성을 향상합니다.

아스트라

"아스트라" 퍼블릭 클라우드와 온프레미스 모두에서 Kubernetes에서 실행되는 데이터 리치 컨테이너 워크로드를 손쉽게 관리, 보호 및 이동할 수 있습니다. Astra는 퍼블릭 클라우드 및 온프레미스에 있는 NetApp의 검증되고 광범위한 스토리지 포트폴리오의 Trident를 사용하여 영구 컨테이너 스토리지를 프로비저닝하고 제공합니다. 또한, 스냅샷, 백업 및 복원, 활동 로그, 데이터 보호, 재해/데이터 복구, 데이터 감사, Kubernetes 워크로드의 마이그레이션 사용 사례를 위한 액티브 클론 복제와 같은 풍부한 고급 애플리케이션 인식 데이터 관리 기능을 제공합니다.

ONTAP

ONTAP는 모든 애플리케이션에 고급 데이터 관리 기능을 제공하는 NetApp의 다중 프로토콜 통합 스토리지 운영 체제입니다. ONTAP 시스템은 All-Flash, 하이브리드 또는 All-HDD 구성을 제공하며 엔지니어링 하드웨어(FAS 및 AFF), 화이트박스(ONTAP Select), 클라우드 전용(Cloud Volumes ONTAP) 등 다양한 구축 모델을 제공합니다.



Trident는 위에 언급된 모든 ONTAP 배포 모델을 지원합니다.

Cloud Volumes ONTAP

"Cloud Volumes ONTAP" 는 클라우드에서 ONTAP 데이터 관리 소프트웨어를 실행하는 소프트웨어 전용 스토리지 어플라이언스입니다. Cloud Volumes ONTAP를 운영 워크로드, 재해 복구, DevOps, 파일 공유 및 데이터베이스 관리에 사용할 수 있습니다. 스토리지 효율성, 고가용성, 데이터 복제, 데이터 계층화, 애플리케이션 정합성을 보장함으로써 엔터프라이즈 스토리지를 클라우드로 확장합니다.

NetApp ONTAP용 Amazon FSx

"NetApp ONTAP용 Amazon FSx" 은 완벽하게 관리되는 AWS 서비스로, 고객이 NetApp의 ONTAP 스토리지 운영 체제에서 제공하는 파일 시스템을 시작하고 실행할 수 있도록 지원합니다. ONTAP용 FSx를 사용하면 고객이 익숙한 NetApp 기능, 성능 및 관리 기능을 활용하는 동시에 AWS에 데이터를 저장할 때 간편성, 민첩성, 보안, 확장성을 활용할 수 있습니다. ONTAP용 FSx는 ONTAP의 다양한 파일 시스템 기능과 관리 API를 지원합니다.

Element 소프트웨어

"요소" 스토리지 관리자가 성능을 보장하고 단순화된 스토리지 설치 공간을 활용하여 워크로드를 통합할 수 있도록 지원합니다. Element는 스토리지 관리의 모든 측면을 자동화하는 API와 결합되어 스토리지 관리자가 더 적은 노력으로 더 많은 작업을 수행할 수 있게 합니다.

NetApp HCI

"NetApp HCI" 일상적인 작업을 자동화하고 인프라 관리자가 보다 중요한 기능에 집중할 수 있도록 하여 데이터 센터의 관리 및 확장을 단순화합니다.

NetApp HCI는 Trident에서 완전히 지원합니다. Trident는 컨테이너화된 애플리케이션에 대한 스토리지 장치를 기본 NetApp HCI 스토리지 플랫폼에 직접 프로비저닝 및 관리할 수 있습니다.

Azure NetApp Files

"Azure NetApp Files" NetApp에서 제공하는 엔터프라이즈급 Azure 파일 공유 서비스입니다. Azure에서 기본적으로 가장 까다로운 파일 기반 워크로드를 실행하고 NetApp에서 기대하는 성능 및 강력한 데이터 관리를 제공할 수 있습니다.

Google Cloud용 Cloud Volumes Service

"Google Cloud용 NetApp Cloud Volumes Service" 는 NFS 및 SMB를 통해 NAS 볼륨을 All-Flash 성능으로 제공하는 클라우드 네이티브 파일 서비스입니다. 이 서비스를 사용하면 기존 애플리케이션을 포함한 모든 워크로드를 GCP 클라우드에서 실행할 수 있습니다. GCE(Google Compute Engine) 인스턴스에 대한 일관된 고성능, 즉각적인 복제, 데이터 보호 및 보안 액세스를 제공하는 완전 관리형 서비스를 제공합니다.

Kubernetes 및 Trident 오브젝트

REST API를 사용하여 리소스 객체를 읽고 쓰면서 Kubernetes 및 Trident와 상호 작용할 수 있습니다. Kubernetes 및 Trident, Trident 및 스토리지와 Kubernetes 및 스토리지 간의 관계를 결정하는 몇 가지 리소스 개체가 있습니다. 이러한 오브젝트 중 일부는 Kubernetes를 통해 관리되며 나머지는 Trident를 통해 관리됩니다.

개체가 서로 어떻게 상호 작용합니까?

오브젝트, 목표 및 상호 작용 방식을 이해하는 가장 쉬운 방법은 Kubernetes 사용자의 스토리지에 대한 단일 요청을 따르는 것입니다.

1. 사용자가 를 만듭니다 PersistentVolumeClaim 신규 요청 중 PersistentVolume Kubernetes를 사용할 경우 특정 크기입니다 StorageClass 이전에 관리자가 구성한 것입니다.
2. Kubernetes를 StorageClass Trident를 프로비저닝자로 식별하고 Trident에 요청된 클래스에 대한 볼륨을 프로비저닝하는 방법을 알려주는 매개 변수를 포함합니다.
3. Trident가 독자적인 모양을 하고 있습니다 StorageClass 일치하는 항목을 식별하는 동일한 이름을 사용합니다 Backends 및 StoragePools 볼륨을 프로비저닝하는 데 사용할 수 있다는 것을 알 수 있습니다.
4. Trident가 일치하는 백엔드에 스토리지를 프로비저닝하고 두 개의 오브젝트, 즉 A를 생성합니다 PersistentVolume Kubernetes에서 볼륨 찾기, 마운트 및 처리 방법과 Trident의 볼륨을 사용하여 간의 관계를 유지하는 방법을 Kubernetes에서 알려줍니다 PersistentVolume 및 실제 스토리지를 누릅니다.

5. Kubernetes에서 에 바인딩합니다 PersistentVolumeClaim 새로운 제품으로 PersistentVolume. 를 포함하는 Pod입니다 PersistentVolumeClaim 실행 중인 모든 호스트에 PersistentVolume을 마운트합니다.
6. 사용자가 를 만듭니다 VolumeSnapshot 를 사용하여 기존 PVC의 VolumeSnapshotClass Trident를 가리키죠.
7. Trident는 PVC와 연결된 볼륨을 식별하고 백엔드에 볼륨의 스냅샷을 생성합니다. 또한 을 생성합니다 VolumeSnapshotContent 이 방법은 Kubernetes에서 스냅샷을 식별하는 방법을 설명합니다.
8. 사용자는 을 만들 수 있습니다 PersistentVolumeClaim 사용 VolumeSnapshot 소스로 사용합니다.
9. Trident는 필요한 스냅샷을 식별하고 과 관련된 동일한 단계를 수행합니다 PersistentVolume 그리고 A Volume.



Kubernetes 객체에 대한 자세한 내용은 를 읽는 것이 좋습니다 "영구 볼륨" 섹션을 참조하십시오.

쿠버네티스 PersistentVolumeClaim 오브젝트

쿠버네티스 PersistentVolumeClaim 개체는 Kubernetes 클러스터 사용자가 만든 스토리지 요청입니다.

Trident는 표준 사양 외에도 사용자가 백엔드 구성에서 설정한 기본값을 무효화하려는 경우 다음 볼륨별 주석을 지정할 수 있도록 합니다.

주석	볼륨 옵션	지원되는 드라이버
trident.netapp.io/fileSystem	파일 시스템	ONTAP-SAN, solidfire-SAN, ONTAP-SAN - 경제성
trident.netapp.io/cloneFromPVC	CloneSourceVolume	ONTAP-NAS, ONTAP-SAN, solidfire-SAN, Azure-NetApp-파일, GCP-CV, ONTAP-SAN - 경제성
trident.netapp.io/splitOnClone	SplitOnClone 을 참조하십시오	ONTAP-NAS, ONTAP-SAN
trident.netapp.io/protocol	프로토콜	모두
trident.netapp.io/exportPolicy	내보내기 정책	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup
trident.netapp.io/snapshotPolicy	스냅샷 정책	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup, ONTAP-SAN
trident.netapp.io/snapshotReserve	snapshotReserve	ONTAP-NAS, ONTAP-NAS-flexgroup, ONTAP-SAN, GCP-CV
trident.netapp.io/snapshotDirectory	스냅샷 디렉토리	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup
trident.netapp.io/blockSize	블록 크기	solidfire-SAN

생성된 PV에 가 있는 경우 Delete 부가세 반환 청구액 정책인 Trident는 PV가 릴리스될 때(즉, 사용자가 PVC를 삭제할 때) PV와 보조 볼륨을 모두 삭제합니다. 삭제 작업이 실패할 경우 Trident는 PV를 해당 상태로 표시하고 성공할 때까지 또는 PV를 수동으로 삭제할 때까지 주기적으로 작업을 다시 시도합니다. PV에서 를 사용하는 경우 Retain Trident는 이 정책을 무시하고 관리자가 Kubernetes 및 백엔드에서 이를 정리하여 제거하기 전에 볼륨을 백업 또는

검사할 수 있다고 가정합니다. PV를 삭제해도 Trident에서 백업 볼륨을 삭제하지 않습니다. REST API를 사용하여 제거해야 합니다 (tridentctl)를 클릭합니다.

Trident는 CSI 사양을 사용하여 볼륨 스냅샷 생성을 지원합니다. 볼륨 스냅샷을 생성하고 이를 데이터 소스로 사용하여 기존 PVC를 복제할 수 있습니다. 이렇게 하면 PVS의 시점 복제본을 스냅샷 형태로 Kubernetes에 표시할 수 있습니다. 그런 다음 스냅샷을 사용하여 새 PVS를 생성할 수 있습니다. 한 번 살펴보십시오 On-Demand Volume Snapshots 어떻게 작동되는지 확인하십시오.

Trident는 도 제공합니다 cloneFromPVC 및 splitOnClone 클론 생성을 위한 주석. 이러한 주석을 사용하여 Kubernetes 1.13 이하 버전에서 CSI 구현을 사용하지 않고 PVC를 복제하거나 Kubernetes 릴리스가 베타 볼륨 스냅샷(Kubernetes 1.16 이하)을 지원하지 않을 경우 사용할 수 있습니다. Trident 19.10은 PVC를 통한 클론 생성을 위한 CSI 워크플로우를 지원합니다.



를 사용할 수 있습니다 cloneFromPVC 및 splitOnClone CSI Trident 및 기존 비 CSI 프론트엔드 주석이 포함된 주석.

다음은 사용자가 이미 PVC를 호출한 경우입니다 mysql`에서 라는 새 PVC를 생성할 수 있습니다 `mysqlclone` 과 같은 주석을 사용합니다 trident.netapp.io/cloneFromPVC: mysql. 이 주석을 설정하면 Trident가 볼륨을 처음부터 프로비저닝하는 대신 MySQL PVC에 해당하는 볼륨을 클론합니다.

다음 사항을 고려하십시오.

- 유휴 볼륨의 클론을 생성하는 것이 좋습니다.
- PVC와 그 클론은 동일한 Kubernetes 네임스페이스에서 동일한 스토리지 클래스를 가져야 합니다.
- 를 사용하여 ontap-nas 및 ontap-san 드라이버는 PVC 주석을 설정하는 것이 좋습니다 trident.netapp.io/splitOnClone 와 함께 제공됩니다 trident.netapp.io/cloneFromPVC. 와 함께 trident.netapp.io/splitOnClone 를 로 설정합니다 true`Trident는 상위 볼륨에서 복제된 볼륨을 분할하여 복제된 볼륨의 라이프사이클을 완전히 상위 볼륨에서 분리함으로써 일부 스토리지 효율성을 잃지 않습니다. 설정 안 합니다 `trident.netapp.io/splitOnClone 또는 로 설정합니다 false 상위 볼륨과 클론 볼륨 간의 종속성을 생성하여 백엔드에서 공간 소비를 줄임으로써 클론이 먼저 삭제되지 않는 한 상위 볼륨을 삭제할 수 없습니다. 클론을 분할하는 것이 올바른 시나리오는 빈 데이터베이스 볼륨을 복제하여 볼륨과 해당 클론이 크게 달라질 것으로 예상되며 ONTAP에서 제공하는 스토리지 효율성의 이점을 얻지 못하는 경우입니다.

를 클릭합니다 sample-input 디렉토리에는 Trident와 함께 사용할 PVC 정의의 예가 포함되어 있습니다. Trident 볼륨과 관련된 매개 변수 및 설정에 대한 자세한 설명은 Trident Volume 개체를 참조하십시오.

쿠버네티스 PersistentVolume 오브젝트

쿠버네티스 PersistentVolume 객체는 Kubernetes 클러스터에서 사용할 수 있는 스토리지 부분을 나타냅니다. 사용 포드와 독립적인 라이프 사이클이 있습니다.



Trident가 작성합니다 PersistentVolume 제공하는 볼륨을 기반으로 하여 Kubernetes 클러스터에 자동으로 개체를 등록하고 등록합니다. 스스로 관리할 수 없습니다.

Trident를 참조하는 PVC를 만들 때 StorageClass, Trident는 해당 저장소 클래스를 사용하여 새 볼륨을 프로비저닝하고 해당 볼륨에 대한 새 PV를 등록합니다. 프로비저닝 볼륨과 해당 PV를 구성할 때 Trident는 다음 규칙을 따릅니다.

- Trident는 Kubernetes의 PV 이름과 스토리지 프로비저닝에 사용되는 내부 이름을 생성합니다. 두 경우 모두 이름은 해당 범위에서 고유합니다.

- 볼륨의 크기는 플랫폼에 따라 가장 가까운 할당 가능한 수량으로 반올림될 수 있지만 PVC에서 요청된 크기와 최대한 가깝게 일치합니다.

쿠버네티스 StorageClass 오브젝트

쿠버네티스 StorageClass 오브젝트는 의 이름으로 지정됩니다 PersistentVolumeClaims 속성 집합을 사용하여 스토리지를 프로비저닝합니다. 스토리지 클래스 자체는 사용할 구축 소유자를 식별하고 프로비저닝이 이해할 수 있는 조건으로 해당 자산 세트를 정의합니다.

관리자가 만들고 관리해야 하는 두 가지 기본 개체 중 하나입니다. 다른 하나는 Trident 백엔드 객체입니다.

쿠버네티스 StorageClass Trident를 사용하는 개체의 모양은 다음과 같습니다.

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate

```

이러한 매개 변수는 Trident에만 해당되며 Trident에 클래스에 볼륨을 프로비저닝하는 방법을 알려줍니다.

스토리지 클래스 매개 변수는 다음과 같습니다.

속성	유형	필수 요소입니다	설명
속성	[string] 문자열을 매핑합니다	아니요	아래의 특성 섹션을 참조하십시오
스토리지 풀	Map [string] StringList 입니다	아니요	내의 스토리지 풀 목록에 백엔드 이름 매핑
추가 StoragePools	Map [string] StringList 입니다	아니요	내의 스토리지 풀 목록에 백엔드 이름 매핑
excludeStoragePools를 참조하십시오	Map [string] StringList 입니다	아니요	내의 스토리지 풀 목록에 백엔드 이름 매핑

스토리지 속성 및 가능한 값은 스토리지 풀 선택 특성 및 Kubernetes 속성으로 분류할 수 있습니다.

스토리지 풀 선택 특성입니다

이러한 매개 변수는 지정된 유형의 볼륨을 프로비저닝하는 데 사용해야 하는 Trident 관리 스토리지 풀을 결정합니다.

속성	유형	값	제공합니다	요청하십시오	에 의해 지원됩니다
미디어 ¹	문자열	HDD, 하이브리드, SSD	풀에는 이 유형의 미디어가 포함되어 있으며, 하이브리드는 둘 모두를 의미합니다	지정된 미디어 유형입니다	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup, ONTAP-SAN, solidfire-SAN
프로비저닝 유형	문자열	얇고 두껍습니다	풀은 이 프로비저닝 방법을 지원합니다	프로비저닝 방법이 지정되었습니다	Thick: All ONTAP; Thin: All ONTAP & solidfire-SAN
백엔드 유형	문자열	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup, ONTAP-SAN, solidfire-SAN, GCP-CV, Azure-NetApp-파일, ONTAP-SAN-이코노미	풀이 이 백엔드 유형에 속합니다	백엔드가 지정되었습니다	모든 드라이버
스냅샷 수	불입니다	참, 거짓	풀은 스냅샷이 있는 볼륨을 지원합니다	스냅샷이 활성화된 볼륨	ONTAP-NAS, ONTAP-SAN, solidfire-SAN, GCP-CV
복제	불입니다	참, 거짓	풀은 볼륨 클론을 지원합니다	클론이 활성화된 볼륨	ONTAP-NAS, ONTAP-SAN, solidfire-SAN, GCP-CV
암호화	불입니다	참, 거짓	풀은 암호화된 볼륨을 지원합니다	암호화가 활성화된 볼륨입니다	ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroups, ONTAP-SAN
IOPS	내부	양의 정수입니다	풀은 이 범위에서 IOPS를 보장할 수 있습니다	볼륨은 이러한 IOPS를 보장합니다	solidfire-SAN

¹: ONTAP Select 시스템에서 지원되지 않습니다

대부분의 경우 요청된 값이 프로비저닝에 직접적인 영향을 미치며, 예를 들어 일반 프로비저닝을 요청하면 볼륨이 걸쭉하게 프로비저닝됩니다. 하지만 Element 스토리지 풀은 제공된 IOPS 최소 및 최대값을 사용하여 요청된 값이 아닌 QoS 값을 설정합니다. 이 경우 요청된 값은 스토리지 풀을 선택하는 데만 사용됩니다.

을 사용하는 것이 가장 좋습니다 attributes 단독으로 특정 클래스의 요구사항을 충족하는 데 필요한 스토리지의

품질을 모델링합니다. Trident는 의 `_ALL_` 과 일치하는 스토리지 풀을 자동으로 검색하여 선택합니다 `attributes` 지정할 수 있습니다.

을(를) 사용할 수 없는 경우 `attributes` 클래스에 맞는 풀을 자동으로 선택하려면 `l` 를 사용할 수 있습니다 `storagePools` 및 `additionalStoragePools` 풀을 더 세분화하거나 특정 풀 세트를 선택하기 위한 매개 변수입니다.

를 사용할 수 있습니다 `storagePools` 매개 변수를 사용하여 지정된 모든 풀과 일치하는 풀 세트를 추가로 제한합니다 `attributes`. 즉, Trident는 로 식별된 풀의 교차를 사용합니다 `attributes` 및 `storagePools` 프로비저닝에 필요한 매개 변수입니다. 매개 변수만 사용하거나 둘 다 함께 사용할 수 있습니다.

를 사용할 수 있습니다 `additionalStoragePools` 에서 선택한 풀에 관계없이 Trident가 프로비저닝에 사용하는 풀 세트를 확장하는 매개 변수입니다 `attributes` 및 `storagePools` 매개 변수.

를 사용할 수 있습니다 `excludeStoragePools` Trident가 프로비저닝에 사용하는 풀 세트를 필터링하는 매개 변수입니다. 이 매개 변수를 사용하면 일치하는 풀이 모두 제거됩니다.

에 있습니다 `storagePools` 및 `additionalStoragePools` 매개 변수, 각 항목은 풀을 사용합니다 `<backend>:<storagePoolList>`, 위치 `<storagePoolList>` 는 지정된 백엔드에 대해 심표로 구분된 스토리지 풀 목록입니다. 예를 들어, 의 값을 입력합니다 `additionalStoragePools` 있을 것입니다 `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`. 이러한 목록에는 백엔드 및 목록 값 모두에 대한 regex 값이 적용됩니다. 을 사용할 수 있습니다 `tridentctl get backend` 백엔드 및 해당 풀의 목록을 가져옵니다.

Kubernetes 특성

이러한 특성은 동적 프로비저닝 중 Trident가 스토리지 풀/백엔드를 선택하는 데 아무런 영향을 주지 않습니다. 대신 이러한 특성은 Kubernetes 영구 볼륨에서 지원하는 매개 변수만 제공합니다. 작업자 노드는 파일 시스템 생성 작업을 담당하며 `xfsprogs`와 같은 파일 시스템 유틸리티가 필요할 수 있습니다.

속성	유형	값	설명	관련 드라이버	Kubernetes 버전
<code>fsType</code> 입니다	문자열	<code>ext4, ext3, xfs</code> 등	블록 볼륨의 파일 시스템 유형입니다	<code>solidfire-SAN, ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup, ONTAP-SAN, ONTAP-SAN -경제성</code>	모두
<code>allowVolumeExpansion</code>	부울	참, 거짓	PVC 크기 증가에 대한 지원을 활성화 또는 비활성화합니다	<code>ONTAP-NAS, ONTAP-NAS-이코노미, ONTAP-NAS-Flexgroup, ONTAP-SAN, ONTAP-SAN-이코노미, solidfire-SAN, GCP-CV, Azure-NetApp-파일</code>	1.11+

속성	유형	값	설명	관련 드라이버	Kubernetes 버전
볼륨BindingMode 를 선택합니다	문자열	Immediate, WaitForFirstCon sumer입니다	볼륨 바인딩 및 동적 프로비저닝이 수행될 시기를 선택합니다	모두	1.19-1.26

- 를 클릭합니다 `fsType` 매개 변수는 SAN LUN에 대해 원하는 파일 시스템 유형을 제어하는 데 사용됩니다. 또한 Kubernetes는 의 존재 여부를 사용합니다 `fsType` 파일 시스템이 있음을 나타내는 스토리지 클래스에 있습니다. 볼륨 소유권은 를 사용하여 제어할 수 있습니다 `fsGroup` POD의 보안 컨텍스트는 에만 해당됩니다 `fsType` 가 설정됩니다. 을 참조하십시오 ["Kubernetes: Pod 또는 컨테이너의 보안 컨텍스트를 구성합니다"](#) 를 사용하여 볼륨 소유권을 설정하는 방법에 대한 개요를 보려면 를 참조하십시오 `fsGroup` 상황. Kubernetes가 에 적용됩니다 `fsGroup` 다음 경우에만 값:



- `fsType` 스토리지 클래스에서 설정됩니다.
- PVC 액세스 모드는 RWO입니다.

NFS 스토리지 드라이버의 경우 파일 시스템이 NFS 내보내기의 일부로 이미 존재합니다. 를 사용합니다 `fsGroup` 스토리지 클래스는 여전히 을 지정해야 합니다 `fsType`. 로 설정할 수 있습니다 `nfs` 또는 `null`이 아닌 값을 입력합니다.

- 을 참조하십시오 ["볼륨 확장"](#) 볼륨 확장에 대한 자세한 내용은 를 참조하십시오.
- Trident 설치 프로그램 번들에는 의 Trident와 함께 사용할 수 있는 여러 가지 예제 스토리지 클래스 정의가 제공됩니다 `sample-input/storage-class-*.yaml`. Kubernetes 스토리지 클래스를 삭제하면 해당 Trident 스토리지 클래스도 삭제됩니다.

쿠버네티스 VolumeSnapshotClass 오브젝트

쿠버네티스 `VolumeSnapshotClass` 개체는 과 유사합니다 `StorageClasses`. 이 기능을 사용하면 여러 스토리지 클래스를 정의할 수 있으며, 스냅샷을 필요한 스냅샷 클래스와 연결하기 위해 볼륨 스냅샷에서 참조할 수 있습니다. 각 볼륨 스냅샷은 단일 볼륨 스냅샷 클래스와 연결됩니다.

A `VolumeSnapshotClass` 스냅샷을 생성하려면 관리자가 정의해야 합니다. 볼륨 스냅샷 클래스는 다음과 같은 정의로 생성됩니다.

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

를 클릭합니다 `driver` 의 볼륨 스냅샷을 요청하는 Kubernetes를 지정합니다 `csi-snapclass` 클래스는 Trident에서 처리합니다. 를 클릭합니다 `deletionPolicy` 스냅샷을 삭제해야 할 때 수행할 작업을 지정합니다. 시기 `deletionPolicy` 가 로 설정되어 있습니다 `Delete` 스냅샷이 삭제되면 스토리지 클러스터의 기본 스냅샷 및 볼륨 스냅샷 객체가 제거됩니다. 또는 로 설정합니다 `Retain` 은 를 의미합니다 `VolumeSnapshotContent` 물리적 스냅샷이 보존됩니다.

쿠버네티스 VolumeSnapshot 오브젝트

쿠버네티스 VolumeSnapshot object는 볼륨의 스냅샷을 생성하는 요청입니다. PVC는 사용자가 볼륨에 대해 요청하는 것처럼 볼륨 스냅샷은 사용자가 기존 PVC의 스냅샷을 생성하도록 요청하는 것입니다.

볼륨 스냅샷 요청이 들어오면 Trident는 백엔드의 볼륨에 대한 스냅샷 생성을 자동으로 관리하고 고유한 를 생성하여 스냅샷을 표시합니다

VolumeSnapshotContent 오브젝트. 기존 PVC에서 스냅샷을 생성하고 새 PVC를 생성할 때 스냅샷을 DataSource로 사용할 수 있습니다.



VolumeSnapshot의 생수는 소스 PVC와는 독립적입니다. 소스 PVC가 삭제된 후에도 스냅샷이 지속됩니다. 연관된 스냅샷이 있는 PVC를 삭제할 때 Trident는 이 PVC에 대한 백업 볼륨을 * Deleting * 상태로 표시하지만 완전히 제거하지는 않습니다. 연결된 모든 스냅샷이 삭제되면 볼륨이 제거됩니다.

쿠버네티스 VolumeSnapshotContent 오브젝트

쿠버네티스 VolumeSnapshotContent 개체는 이미 프로비저닝된 볼륨에서 생성된 스냅샷을 나타냅니다. 이는 와 유사합니다 PersistentVolume 및 은 스토리지 클러스터에서 프로비저닝된 스냅샷을 나타냅니다. 과 유사합니다 PersistentVolumeClaim 및 PersistentVolume 객체, 스냅샷이 생성될 때 VolumeSnapshotContent 개체는 에 대한 일대일 매핑을 유지합니다 VolumeSnapshot 스냅샷 생성을 요청한 객체입니다.



Trident가 작성합니다 VolumeSnapshotContent 제공하는 볼륨을 기반으로 하여 Kubernetes 클러스터에 자동으로 개체를 등록하고 등록합니다. 스스로 관리할 수 없습니다.

를 클릭합니다 VolumeSnapshotContent 객체에는 과 같이 스냅샷을 고유하게 식별하는 세부 정보가 포함되어 있습니다 snapshotHandle. 여기 snapshotHandle 은 PV의 이름과 의 이름을 고유하게 조합한 것입니다 VolumeSnapshotContent 오브젝트.

스냅샷 요청이 들어오면 Trident가 백엔드에 스냅샷을 생성합니다. 스냅샷이 생성되면 Trident에서 을 구성합니다 VolumeSnapshotContent Kubernetes API에 스냅샷을 노출합니다.

쿠버네티스 CustomResourceDefinition 오브젝트

Kubernetes 사용자 지정 리소스는 관리자가 정의하며 비슷한 객체를 그룹화하는 데 사용되는 Kubernetes API의 엔드포인트입니다. Kubernetes에서는 오브젝트 컬렉션을 저장하기 위한 사용자 지정 리소스의 생성을 지원합니다. 를 실행하여 이러한 리소스 정의를 가져올 수 있습니다 `kubectl get crds`.

사용자 정의 리소스 정의(CRD) 및 관련 오브젝트 메타데이터는 Kubernetes에서 메타데이터 저장소에 저장됩니다. 따라서 Trident를 위한 별도의 저장소가 필요하지 않습니다.

19.07 릴리즈부터 Trident는 다양한 버전을 사용합니다 CustomResourceDefinition Trident 백 엔드, Trident 스토리지 클래스, Trident 볼륨과 같은 Trident 개체의 ID를 보존할 개체입니다. 이러한 오브젝트는 Trident에서 관리합니다. 또한 CSI 볼륨 스냅샷 프레임워크는 볼륨 스냅샷을 정의하는 데 필요한 일부 CRD를 소개합니다.

CRD는 Kubernetes를 구성하는 것입니다. 위에 정의된 리소스의 객체는 Trident에 의해 생성됩니다. 간단한 예로, 를 사용하여 백엔드를 생성할 수 있습니다 `tridentctl`, 해당 `tridentbackends` CRD 객체는 Kubernetes에서 사용할 수 있도록 생성되었습니다.

다음은 Trident의 CRD에 대해 고려해야 할 몇 가지 사항입니다.

- Trident가 설치되면 일련의 CRD가 생성되어 다른 리소스 유형과 마찬가지로 사용할 수 있습니다.
- Trident의 이전 버전(사용된 버전)에서 업그레이드할 때 etcd 상태를 유지하기 위해), Trident 설치 프로그램이 에서 데이터를 마이그레이션합니다 etcd 키 값 데이터 저장소 및 해당 CRD 개체 생성
- 를 사용하여 Trident를 제거하는 경우 tridentctl uninstall Command, Trident Pod가 삭제되지만 생성된 CRD는 정리되지 않습니다. 을 참조하십시오 ["Trident를 제거합니다"](#) Trident를 완전히 제거하고 처음부터 다시 구성할 수 있는 방법을 이해합니다.

트라이던트 StorageClass 오브젝트

Trident가 Kubernetes에 맞는 스토리지 클래스를 생성합니다 StorageClass 지정하는 개체입니다 `csi.trident.netapp.io/netapp.io/trident` 그들의 공급자 분야. 스토리지 클래스 이름이 Kubernetes의 클래스 이름과 일치합니다 StorageClass 나타내는 개체입니다.



Kubernetes를 사용하면 이러한 오브젝트는 Kubernetes에서 자동으로 생성됩니다 StorageClass Trident를 프로비저닝자로 사용하는 등록이 완료되었습니다.

스토리지 클래스는 볼륨에 대한 일련의 요구 사항으로 구성됩니다. Trident는 이러한 요구 사항을 각 스토리지 풀에 있는 속성과 일치시킵니다. 일치하는 경우 해당 스토리지 풀이 해당 스토리지 클래스를 사용하여 볼륨을 프로비저닝할 수 있는 유효한 타겟입니다.

REST API를 사용하여 스토리지 클래스를 직접 정의하는 스토리지 클래스 구성을 생성할 수 있습니다. 그러나 Kubernetes 배포의 경우 새 Kubernetes 등록 시 Kubernetes가 생성될 것으로 예상합니다 StorageClass 오브젝트.

Trident 백엔드 객체

백엔드는 Trident가 볼륨을 프로비저닝하는 스토리지 공급자를 나타냅니다. 단일 Trident 인스턴스가 원하는 수의 백엔드를 관리할 수 있습니다.



이것은 직접 만들고 관리하는 두 가지 개체 유형 중 하나입니다. 다른 하나는 Kubernetes입니다 StorageClass 오브젝트.

이러한 개체를 구성하는 방법에 대한 자세한 내용은 을 참조하십시오 ["백엔드 구성 중"](#).

트라이던트 StoragePool 오브젝트

스토리지 풀은 각 백엔드에서 용량 할당에 사용할 수 있는 고유한 위치를 나타냅니다. ONTAP의 경우 SVM에 있는 애그리게이트와 대응합니다. NetApp HCI/SolidFire의 경우 관리자 지정 QoS 밴드에 해당합니다. Cloud Volumes Service의 경우 클라우드 공급자 지역에 해당합니다. 각 스토리지 풀에는 고유한 스토리지 특성 세트가 있으며, 이 특성 집합은 성능 특성과 데이터 보호 특성을 정의합니다.

다른 오브젝트와 달리 스토리지 풀 후보 는 항상 자동으로 검색되고 관리됩니다.

트라이던트 Volume 오브젝트

볼륨은 NFS 공유 및 iSCSI LUN과 같은 백엔드 엔드포인트로 구성된 기본 프로비저닝 단위입니다. Kubernetes에서 이러한 항목은 에 직접 대응합니다 PersistentVolumes. 볼륨을 생성할 때 볼륨의 용량을 할당할 수 있는 위치와 크기를 결정하는 스토리지 클래스가 있는지 확인합니다.



Kubernetes에서 이러한 오브젝트는 자동으로 관리됩니다. 프로비저닝 Trident를 보려면 해당 Trident를 확인하십시오.



연결된 스냅샷이 있는 PV를 삭제하면 해당 Trident 볼륨이 * Deleting * 상태로 업데이트됩니다. Trident 볼륨을 삭제하려면 볼륨의 스냅샷을 제거해야 합니다.

볼륨 구성은 프로비저닝된 볼륨에 있어야 하는 속성을 정의합니다.

속성	유형	필수 요소입니다	설명
버전	문자열	아니요	Trident API 버전("1")
이름	문자열	예	생성할 볼륨의 이름입니다
storageClass 를 선택합니다	문자열	예	볼륨을 프로비저닝할 때 사용할 스토리지 클래스입니다
크기	문자열	예	용량 할당할 볼륨의 크기 (바이트)입니다
프로토콜	문자열	아니요	사용할 프로토콜 유형;"파일" 또는 "블록"
내부 이름	문자열	아니요	스토리지 시스템에 있는 객체의 이름으로, Trident에서 생성
CloneSourceVolume	문자열	아니요	ONTAP(NAS, SAN) 및 SolidFire - *: 복제할 볼륨의 이름입니다
SplitOnClone 을 참조하십시오	문자열	아니요	ONTAP(NAS, SAN): 상위 클론에서 클론을 분할합니다
스냅샷 정책	문자열	아니요	ONTAP - *: 사용할 스냅샷 정책
snapshotReserve	문자열	아니요	ONTAP - *: 스냅샷용으로 예약된 볼륨의 비율입니다
내보내기 정책	문자열	아니요	ONTAP-NAS *: 사용할 익스포트 정책
스냅샷 디렉토리	불입니다	아니요	ONTAP-NAS *: 스냅샷 디렉토리가 표시되는지 여부를 나타냅니다
unixPermissions	문자열	아니요	ONTAP-NAS *: 초기 UNIX 권한
블록 크기	문자열	아니요	SolidFire - *: 블록/섹터 크기
파일 시스템	문자열	아니요	파일 시스템 유형입니다

Trident가 생성합니다 `internalName` 볼륨을 생성할 때 이 단계는 두 단계로 구성됩니다. 먼저, 스토리지 접두어 앞에 추가됩니다(기본값 중 하나) `trident` 또는 백엔드 구성의 접두사)를 볼륨 이름에 입력하여 양식 이름을 만듭니다 `<prefix>-<volume-name>`. 그런 다음 백엔드에서 허용되지 않는 문자를 대체하여 이름을 삭제하는 작업을

진행합니다. ONTAP 백엔드의 경우 하이픈을 밑줄로 대체하므로 내부 이름은 이 됩니다 <prefix>_<volume-name>)를 클릭합니다. 요소 백엔드의 경우 밑줄을 하이픈으로 바꿉니다.

볼륨 구성을 사용하여 REST API를 사용하여 볼륨을 직접 프로비저닝할 수 있지만 Kubernetes 배포에서는 대부분의 사용자가 표준 Kubernetes를 사용할 것으로 예상합니다 PersistentVolumeClaim 방법. Trident는 프로비저닝 프로세스의 일부로 이 볼륨 개체를 자동으로 만듭니다.

트라이던트 Snapshot 오브젝트

스냅샷은 볼륨의 시점 복제본으로, 새 볼륨을 용량 할당하거나 복구 상태를 복구하는 데 사용할 수 있습니다. Kubernetes에서 이러한 항목은 예 직접 대응합니다 VolumeSnapshotContent 오브젝트. 각 스냅샷은 스냅샷에 대한 데이터의 소스인 볼륨에 연결됩니다.

각각 Snapshot 개체에는 아래 나열된 속성이 포함됩니다.

속성	유형	필수 요소입니다	설명
버전	문자열	예	Trident API 버전("1")
이름	문자열	예	Trident 스냅샷 개체의 이름입니다
내부 이름	문자열	예	스토리지 시스템의 Trident 스냅샷 개체의 이름입니다
볼륨 이름	문자열	예	스냅샷이 생성된 영구 볼륨의 이름입니다
볼륨 국제 이름	문자열	예	스토리지 시스템에서 연결된 Trident 볼륨 개체의 이름입니다



Kubernetes에서 이러한 오브젝트는 자동으로 관리됩니다. 프로비저닝 Trident를 보려면 해당 Trident를 확인하십시오.

Kubernetes를 사용할 경우 VolumeSnapshot 객체 요청이 생성되면 Trident는 백업 스토리지 시스템에 스냅샷 객체를 생성하여 작동합니다. 를 클릭합니다 internalName 이 스냅샷 개체의 접두어를 결합하여 생성됩니다 snapshot- 를 사용하여 UID 의 VolumeSnapshot 개체(예: snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660)를 클릭합니다. volumeName 및 volumeInternalName 백업 볼륨의 세부 정보를 가져오는 방식으로 채워집니다.

아스트라 트리던트 ResourceQuota 오브젝트

Trident daemonset은 을 사용합니다 system-node-critical 우선 순위 클래스 - Kubernetes에서 가장 높은 우선 순위 클래스 - Astra Trident가 정상 노드 종료 중에 볼륨을 식별 및 정리하고, Trident에서 POD를 사용하여 리소스 압력이 높은 클러스터에서 낮은 우선 순위로 워크로드를 사전 예방할 수 있습니다.

이를 위해 Astra Trident는 을(를) 사용합니다 ResourceQuota Trident 데모에서 "system-node-critical" 우선 순위 클래스가 만족되는지 확인하는 개체입니다. Astra Trident는 구축 및 디멘션 생성 전에 을(를) 찾습니다 ResourceQuota 객체를 검색한 후, 검색되지 않은 경우 적용합니다.

기본 리소스 할당량 및 우선순위 클래스에 대한 추가 제어가 필요한 경우 을 생성할 수 있습니다 custom.yaml 또는 를 구성합니다 ResourceQuota 제어 차트를 사용하는 개체.

다음은 Trident 데모의 우선 순위를 지정하는 'ResourceQuota' 개체의 예입니다.

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
    matchExpressions:
      - operator : In
        scopeName: PriorityClass
        values: ["system-node-critical"]
```

리소스 할당량에 대한 자세한 내용은 을 참조하십시오 ["Kubernetes: 리소스 할당량"](#).

정리 ResourceQuota 설치에 실패한 경우

드문 경우지만 이후 설치가 실패하는 경우가 있습니다 ResourceQuota 객체가 생성되었습니다. 먼저 시도하십시오 **"제거 중"** 그런 다음 다시 설치합니다.

이 기능이 작동하지 않으면 를 수동으로 제거합니다 ResourceQuota 오브젝트.

제거 ResourceQuota

자체 리소스 할당을 제어하려는 경우 Astra Trident를 제거할 수 있습니다 ResourceQuota 다음 명령을 사용하는 개체:

```
kubectl delete quota trident-csi -n trident
```

tridentctl 명령 및 옵션

를 클릭합니다 ["Trident 설치 프로그램 번들"](#) 명령줄 유틸리티, `tridentctl` Astra Trident에 간편하게 액세스할 수 있습니다. 충분한 권한을 가진 Kubernetes 사용자는 이를 사용하여 Astra Trident를 설치할 뿐 아니라 직접 상호 작용하여 Astra Trident Pod가 포함된 네임스페이스를 관리할 수 있습니다.

사용 가능한 명령 및 옵션

사용 정보를 보려면 를 실행합니다 `tridentctl --help`.

사용 가능한 명령 및 전역 옵션은 다음과 같습니다.

Usage:

```
tridentctl [command]
```

사용 가능한 명령:

- ``create`` Astra Trident에 리소스를 추가합니다.
- ``delete`` Astra Trident에서 하나 이상의 리소스를 제거합니다.
- ``get`` Astra Trident에서 하나 이상의 리소스를 제공합니다.
- `help`: 명령에 대한 도움말입니다.
- ``images`` Astra Trident에 필요한 컨테이너 이미지 테이블을 인쇄합니다.
- ``import`` Astra Trident로 기존 리소스를 가져옵니다.
- ``install`` Astra Trident를 설치합니다.
- ``logs`` Astra Trident에서 로그를 인쇄합니다.
- ``send`` Astra Trident에서 리소스를 보냅니다.
- ``uninstall`` Astra Trident를 제거합니다.
- ``update`` Astra Trident에서 리소스를 수정합니다.
- `upgrade`: Astra Trident에서 리소스를 업그레이드합니다.
- ``version`` Astra Trident의 버전을 인쇄하십시오.

플래그:

- ``-d, --debug``: 출력 디버그.
- ``-h, --help``: 도움말을 참조하십시오 ``tridentctl``.
- ``-n, --namespace string``: Astra Trident 배포의 네임스페이스입니다.
- ``-o, --output string``: 출력 형식. `json|YAML|name|wide|ps`(기본값) 중 하나.
- ``-s, --server string``: Astra Trident REST 인터페이스의 주소/포트입니다.



Trident REST 인터페이스는 127.0.0.1(IPv4의 경우) 또는 `:::1`(IPv6의 경우)에서만 수신 및 서비스하도록 구성할 수 있습니다.



Trident REST 인터페이스는 127.0.0.1(IPv4의 경우) 또는 `:::1`(IPv6의 경우)에서만 수신 및 서비스하도록 구성할 수 있습니다.

`create`

를 실행할 수 있습니다 `create` Astra Trident에 리소스를 추가하는 명령입니다.

```
Usage:
  tridentctl create [option]
```

사용 가능한 옵션:

`backend` Astra Trident에 백엔드를 추가합니다.

delete

를 실행할 수 있습니다 delete Astra Trident에서 하나 이상의 리소스를 제거하는 명령입니다.

```
Usage:
  tridentctl delete [option]
```

사용 가능한 옵션:

- `backend` Astra Trident에서 하나 이상의 스토리지 백엔드를 삭제합니다.
- `snapshot` Astra Trident에서 하나 이상의 볼륨 스냅샷을 삭제합니다.
- `storageclass` Astra Trident에서 하나 이상의 스토리지 클래스를 삭제합니다.
- `volume` Astra Trident에서 하나 이상의 스토리지 볼륨을 삭제합니다.

get

를 실행할 수 있습니다 get Astra Trident에서 하나 이상의 리소스를 제공하는 명령입니다.

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

사용 가능한 옵션:

- `backend` Astra Trident에서 하나 이상의 스토리지 백엔드를 받으세요.
- snapshot: Astra Trident에서 하나 이상의 스냅샷을 가져옵니다.
- `storageclass` Astra Trident에서 하나 이상의 스토리지 클래스를 다운로드하십시오.
- `volume` Astra Trident에서 하나 이상의 볼륨을 가져오십시오.

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

images

를 실행할 수 있습니다 images Astra Trident에 필요한 컨테이너 이미지 테이블을 인쇄하려면 플래그를 지정합니다.

```
Usage:
  tridentctl images [flags]
```

플래그: * -h, --help: Help for images.
* -v, --k8s-version string: Kubernetes 클러스터의 의미 있는 버전

`import volume`

를 실행할 수 있습니다 `import volume` 기존 볼륨을 Astra Trident로 가져오는 명령입니다.

```
Usage:
  tridentctl import volume <backendName> <volumeName> [flags]
```

별칭:
volume, v

플래그:

- `--filename string`: YAML 또는 JSON PVC 파일로 이동합니다.
- `-h, --help`: 볼륨에 대한 도움말입니다.
- `--no-manage`: PV/PVC만 생성 볼륨 라이프사이클 관리를 가정하지 마십시오.

`install`

를 실행할 수 있습니다 `install` Astra Trident를 설치하는 플래그입니다.

```
Usage:
  tridentctl install [flags]
```

플래그:

- `--autosupport-image string`: AutoSupport 원격 측정(기본값: "NetApp/트리덴트 자동 지원: 20.07.0")의 컨테이너 이미지입니다.
- `--autosupport-proxy string`: AutoSupport 텔레메트리 전송을 위한 프록시의 주소/포트입니다.
- `--csi`: CSI Trident 설치(Kubernetes 1.13에만 재정의, 기능 게이트 필요)
- `--enable-node-prep`: 노드에 필요한 패키지 설치를 시도합니다.
- `--generate-custom-yaml`: 아무 것도 설치하지 않고 YAML 파일을 생성합니다.
- `-h, --help`: 설치 도움말.
- `--http-request-timeout`: Trident 컨트롤러의 REST API에 대한 HTTP 요청 시간 초과를 재정의합니다 (기본값 1m30s).
- `--image-registry string`: 내부 이미지 레지스트리의 주소/포트입니다.

- `--k8s-timeout duration`` 모든 Kubernetes 작업(기본값 3m0)의 시간 초과.
- `--kubelet-dir string`` kubelet의 내부 상태(기본값 `"/var/lib/kubelet"`)의 호스트 위치입니다.
- `--log-format string`` Astra Trident 로깅 형식(text, json)(기본 `"text"`).
- `--pv string`` Astra Trident에서 사용하는 레거시 PV의 이름입니다. 이 이름이 존재하지 않는지 확인합니다(기본 `"삼중류"`).
- `--pvc string`` Astra Trident에서 사용하는 기존 PVC의 이름입니다. 이 이름이 존재하지 않는지 확인합니다(기본 `"삼중류"`).
- `--silence-autosupport`` AutoSupport 번들을 NetApp에 자동으로 보내지 않습니다(기본값: true).
- `--silent`` 설치하는 동안 대부분의 출력을 비활성화합니다.
- `--trident-image string`` 설치할 Astra Trident 이미지.
- `--use-custom-yaml`` 설정 디렉토리에 있는 기존 YAML 파일을 사용합니다.
- `--use-ipv6`` Astra Trident의 통신에는 IPv6를 사용합니다.

logs

를 실행할 수 있습니다 `logs` Astra Trident의 로그를 인쇄할 플래그입니다.

```
Usage:
  tridentctl logs [flags]
```

플래그:

- `-a, --archive`` 별도로 지정하지 않는 한 모든 로그를 사용하여 지원 아카이브를 생성합니다.
- `-h, --help`` 로그 도움말.
- `-l, --log string`` 표시할 Astra Trident 로그. 트리덴트|auto|트리덴트-operator|all 중 하나(기본 `"자동"`).
- `--node string`` 노드 POD 로그를 수집할 Kubernetes 노드 이름입니다.
- `-p, --previous`` 이전 컨테이너 인스턴스에 대한 로그가 있으면 가져옵니다.
- `--sidecars`` 사이드카 컨테이너의 로그를 가져옵니다.

send

를 실행할 수 있습니다 `send` Astra Trident에서 리소스를 보내는 명령입니다.

```
Usage:
  tridentctl send [option]
```

사용 가능한 옵션:

`autosupport`` AutoSupport 아카이브를 NetApp으로 전송합니다.

uninstall

를 실행할 수 있습니다 `uninstall` Astra Trident를 제거하는 플래그입니다.

```
Usage:
  tridentctl uninstall [flags]
```

플래그: * `-h`, `--help`: 제거 도움말입니다. * `--silent`: 제거 중 대부분의 출력을 비활성화합니다.

update

를 실행할 수 있습니다 `update` Astra Trident에서 리소스를 수정하는 명령입니다.

```
Usage:
  tridentctl update [option]
```

사용 가능한 옵션:

`'backend'` Astra Trident에서 백엔드를 업데이트합니다.

upgrade

를 실행할 수 있습니다 `upgrade` Astra Trident에서 리소스를 업그레이드하는 명령입니다.

```
Usage:
  tridentctl upgrade [option]
```

사용 가능한 옵션:

`volume`: NFS/iSCSI에서 CSI로 하나 이상의 영구 볼륨을 업그레이드합니다.

version

를 실행할 수 있습니다 `version` 플래그를 사용하여 의 버전을 인쇄합니다 `tridentctl` 및 실행 중인 Trident 서비스를 제공합니다.

```
Usage:
  tridentctl version [flags]
```

플래그: * `--client`: 클라이언트 버전만(서버가 필요하지 않음). * `-h`, `--help`: 버전에 대한 도움말입니다.

POD 보안 표준(PSS) 및 보안 컨텍스트 제약(SCC)

Kubernetes Pod 보안 표준(PSS) 및 Pod 보안 정책(PSP)에서 사용 권한 수준을 정의하고

Pod의 동작을 제한합니다. OpenShift Security Context Constraints(SCC)도 OpenShift Kubernetes Engine에 특정한 POD 제한을 정의합니다. 이러한 사용자 지정을 제공하기 위해 Astra Trident는 설치 중에 특정 권한을 활성화합니다. 다음 섹션에서는 Astra Trident에서 설정한 사용 권한에 대해 자세히 설명합니다.



PSS는 Pod 보안 정책(PSP)을 대체합니다. PSP는 Kubernetes v1.21에서 사용되지 않으며 v1.25에서 제거됩니다. 자세한 내용은 을 참조하십시오 "[Kubernetes: 보안](#)".

필요한 **Kubernetes** 보안 컨텍스트 및 관련 필드

권한	설명
특별 권한	CSI를 사용하려면 마운트 지점이 양방향이어야 합니다. 즉, Trident 노드 포드가 권한이 있는 컨테이너를 실행해야 합니다. 자세한 내용은 을 참조하십시오 " Kubernetes: 마운트 전파 ".
호스트 네트워킹	iSCSI 데몬에 필요합니다. <code>iscsiadm</code> iSCSI 마운트를 관리하고 호스트 네트워킹을 사용하여 iSCSI 데몬과 통신합니다.
호스트 IPC	NFS는 IPC(프로세스 간 통신)를 사용하여 NFSD와 통신합니다.
호스트 PID	를 시작해야 합니다 <code>rpc-statd</code> 대해 NFS를 선택합니다. Astra Trident가 호스트 프로세스를 쿼리하여 확인 여부를 결정합니다 <code>rpc-statd</code> NFS 볼륨을 마운트하기 전에 실행 중입니다.
제공합니다	를 클릭합니다 <code>SYS_ADMIN</code> 권한이 있는 컨테이너에 대한 기본 기능의 일부로 기능이 제공됩니다. 예를 들어, Docker는 권한이 있는 컨테이너에 대해 다음 기능을 설정합니다. <code>CapPrm: 0000003fffffffff</code> <code>CapEff: 0000003fffffffff</code>
Seccomp	Seccomp 프로파일은 권한 있는 컨테이너에서 항상 "제한 없음"이므로 Astra Trident에서 활성화할 수 없습니다.
SELinux	OpenShift에서는 권한이 있는 컨테이너가 에서 실행됩니다 <code>spc_t</code> ("슈퍼 프리권한 컨테이너") 도메인 및 권한 없는 컨테이너는 에서 실행됩니다 <code>container_t</code> 도메인. 커짐 <code>containerd</code> , 및 <code>container-selinux</code> 설치된 모든 컨테이너는 에서 실행됩니다 <code>spc_t</code> SELinux를 효과적으로 비활성화하는 도메인입니다. 따라서 Astra Trident는 추가되지 않습니다 <code>seLinuxOptions</code> 컨테이너로.
DAC	권한이 있는 컨테이너는 루트로 실행되어야 합니다. 권한이 없는 컨테이너는 <code>root</code> 로 실행되어 CSI에 필요한 UNIX 소켓에 액세스합니다.

POD 보안 표준(PSS)

라벨	설명	기본값
pod-security.kubernetes.io/enforce	Trident 컨트롤러와 노드를 설치 네임스페이스에 받아들일 수 있습니다. 네임스페이스 레이블을 변경하지 마십시오.	enforce: privileged
pod-security.kubernetes.io/enforce-version		enforce-version: <version of the current cluster or highest version of PSS tested.>



네임스페이스 레이블을 변경하면 포드가 예약되지 않고 "오류 생성:..." 또는 "경고: 트리덴트 - CSI -..."가 발생할 수 있습니다. 이 경우 에 대한 네임스페이스 레이블이 있는지 확인합니다 privileged 변경되었습니다. 있는 경우 Trident를 다시 설치합니다.

PSP(POD 보안 정책)

필드에 입력합니다	설명	기본값
allowPrivilegeEscalation	권한 있는 컨테이너는 권한 에스컬레이션을 허용해야 합니다.	true
allowedCSIDrivers	Trident는 인라인 CSI 임시 볼륨을 사용하지 않습니다.	비어 있습니다
allowedCapabilities	권한이 없는 Trident 컨테이너는 기본 세트보다 더 많은 기능을 필요로 하지 않으며 권한이 있는 컨테이너에 모든 가능한 기능이 부여됩니다.	비어 있습니다
allowedFlexVolumes	Trident는 을 사용하지 않습니다 "FlexVolume 드라이버"따라서 허용된 볼륨 목록에 포함되지 않습니다.	비어 있습니다
allowedHostPaths	Trident 노드 포드는 노드의 루트 파일 시스템을 마운트하므로 이 목록을 설정하는 데는 아무런 이점이 없습니다.	비어 있습니다
allowedProcMountTypes	Trident는 아무 용하지 않습니다 ProcMountTypes.	비어 있습니다
allowedUnsafeSysctls	Trident는 안전하지 않을 필요가 없습니다 sysctls.	비어 있습니다
defaultAddCapabilities	권한이 있는 컨테이너에 기능을 추가할 필요가 없습니다.	비어 있습니다
defaultAllowPrivilegeEscalation	권한 에스컬레이션을 허용하는 작업은 각 Trident 포드에서 처리됩니다.	false
forbiddenSysctls	아니요 sysctls 허용됩니다.	비어 있습니다
fsGroup	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny

필드에 입력합니다	설명	기본값
hostIPC	NFS 볼륨을 마운트하려면 호스트 IPC가 와 통신해야 합니다 nfsd	true
hostNetwork	iscsiadm을 사용하려면 호스트 네트워크가 iSCSI 데몬과 통신해야 합니다.	true
hostPID	호스트 PID가 있는지 확인해야 합니다 rpc-statd 노드에서 실행 중입니다.	true
hostPorts	Trident는 호스트 포트를 사용하지 않습니다.	비어 있습니다
privileged	Trident 노드 포드는 볼륨을 마운트하려면 권한이 있는 컨테이너를 실행해야 합니다.	true
readOnlyRootFilesystem	Trident 노드 포드는 노드 파일 시스템에 써야 합니다.	false
requiredDropCapabilities	Trident 노드 포드는 권한이 있는 컨테이너를 실행하고 기능을 삭제할 수 없습니다.	none
runAsGroup	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny
runAsUser	Trident 컨테이너가 루트로 실행됩니다.	runAsAny
runtimeClass	Trident가 사용되지 않습니다 RuntimeClasses.	비어 있습니다
seLinux	Trident가 설정되지 않았습니다 seLinuxOptions 현재 컨테이너 실행 시간과 Kubernetes 배포에서 SELinux를 처리하는 방법은 서로 다릅니다.	비어 있습니다
supplementalGroups	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny
volumes	Trident Pod에는 이러한 볼륨 플러그인이 필요합니다.	hostPath, projected, emptyDir

SCC(Security Context Constraints)

라벨	설명	기본값
allowHostDirVolumePlugin	Trident 노드 포드는 노드의 루트 파일 시스템을 마운트합니다.	true
allowHostIPC	NFS 볼륨을 마운트하려면 호스트 IPC가 와 통신해야 합니다 nfsd.	true

라벨	설명	기본값
allowHostNetwork	iscsiadm을 사용하려면 호스트 네트워크가 iSCSI 데몬과 통신해야 합니다.	true
allowHostPID	호스트 PID가 있는지 확인해야 합니다. rpc-statd 노드에서 실행 중입니다.	true
allowHostPorts	Trident는 호스트 포트를 사용하지 않습니다.	false
allowPrivilegeEscalation	권한 있는 컨테이너는 권한 에스컬레이션을 허용해야 합니다.	true
allowPrivilegedContainer	Trident 노드 포드는 볼륨을 마운트하려면 권한이 있는 컨테이너를 실행해야 합니다.	true
allowedUnsafeSysctls	Trident는 안전하지 않을 필요가 없습니다. sysctls.	none
allowedCapabilities	권한이 없는 Trident 컨테이너는 기본 세트보다 더 많은 기능을 필요로 하지 않으며 권한이 있는 컨테이너에 모든 가능한 기능이 부여됩니다.	비어 있습니다
defaultAddCapabilities	권한이 있는 컨테이너에 기능을 추가할 필요가 없습니다.	비어 있습니다
fsGroup	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny
groups	이 SCC는 Trident에만 해당되며 사용자에게 바인딩됩니다.	비어 있습니다
readOnlyRootFilesystem	Trident 노드 포드는 노드 파일 시스템에 써야 합니다.	false
requiredDropCapabilities	Trident 노드 포드는 권한이 있는 컨테이너를 실행하고 기능을 삭제할 수 없습니다.	none
runAsUser	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny
seLinuxContext	Trident가 설정되지 않았습니다. seLinuxOptions 현재 컨테이너 실행 시간과 Kubernetes 배포에서 SELinux를 처리하는 방법은 서로 다릅니다.	비어 있습니다
seccompProfiles	특권 컨테이너는 항상 "비제한" 상태로 실행됩니다.	비어 있습니다
supplementalGroups	Trident 컨테이너가 루트로 실행됩니다.	RunAsAny
users	이 SCC를 Trident 네임스페이스의 Trident 사용자에게 바인딩하기 위해 하나의 항목이 제공됩니다.	해당 없음

라벨	설명	기본값
volumes	Trident Pod에는 이러한 볼륨 플러그인이 필요합니다.	hostPath, downwardAPI, projected, emptyDir

이전 버전의 문서

Astra Trident 23.01을 실행하지 않는 경우 이전 릴리즈에 대한 문서를 사용할 수 있습니다.

- ["Astra Trident 22.10"](#)
- ["Astra Trident 22.07"](#)
- ["Astra Trident 22.04"](#)
- ["Astra Trident 22.01"](#)
- ["Astra Trident 21.10"](#)
- ["Astra Trident 21.07"](#)

법적 고지

법적 고지 사항은 저작권 선언, 상표, 특허 등에 대한 액세스를 제공합니다.

저작권

["https://www.netapp.com/company/legal/copyright/"](https://www.netapp.com/company/legal/copyright/)

상표

NetApp, NetApp 로고, NetApp 상표 페이지에 나열된 마크는 NetApp Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.

["https://www.netapp.com/company/legal/trademarks/"](https://www.netapp.com/company/legal/trademarks/)

특허

NetApp 소유 특허 목록은 다음 사이트에서 확인할 수 있습니다.

<https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf>

개인 정보 보호 정책

["https://www.netapp.com/company/legal/privacy-policy/"](https://www.netapp.com/company/legal/privacy-policy/)

오픈 소스

각 릴리스의 공지 파일에서 Astra Trident용 NetApp 소프트웨어에 사용된 타사 저작권 및 라이선스를 검토할 수 있습니다 <https://github.com/NetApp/trident/>.

저작권 정보

Copyright © 2024 NetApp, Inc. All Rights Reserved. 미국에서 인쇄된 본 문서의 어떠한 부분도 저작권 소유자의 사전 서면 승인 없이는 어떠한 형식이나 수단(복사, 녹음, 녹화 또는 전자 검색 시스템에 저장하는 것을 비롯한 그래픽, 전자적 또는 기계적 방법)으로도 복제될 수 없습니다.

NetApp이 저작권을 가진 자료에 있는 소프트웨어에는 아래의 라이선스와 고지사항이 적용됩니다.

본 소프트웨어는 NetApp에 의해 '있는 그대로' 제공되며 상품성 및 특정 목적에의 적합성에 대한 명시적 또는 묵시적 보증을 포함하여(이에 제한되지 않음) 어떠한 보증도 하지 않습니다. NetApp은 대체품 또는 대체 서비스의 조달, 사용 불능, 데이터 손실, 이익 손실, 영업 중단을 포함하여(이에 국한되지 않음), 이 소프트웨어의 사용으로 인해 발생하는 모든 직접 및 간접 손해, 우발적 손해, 특별 손해, 징벌적 손해, 결과적 손해의 발생에 대하여 그 발생 이유, 책임론, 계약 여부, 엄격한 책임, 불법 행위(과실 또는 그렇지 않은 경우)와 관계없이 어떠한 책임도 지지 않으며, 이와 같은 손실의 발생 가능성이 통지되었다 하더라도 마찬가지입니다.

NetApp은 본 문서에 설명된 제품을 언제든지 예고 없이 변경할 권리를 보유합니다. NetApp은 NetApp의 명시적인 서면 동의를 받은 경우를 제외하고 본 문서에 설명된 제품을 사용하여 발생하는 어떠한 문제에도 책임을 지지 않습니다. 본 제품의 사용 또는 구매의 경우 NetApp에서는 어떠한 특허권, 상표권 또는 기타 지적 재산권이 적용되는 라이선스도 제공하지 않습니다.

본 설명서에 설명된 제품은 하나 이상의 미국 특허, 해외 특허 또는 출원 중인 특허로 보호됩니다.

제한적 권리 표시: 정부에 의한 사용, 복제 또는 공개에는 DFARS 252.227-7013(2014년 2월) 및 FAR 52.227-19(2007년 12월)의 기술 데이터-비상업적 품목에 대한 권리(Rights in Technical Data -Noncommercial Items) 조항의 하위 조항 (b)(3)에 설명된 제한사항이 적용됩니다.

여기에 포함된 데이터는 상업용 제품 및/또는 상업용 서비스(FAR 2.101에 정의)에 해당하며 NetApp, Inc.의 독점 자산입니다. 본 계약에 따라 제공되는 모든 NetApp 기술 데이터 및 컴퓨터 소프트웨어는 본질적으로 상업용이며 개인 비용만으로 개발되었습니다. 미국 정부는 데이터가 제공된 미국 계약과 관련하여 해당 계약을 지원하는 데에만 데이터에 대한 전 세계적으로 비독점적이고 양도할 수 없으며 재사용이 불가능하며 취소 불가능한 라이선스를 제한적으로 가집니다. 여기에 제공된 경우를 제외하고 NetApp, Inc.의 사전 서면 승인 없이는 이 데이터를 사용, 공개, 재생산, 수정, 수행 또는 표시할 수 없습니다. 미국 국방부에 대한 정부 라이선스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

NETAPP, NETAPP 로고 및 <http://www.netapp.com/TM>에 나열된 마크는 NetApp, Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.