# Trident Protect로 애플리케이션을 보호하세요

Trident

NetApp
March 05, 2026

This PDF was generated from https://docs.netapp.com/ko-kr/trident-2506/trident-protect/learn-about-trident-protect.html on March 05, 2026. Always check docs.netapp.com for the latest.

# 목차

# Trident Protect로 애플리케이션을 보호하세요

## Trident Protect에 대해 알아보세요

NetApp Trident Protect는 NetApp ONTAP 스토리지 시스템과 NetApp Trident CSI 스토리지 프로비저너가 지원하는 상태 저장 Kubernetes 애플리케이션의 기능과 가용성을 향상시키는 고급 애플리케이션 데이터 관리 기능을 제공합니다. Trident Protect는 퍼블릭 클라우드와 온프레미스 환경에서 컨테이너화된 워크로드의 관리, 보호 및 이동을 간소화합니다. 또한 API와 CLI를 통해 자동화 기능도 제공합니다.

Trident Protect를 사용하면 사용자 정의 리소스(CR)를 생성하거나 Trident Protect CLI를 사용하여 애플리케이션을 보호할 수 있습니다.

### 다음은 무엇인가요?

Trident Protect를 설치하기 전에 요구 사항에 대해 알아볼 수 있습니다.

- "Trident 프로텍트 요구 사항"

## Trident Protect 설치

### Trident 프로텍트 요구 사항

운영 환경, 애플리케이션 클러스터, 애플리케이션 및 라이선스의 준비 상태를 확인하여 시작하세요. Trident Protect를 배포하고 운영하려면 환경이 이러한 요구 사항을 충족하는지 확인하세요.

#### Trident Protect Kubernetes 클러스터 호환성

Trident Protect는 다음을 포함한 광범위한 완전 관리형 및 자체 관리형 Kubernetes 제품과 호환됩니다.

- 아마존 엘라스틱 쿠버네티스 서비스(EKS)
- Google Kubernetes 엔진(GKE)
- Microsoft Azure 쿠버네티스 서비스(AKS)
- 레드햇 오픈시프트
- SUSE Rancher
- VMware Tanzu 포트폴리오
- 업스트림 쿠버네티스

- Trident Protect 백업은 Linux 컴퓨트 노드에서만 지원됩니다. Windows 컴퓨팅 노드에서는 백업 작업이 지원되지 않습니다.
- Trident Protect를 설치하는 클러스터가 실행 중인 스냅샷 컨트롤러와 관련 CRD로 구성되어 있는지 확인하세요. 스냅샷 컨트롤러를 설치하려면 다음을 참조하세요. "이 지침" .

### Trident Protect 스토리지 백엔드 호환성

Trident Protect는 다음과 같은 스토리지 백엔드를 지원합니다.

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP 스토리지 어레이
- Google Cloud NetApp Volumes
- Azure NetApp Files

스토리지 백엔드가 다음 요구 사항을 충족하는지 확인하세요.

- 클러스터에 연결된 NetApp 스토리지가 Trident 24.02 이상(Trident 24.10 권장)을 사용하는지 확인하세요.
- NetApp ONTAP 스토리지 백엔드가 있는지 확인하세요.
- 백업을 저장하기 위해 개체 스토리지 버킷을 구성했는지 확인하세요.
- 애플리케이션이나 애플리케이션 데이터 관리 작업에 사용할 애플리케이션 네임스페이스를 만듭니다. Trident Protect는 이러한 네임스페이스를 자동으로 생성하지 않습니다. 사용자 정의 리소스에 존재하지 않는 네임스페이스를 지정하면 작업이 실패합니다.

### nas-economy 볼륨에 대한 요구 사항

Trident Protect는 NAS 경제형 볼륨에 대한 백업 및 복원 작업을 지원합니다. 현재 NAS 경제 볼륨에 대한 스냅샷, 복제 및 SnapMirror 복제는 지원되지 않습니다. Trident Protect와 함께 사용하려는 각 NAS-Economy 볼륨에 대해 스냅샷 디렉토리를 활성화해야 합니다.

일부 애플리케이션은 스냅샷 디렉토리를 사용하는 볼륨과 호환되지 않습니다. 이러한 애플리케이션의 경우 ONTAP 스토리지 시스템에서 다음 명령을 실행하여 스냅샷 디렉토리를 숨겨야 합니다.

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

다음 명령을 각 nas-economy 볼륨에 대해 실행하여 스냅샷 디렉토리를 활성화할 수 있습니다. `<volume-UUID>` 변경하려는 볼륨의 UUID를 사용하여:

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

Trident 백엔드 구성 옵션을 설정하여 새 볼륨에 대해 기본적으로 스냅샷 디렉토리를 활성화할 수 있습니다. `snapshotDir` 에게 `true` . 기존 볼륨에는 영향을 미치지 않습니다.

### KubeVirt VM을 사용하여 데이터 보호

Trident Protect 24.10 및 24.10.1 이상 버전은 KubeVirt VM에서 실행되는 애플리케이션을 보호할 때 동작이 다릅니다. 두 버전 모두 데이터 보호 작업 중에 파일 시스템 동결 및 동결 해제를 활성화하거나 비활성화할 수 있습니다.

복원 작업 중에 `VirtualMachineSnapshots` 가상 머신(VM)에 대해 생성된 항목은 복원되지 않습니다.

**Trident** 프로텍트 **24.10**

Trident Protect 24.10은 데이터 보호 작업 중에 KubeVirt VM 파일 시스템의 일관된 상태를 자동으로 보장하지 않습니다. Trident Protect 24.10을 사용하여 KubeVirt VM 데이터를 보호하려면 데이터 보호 작업을 시작하기 전에 파일 시스템의 동결/동결 해제 기능을 수동으로 활성화해야 합니다. 이렇게 하면 파일 시스템이 일관된 상태를 유지하게 됩니다.

데이터 보호 작업 중 VM 파일 시스템의 동결 및 동결 해제를 관리하도록 Trident Protect 24.10을 구성할 수 있습니다. "가상화 구성" 그리고 다음 명령을 사용합니다.

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

**Trident Protect 24.10.1** 이상

Trident Protect 24.10.1부터 Trident Protect는 데이터 보호 작업 중에 KubeVirt 파일 시스템을 자동으로 동결 및 동결 해제합니다. 선택적으로 다음 명령을 사용하여 이 자동 동작을 비활성화할 수 있습니다.

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

**SnapMirror** 복제 요구 사항

NetApp SnapMirror 복제는 다음 ONTAP 솔루션에 대해 Trident Protect와 함께 사용할 수 있습니다.

- 온프레미스 NetApp FAS, AFF 및 ASA 클러스터
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

**SnapMirror** 복제를 위한 **ONTAP** 클러스터 요구 사항

SnapMirror 복제를 사용하려면 ONTAP 클러스터가 다음 요구 사항을 충족하는지 확인하세요.

- * NetApp Trident*: NetApp Trident 백엔드로 ONTAP 활용하는 소스 및 대상 Kubernetes 클러스터 모두에 존재해야 합니다. Trident Protect는 다음 드라이버로 지원되는 스토리지 클래스를 사용하여 NetApp SnapMirror 기술을 통한 복제를 지원합니다.
  - `ontap-nas`: NFS
  - `ontap-san`: iSCSI
  - `ontap-san`: FC
  - `ontap-san`: NVMe/TCP(최소 ONTAP 버전 9.15.1 필요)
- 라이선스: 데이터 보호 번들을 사용하는 ONTAP SnapMirror 비동기 라이선스는 소스 및 대상 ONTAP 클러스터 모두에서 활성화되어야 합니다. 참조하다 "ONTAP 의 SnapMirror 라이선싱 개요" 자세한 내용은.

ONTAP 9.10.1부터 모든 라이선스는 여러 기능을 활성화하는 단일 파일인 NetApp 라이선스 파일(NLF)로 제공됩니다. 참조하다"ONTAP One에 포함된 라이센스" 자세한 내용은.

SnapMirror 비동기 보호만 지원됩니다.

#### SnapMirror 복제를 위한 피어링 고려 사항

스토리지 백엔드 피어링을 사용하려면 환경이 다음 요구 사항을 충족하는지 확인하세요.

- 클러스터 및 **SVM**: ONTAP 스토리지 백엔드는 피어링되어야 합니다. 참조하다 "클러스터 및 SVM 피어링 개요" 자세한 내용은.

  두 ONTAP 클러스터 간 복제 관계에 사용된 SVM 이름이 고유한지 확인하세요.

- * NetApp Trident 및 SVM*: 피어링된 원격 SVM은 대상 클러스터의 NetApp Trident 에서 사용할 수 있어야 합니다.
- 관리형 백엔드: 복제 관계를 생성하려면 Trident Protect에서 ONTAP 스토리지 백엔드를 추가하고 관리해야 합니다.

#### SnapMirror 복제를 위한 Trident / ONTAP 구성

Trident Protect를 사용하려면 소스 및 대상 클러스터 모두에 대한 복제를 지원하는 하나 이상의 스토리지 백엔드를 구성해야 합니다. 소스 및 대상 클러스터가 동일한 경우, 최상의 복원력을 위해 대상 애플리케이션은 소스 애플리케이션과 다른 스토리지 백엔드를 사용해야 합니다.

#### SnapMirror 복제를 위한 Kubernetes 클러스터 요구 사항

Kubernetes 클러스터가 다음 요구 사항을 충족하는지 확인하세요.

- **AppVault** 접근성: 애플리케이션 개체 복제를 위해 소스 클러스터와 대상 클러스터 모두 AppVault에서 읽고 쓸 수 있는 네트워크 액세스 권한이 있어야 합니다.
- 네트워크 연결: WAN을 통해 클러스터와 AppVault 간의 통신을 활성화하기 위해 방화벽 규칙, 버킷 권한 및 IP 허용 목록을 구성합니다.

많은 기업 환경에서는 WAN 연결 전반에 걸쳐 엄격한 방화벽 정책을 구현합니다. 복제를 구성하기 전에 인프라 팀과 함께 이러한 네트워크 요구 사항을 확인하세요.

## Trident Protect 설치 및 구성

사용자 환경이 Trident Protect에 대한 요구 사항을 충족하는 경우 다음 단계에 따라 클러스터에 Trident Protect를 설치할 수 있습니다. NetApp 에서 Trident Protect를 구입하거나 개인 레지스트리에서 설치할 수 있습니다. 클러스터가 인터넷에 액세스할 수 없는 경우 개인 레지스트리에서 설치하는 것이 도움이 됩니다.

#### Trident Protect 설치

**NetApp** 에서 **Trident Protect** 설치

단계

1. Trident Helm 저장소를 추가합니다.

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

2. Helm을 사용하여 Trident Protect를 설치하세요. 바꾸다 `<name-of-cluster>` 클러스터에 할당되고 클러스터의 백업과 스냅샷을 식별하는 데 사용되는 클러스터 이름이 포함됩니다.

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2506.0 --create
-namespace --namespace trident-protect
```

개인 레지스트리에서 **Trident Protect** 설치

Kubernetes 클러스터가 인터넷에 액세스할 수 없는 경우 개인 이미지 레지스트리에서 Trident Protect를 설치할 수 있습니다. 다음 예에서 괄호 안의 값을 사용자 환경의 정보로 바꾸세요.

단계

1. 다음 이미지를 로컬 머신으로 가져와서 태그를 업데이트한 다음 개인 레지스트리에 푸시합니다.

```
netapp/controller:25.06.0
netapp/restic:25.06.0
netapp/kopia:25.06.0
netapp/trident-autosupport:25.06.0
netapp/exechook:25.06.0
netapp/resourcebackup:25.06.0
netapp/resourcerestore:25.06.0
netapp/resourcedelete:25.06.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

예를 들어:

```
docker pull netapp/controller:25.06.0
```

```
docker tag netapp/controller:25.06.0 <private-registry-
url>/controller:25.06.0
```

```
docker push <private-registry-url>/controller:25.06.0
```

2. Trident Protect 시스템 네임스페이스를 만듭니다.

```
kubectl create ns trident-protect
```

3. 레지스트리에 로그인하세요:

```
helm registry login <private-registry-url> -u <account-id> -p <api-token>
```

4. 개인 레지스트리 인증에 사용할 풀 시크릿을 만듭니다.

```
kubectl create secret docker-registry regcred --docker-username=<registry-username> --docker-password=<api-token> -n trident-protect --docker-server=<private-registry-url>
```

5. Trident Helm 저장소를 추가합니다.

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

6. 라는 이름의 파일을 만듭니다. `protectValues.yaml` . 다음 Trident Protect 설정이 포함되어 있는지 확인하세요.

```
---
image:
  registry: <private-registry-url>
imagePullSecrets:
  - name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
    - name: regcred
webhooksCleanup:
  imagePullSecrets:
    - name: regcred
```

7. Helm을 사용하여 Trident Protect를 설치하세요. 바꾸다 `<name_of_cluster>` 클러스터에 할당되고 클러스터의 백업과 스냅샷을 식별하는 데 사용되는 클러스터 이름이 포함됩니다.

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name_of_cluster> --version 100.2506.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

## Trident Protect CLI 플러그인을 설치하세요

Trident Protect 명령줄 플러그인을 사용할 수 있습니다. 이 플러그인은 Trident 의 확장 기능입니다. `tridentctl` Trident Protect 사용자 정의 리소스(CR)를 생성하고 상호 작용할 수 있는 유틸리티입니다.

### Trident Protect CLI 플러그인을 설치하세요

명령줄 유틸리티를 사용하려면 먼저 클러스터에 액세스하는 데 사용하는 컴퓨터에 유틸리티를 설치해야 합니다. 컴퓨터에서 x64 또는 ARM CPU를 사용하는지 여부에 따라 다음 단계를 따르세요.

**Linux AMD64 CPU**용 플러그인 다운로드

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-linux-amd64
```

**Linux ARM64 CPU**용 플러그인 다운로드

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-linux-arm64
```

**Mac AMD64 CPU**용 플러그인 다운로드

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-macos-amd64
```

**Mac ARM64 CPU**용 플러그인 다운로드

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-macos-arm64
```

1. 플러그인 바이너리에 대한 실행 권한을 활성화합니다.

```
chmod +x tridentctl-protect
```

2. 플러그인 바이너리를 PATH 변수에 정의된 위치에 복사합니다. 예를 들어, `/usr/bin` 또는 `/usr/local/bin` (높은 권한이 필요할 수 있음):

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 선택적으로 플러그인 바이너리를 홈 디렉토리의 원하는 위치에 복사할 수 있습니다. 이 경우 해당 위치가 PATH 변수의 일부인지 확인하는 것이 좋습니다.

```
cp ./tridentctl-protect ~/bin/
```

플러그인을 PATH 변수의 위치에 복사하면 다음을 입력하여 플러그인을 사용할 수 있습니다. `tridentctl-protect` 또는 `tridentctl protect` 어느 위치에서나.

**Trident CLI** 플러그인 도움말 보기

플러그인의 기능에 대한 자세한 도움말을 얻으려면 내장된 플러그인 도움말 기능을 사용할 수 있습니다.

단계

1. 도움말 기능을 사용하여 사용 지침을 확인하세요.

```
tridentctl-protect help
```

명령 자동 완성 활성화

Trident Protect CLI 플러그인을 설치한 후 특정 명령에 대한 자동 완성을 활성화할 수 있습니다.

**Bash** 셸에 대한 자동 완성을 활성화합니다.

단계

1. 완성 스크립트를 다운로드하세요:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/tridentctl-completion.bash
```

2. 스크립트를 포함할 새 디렉토리를 홈 디렉토리에 만듭니다.

```
mkdir -p ~/.bash/completions
```

3. 다운로드한 스크립트를 다음으로 이동합니다. `~/.bash/completions` 예배 규칙서:

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. 다음 줄을 추가하세요 `~/.bashrc` 홈 디렉토리에 있는 파일:

```
source ~/.bash/completions/tridentctl-completion.bash
```

**Z** 셸에 대한 자동 완성을 활성화합니다.

단계

1. 완성 스크립트를 다운로드하세요:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/tridentctl-completion.zsh
```

2. 스크립트를 포함할 새 디렉토리를 홈 디렉토리에 만듭니다.

```
mkdir -p ~/.zsh/completions
```

3. 다운로드한 스크립트를 다음으로 이동합니다. `~/.zsh/completions` 예배 규칙서:

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. 다음 줄을 추가하세요 `~/.zprofile` 홈 디렉토리에 있는 파일:

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

결과

다음 셸 로그인 시 tridentctl-protect 플러그인을 사용하여 명령 자동 완성 기능을 사용할 수 있습니다.

## Trident Protect 설치 사용자 정의

Trident Protect의 기본 구성을 사용자 환경의 특정 요구 사항에 맞게 사용자 정의할 수 있습니다.

### Trident Protect 컨테이너 리소스 제한 지정

Trident Protect를 설치한 후에는 구성 파일을 사용하여 Trident Protect 컨테이너에 대한 리소스 제한을 지정할 수 있습니다. 리소스 제한을 설정하면 Trident Protect 작업에서 클러스터 리소스가 얼마나 소모되는지 제어할 수 있습니다.

단계

1. 라는 이름의 파일을 만듭니다. `resourceLimits.yaml` .
2. 사용자 환경의 요구 사항에 따라 Trident Protect 컨테이너에 대한 리소스 제한 옵션으로 파일을 채웁니다.

   다음 예제 구성 파일은 사용 가능한 설정을 보여주고 각 리소스 제한에 대한 기본값을 포함합니다.

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
```

```
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. 값을 적용합니다 `resourceLimits.yaml` 파일:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

보안 컨텍스트 제약 조건 사용자 정의

Trident Protect를 설치한 후에는 구성 파일을 사용하여 Trident Protect 컨테이너에 대한 OpenShift 보안 컨텍스트 제약 조건(SCC)을 수정할 수 있습니다. 이러한 제약 조건은 Red Hat OpenShift 클러스터의 포드에 대한 보안 제한 사항을 정의합니다.

단계

1. 라는 이름의 파일을 만듭니다. `sccconfig.yaml` .
2. 파일에 SCC 옵션을 추가하고 환경의 요구 사항에 맞게 매개변수를 수정합니다.

   다음 예에서는 SCC 옵션에 대한 매개변수의 기본값을 보여줍니다.

```
scc:
  create: true
  name: trident-protect-job
  priority: 1
```

이 표에서는 SCC 옵션의 매개변수를 설명합니다.

| 매개변수 | 설명 | 기본 |
|---|---|---|
| 만들다 | SCC 리소스를 생성할 수 있는지 여부를 결정합니다. SCC 리소스는 다음 경우에만 생성됩니다. `scc.create` 로 설정됩니다 `true` Helm 설치 프로세스는 OpenShift 환경을 식별합니다. OpenShift에서 작동하지 않는 경우 또는 `scc.create` 로 설정됩니다 `false` , SCC 리소스가 생성되지 않습니다. | true |
| 이름 | SCC의 이름을 지정합니다. | 삼지창 보호 직업 |
| 우선 사항 | SCC의 우선순위를 정의합니다. 우선순위 값이 높은 SCC는 값이 낮은 SCC보다 먼저 평가됩니다. | 1 |

3. 값을 적용합니다 `sccconfig.yaml` 파일:

```
helm upgrade trident-protect netapp-trident-protect/trident-protect -f
sccconfig.yaml --reuse-values
```

이렇게 하면 기본값이 다음에 지정된 값으로 대체됩니다. `sccconfig.yaml` 파일.

추가 **Trident Protect** 헬름 차트 설정 구성

사용자의 특정 요구 사항에 맞게 AutoSupport 설정과 네임스페이스 필터링을 사용자 정의할 수 있습니다. 다음 표에서는 사용 가능한 구성 매개변수를 설명합니다.

| 매개변수 | 유형 | 설명 |
|---|---|---|
| autoSupport.proxy | 끈 | NetApp AutoSupport 연결을 위한 프록시 URL을 구성합니다. 이를 사용하면 프록시 서버를 통해 지원 번들 업로드를 라우팅할 수 있습니다. 예: `http://my.proxy.url` . |
| autoSupport.안전하지 않음 | 부울 | AutoSupport 프록시 연결에 대한 TLS 검증을 건너뜁니다. `true` . 안전하지 않은 프록시 연결에만 사용하세요. (기본: `false` ) |

| 매개변수 | 유형 | 설명 |
| --- | --- | --- |
| autoSupport.활성화됨 | 부울 | 일일 Trident Protect AutoSupport 번들 업로드를 활성화하거나 비활성화합니다. 설정 시 `false` , 예약된 일일 업로드는 비활성화되지만, 여전히 수동으로 지원 번들을 생성할 수 있습니다. (기본: `true` ) |
| restoreSkipNamespaceAnnotations | 끈 | 백업 및 복원 작업에서 제외할 네임스페이스 주석의 쉼표로 구분된 목록입니다. 주석을 기준으로 네임스페이스를 필터링할 수 있습니다. |
| restoreSkipNamespaceLabels | 끈 | 백업 및 복원 작업에서 제외할 네임스페이스 레이블의 쉼표로 구분된 목록입니다. 라벨을 기준으로 네임스페이스를 필터링할 수 있습니다. |

YAML 구성 파일이나 명령줄 플래그를 사용하여 이러한 옵션을 구성할 수 있습니다.

**YAML** 파일 사용

단계

1. 구성 파일을 만들고 이름을 지정하세요. `values.yaml` .
2. 생성한 파일에 사용자 정의하려는 구성 옵션을 추가합니다.

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
```

3. 다음을 채운 후 `values.yaml` 올바른 값을 가진 파일을 만들려면 구성 파일을 적용하세요:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

**CLI** 플래그 사용

단계

1. 다음 명령을 사용하십시오. `--set` 개별 매개변수를 지정하는 플래그:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set restoreSkipNamespaceAnnotations="annotation1,annotation2" \
  --set restoreSkipNamespaceLabels="label1,label2" \
  --reuse-values
```

**Trident Protect** 포드를 특정 노드로 제한

Kubernetes nodeSelector 노드 선택 제약 조건을 사용하면 노드 레이블을 기준으로 Trident Protect 포드를 실행할 수 있는 노드를 제어할 수 있습니다. 기본적으로 Trident Protect는 Linux를 실행하는 노드로 제한됩니다. 사용자의 요구 사항에 따라 이러한 제약 조건을 추가로 사용자 정의할 수 있습니다.

단계

1. 라는 이름의 파일을 만듭니다. `nodeSelectorConfig.yaml` .
2. 파일에 nodeSelector 옵션을 추가하고 파일을 수정하여 환경의 요구 사항에 맞게 노드 레이블을 추가하거나 변경하여 제한합니다. 예를 들어, 다음 파일에는 기본 OS 제한이 포함되어 있지만 특정 지역과 앱 이름도 대상으로 합니다.

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. 값을 적용합니다 `nodeSelectorConfig.yaml` 파일:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

이렇게 하면 기본 제한 사항이 사용자가 지정한 제한 사항으로 대체됩니다. `nodeSelectorConfig.yaml` 파일.

# Trident Protect 관리

## Trident Protect 권한 및 액세스 제어 관리

Trident Protect는 역할 기반 액세스 제어(RBAC)의 Kubernetes 모델을 사용합니다. 기본적으로 Trident Protect는 단일 시스템 네임스페이스와 연관된 기본 서비스 계정을 제공합니다. 사용자 수가 많거나 특정 보안 요구 사항이 있는 조직인 경우 Trident Protect의 RBAC 기능을 사용하여 리소스와 네임스페이스에 대한 액세스를 보다 세부적으로 제어할 수 있습니다.

클러스터 관리자는 항상 기본 리소스에 액세스할 수 있습니다. `trident-protect` 네임스페이스에 액세스할 수 있으며 다른 모든 네임스페이스의 리소스에도 액세스할 수 있습니다. 리소스와 애플리케이션에 대한 액세스를 제어하려면 추가 네임스페이스를 만들고 해당 네임스페이스에 리소스와 애플리케이션을 추가해야 합니다.

기본적으로 사용자는 애플리케이션 데이터 관리 CR을 생성할 수 없습니다. `trident-protect` 네임스페이스. 애플리케이션 네임스페이스에 애플리케이션 데이터 관리 CR을 만들어야 합니다(가장 좋은 방법은 연관된 애플리케이션과 동일한 네임스페이스에 애플리케이션 데이터 관리 CR을 만드는 것입니다).

> ⓘ 다음을 포함하는 권한이 있는 Trident Protect 사용자 정의 리소스 개체에 대한 액세스 권한은 관리자에게만 부여됩니다.
>
> - **AppVault**: 버킷 자격 증명 데이터가 필요합니다.
> - **AutoSupportBundle**: 메트릭, 로그 및 기타 중요한 Trident Protect 데이터를 수집합니다.
> - **AutoSupportBundleSchedule**: 로그 수집 일정을 관리합니다.
>
> 가장 좋은 방법은 RBAC를 사용하여 권한이 있는 개체에 대한 액세스를 관리자로 제한하는 것입니다.

RBAC가 리소스 및 네임스페이스에 대한 액세스를 규제하는 방법에 대한 자세한 내용은 다음을 참조하세요. "Kubernetes RBAC 문서" .

서비스 계정에 대한 자세한 내용은 다음을 참조하세요. "Kubernetes 서비스 계정 문서" .

예: 두 그룹의 사용자에 대한 액세스 관리

예를 들어, 어떤 조직에 클러스터 관리자, 엔지니어링 사용자 그룹, 마케팅 사용자 그룹이 있다고 가정해 보겠습니다. 클러스터 관리자는 엔지니어링 그룹과 마케팅 그룹이 각자의 네임스페이스에 할당된 리소스에만 액세스할 수 있는 환경을 만들기 위해 다음 작업을 완료합니다.

**1**단계**:** 각 그룹의 리소스를 포함할 네임스페이스 만들기

네임스페이스를 만들면 리소스를 논리적으로 분리하고 해당 리소스에 누가 액세스할 수 있는지 더 효과적으로 제어할 수 있습니다.

단계

1. 엔지니어링 그룹에 대한 네임스페이스를 만듭니다.

```
kubectl create ns engineering-ns
```

2. 마케팅 그룹을 위한 네임스페이스를 만듭니다.

```
kubectl create ns marketing-ns
```

**2**단계**:** 각 네임스페이스의 리소스와 상호 작용하기 위한 새 서비스 계정 만들기

새로 만든 네임스페이스마다 기본 서비스 계정이 제공되지만, 나중에 필요한 경우 그룹 간에 권한을 더욱 세분화할 수 있도록 각 사용자 그룹에 대한 서비스 계정을 만들어야 합니다.

단계

1. 엔지니어링 그룹에 대한 서비스 계정을 만듭니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. 마케팅 그룹을 위한 서비스 계정을 만듭니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

**3**단계**:** 각 새 서비스 계정에 대한 비밀 만들기

서비스 계정 비밀번호는 서비스 계정을 인증하는 데 사용되며, 손상된 경우 쉽게 삭제하고 다시 만들 수 있습니다.

단계

1. 엔지니어링 서비스 계정에 대한 비밀을 만듭니다.

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. 마케팅 서비스 계정에 대한 비밀을 만드세요.

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

**4**단계**: ClusterRole** 객체를 각각의 새 서비스 계정에 바인딩하기 위해 **RoleBinding** 객체를 만듭니다**.**

Trident Protect를 설치하면 기본 ClusterRole 개체가 생성됩니다. RoleBinding 객체를 생성하고 적용하여 이 ClusterRole을 서비스 계정에 바인딩할 수 있습니다.

단계

1. 엔지니어링 서비스 계정에 ClusterRole을 바인딩합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. ClusterRole을 마케팅 서비스 계정에 바인딩합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

**5**단계**:** 권한 테스트

권한이 올바른지 테스트합니다.

단계

1. 엔지니어링 사용자가 엔지니어링 리소스에 액세스할 수 있는지 확인하세요.

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. 엔지니어링 사용자가 마케팅 리소스에 액세스할 수 없는지 확인하세요.

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

#### 6단계: AppVault 개체에 대한 액세스 권한 부여

백업 및 스냅샷과 같은 데이터 관리 작업을 수행하려면 클러스터 관리자가 개별 사용자에게 AppVault 개체에 대한 액세스 권한을 부여해야 합니다.

단계

1. AppVault 및 비밀번호 조합 YAML 파일을 만들고 적용하여 사용자에게 AppVault에 대한 액세스 권한을 부여합니다. 예를 들어, 다음 CR은 사용자에게 AppVault에 대한 액세스 권한을 부여합니다. `eng-user` :

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 클러스터 관리자가 네임스페이스의 특정 리소스에 대한 액세스 권한을 부여할 수 있도록 역할 CR을 만들고 적용합니다. 예를 들어:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. RoleBinding CR을 생성하고 적용하여 사용자 eng-user에게 권한을 바인딩합니다. 예를 들어:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 권한이 올바른지 확인하세요.

   a. 모든 네임스페이스에 대한 AppVault 개체 정보를 검색해 보세요.

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   다음과 비슷한 출력이 표시됩니다.

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. 사용자가 이제 액세스 권한이 있는 AppVault 정보를 얻을 수 있는지 테스트합니다.

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

다음과 비슷한 출력이 표시됩니다.

```
yes
```

결과

AppVault 권한을 부여한 사용자는 애플리케이션 데이터 관리 작업에 대해 권한이 있는 AppVault 개체를 사용할 수 있어야 하며, 할당된 네임스페이스 외부의 리소스에 액세스하거나 액세스 권한이 없는 새 리소스를 만들 수 없습니다.

## Trident Protect 리소스 모니터링

### kube-state-metrics, Prometheus, Alertmanager 오픈 소스 도구를 사용하여 Trident Protect로 보호되는 리소스의 상태를 모니터링할 수 있습니다.

kube-state-metrics 서비스는 Kubernetes API 통신에서 메트릭을 생성합니다. Trident Protect와 함께 사용하면 사용자 환경의 리소스 상태에 대한 유용한 정보가 제공됩니다.

Prometheus는 kube-state-metrics에서 생성된 데이터를 수집하고 이러한 객체에 대한 쉽게 읽을 수 있는 정보로 제공할 수 있는 툴킷입니다. kube-state-metrics와 Prometheus를 함께 사용하면 Trident Protect로 관리하는 리소스의 상태와 상태를 모니터링할 수 있는 방법을 제공합니다.

Alertmanager는 Prometheus와 같은 도구에서 보낸 알림을 수집하여 사용자가 구성한 대상으로 라우팅하는 서비스입니다.

> 이 단계에 포함된 구성과 지침은 단지 예시일 뿐입니다. 사용자 환경에 맞게 사용자 정의해야 합니다. 구체적인 지침과 지원에 대해서는 다음 공식 문서를 참조하세요.
>
> - "kube-state-metrics 문서"
> - "프로메테우스 문서"
> - "Alertmanager 문서"

**1**단계**:** 모니터링 도구 설치

Trident Protect에서 리소스 모니터링을 활성화하려면 kube-state-metrics, Promethus, Alertmanager를 설치하고 구성해야 합니다.

#### **kube-state-metrics** 설치

Helm을 사용하여 kube-state-metrics를 설치할 수 있습니다.

단계

1. kube-state-metrics Helm 차트를 추가합니다. 예를 들어:

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. 클러스터에 Prometheus ServiceMonitor CRD를 적용합니다.

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. Helm 차트에 대한 구성 파일을 만듭니다(예: `metrics-config.yaml` ). 다음 예제 구성을 사용자 환경에 맞게 사용자 정의할 수 있습니다.

**metrics-config.yaml: kube-state-metrics Helm 차트 구성**

```
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: protect.trident.netapp.io
          kind: "Backup"
          version: "v1"
        labelsFromPath:
          backup_uid: [metadata, uid]
          backup_name: [metadata, name]
          creation_time: [metadata, creationTimestamp]
        metrics:
        - name: backup_info
          help: "Exposes details about the Backup state"
          each:
            type: Info
            info:
              labelsFromPath:
                appVaultReference: ["spec", "appVaultRef"]
                appReference: ["spec", "applicationRef"]
rbac:
  extraRules:
  - apiGroups: ["protect.trident.netapp.io"]
    resources: ["backups"]
    verbs: ["list", "watch"]

# Collect metrics from all namespaces
namespaces: ""

# Ensure that the metrics are collected by Prometheus
prometheus:
  monitor:
    enabled: true
```

4. Helm 차트를 배포하여 kube-state-metrics를 설치합니다. 예를 들어:

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. Trident Protect에서 사용하는 사용자 정의 리소스에 대한 메트릭을 생성하도록 kube-state-metrics를 구성하려면 다음 지침을 따르세요. "kube-state-metrics 사용자 정의 리소스 설명서" .

#### 프로메테우스 설치

다음 지침에 따라 Prometheus를 설치할 수 있습니다. "프로메테우스 문서" .

#### Alertmanager 설치

다음 지침에 따라 Alertmanager를 설치할 수 있습니다. "Alertmanager 문서" .

### 2단계: 모니터링 도구를 함께 작동하도록 구성

모니터링 도구를 설치한 후에는 도구가 함께 작동하도록 구성해야 합니다.

단계

1. kube-state-metrics를 Prometheus와 통합합니다. Prometheus 구성 파일 편집(`prometheus.yaml` ) 및 kube-state-metrics 서비스 정보를 추가합니다. 예를 들어:

   **prometheus.yaml: Prometheus**와 **kube-state-metrics** 서비스 통합

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: trident-protect
data:
  prometheus.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.trident-protect.svc:8080']
```

2. Prometheus를 구성하여 알림을 Alertmanager로 라우팅합니다. Prometheus 구성 파일 편집(`prometheus.yaml` ) 다음 섹션을 추가합니다.

**prometheus.yaml: Alertmanager**에 알림 보내기

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

결과

이제 Prometheus는 kube-state-metrics에서 메트릭을 수집하고 Alertmanager에 알림을 보낼 수 있습니다. 이제 알림을 트리거하는 조건과 알림을 보낼 위치를 구성할 준비가 되었습니다.

**3**단계**:** 알림 및 알림 대상 구성

도구가 함께 작동하도록 구성한 후에는 어떤 유형의 정보가 알림을 트리거하는지, 알림을 어디로 보낼지 구성해야 합니다.

경고 예**:** 백업 실패

다음 예제에서는 백업 사용자 정의 리소스의 상태가 다음과 같이 설정될 때 트리거되는 중요 경고를 정의합니다. `Error` 5초 이상. 이 예제를 사용자 환경에 맞게 사용자 정의하고 이 YAML 스니펫을 포함할 수 있습니다. `prometheus.yaml` 구성 파일:

**rules.yaml:** 실패한 백업에 대한 **Prometheus** 알림을 정의합니다**.**

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

다른 채널에 알림을 보내도록 **Alertmanager**를 구성합니다**.**

이메일, PagerDuty, Microsoft Teams 또는 기타 알림 서비스와 같은 다른 채널에 알림을 보내도록 Alertmanager를 구성하려면 해당 구성을 지정해야 합니다. `alertmanager.yaml` 파일.

다음 예제에서는 Alertmanager가 Slack 채널에 알림을 보내도록 구성합니다. 이 예제를 사용자 환경에 맞게 사용자 지정하려면 다음 값을 바꾸십시오. `api_url` 사용자 환경에서 사용되는 Slack 웹훅 URL이 포함된 키:

**alertmanager.yaml: Slack** 채널에 알림 보내기

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

## Trident Protect 지원 번들 생성

Trident Protect를 사용하면 관리자는 관리 중인 클러스터와 앱에 대한 로그, 메트릭, 토폴로지 정보 등 NetApp 지원에 유용한 정보가 포함된 번들을 생성할 수 있습니다. 인터넷에 연결되어 있는 경우 사용자 정의 리소스(CR) 파일을 사용하여 NetApp 지원 사이트(NSS)에 지원 번들을 업로드할 수 있습니다.

**CR**을 사용하여 지원 번들을 만듭니다.

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-support-bundle.yaml` ).
2. 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.triggerType**: (필수) 지원 번들이 즉시 생성되는지, 아니면 예약되는지를 결정합니다. 예약된 번들 생성은 UTC 기준 오전 12시에 이루어집니다. 가능한 값:
     - 예정됨
     - 수동
   - **spec.uploadEnabled**: (선택 사항) 지원 번들이 생성된 후 NetApp 지원 사이트에 업로드할지 여부를 제어합니다. 지정하지 않으면 기본값으로 설정됩니다. `false` . 가능한 값:
     - true
     - false (기본값)
   - **spec.dataWindowStart**: (선택 사항) 지원 번들에 포함된 데이터 창이 시작되어야 하는 날짜와 시간을 지정하는 RFC 3339 형식의 날짜 문자열입니다. 지정하지 않으면 기본적으로 24시간 전으로 설정됩니다. 지정할 수 있는 가장 빠른 날짜는 7일 전입니다.

     YAML 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. 다음을 채운 후 `trident-protect-support-bundle.yaml` 올바른 값으로 파일을 만들고 CR을 적용합니다.

```
kubectl apply -f trident-protect-support-bundle.yaml -n trident-
protect
```

**CLI**를 사용하여 지원 번들을 만듭니다.

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 지원 번들을 만듭니다. 그만큼 `trigger-type` 번들이 즉시

생성되는지 또는 생성 시간이 일정에 따라 결정되는지 여부를 결정하고 다음을 수행할 수 있습니다. `Manual` 또는 `Scheduled` . 기본 설정은 다음과 같습니다. `Manual` .

예를 들어:

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

지원 번들을 모니터링하고 검색합니다.

두 가지 방법 중 하나를 사용하여 지원 번들을 만든 후에는 생성 진행 상황을 모니터링하고 로컬 시스템으로 검색할 수 있습니다.

단계

1. 기다리다 `status.generationState` 도달하다 `Completed` 상태. 다음 명령을 사용하여 생성 진행 상황을 모니터링할 수 있습니다.

```
kubectl get autosupportbundle trident-protect-support-bundle -n trident-
protect
```

2. 지원 번들을 로컬 시스템으로 가져옵니다. 완성된 AutoSupport 번들에서 복사 명령을 가져옵니다.

```
kubectl describe autosupportbundle trident-protect-support-bundle -n
trident-protect
```

찾아라 `kubectl cp` 출력에서 명령을 선택하고 대상 인수를 원하는 로컬 디렉토리로 바꿔서 실행합니다.

## Trident 프로텍트 업그레이드

### Trident Protect를 최신 버전으로 업그레이드하면 새로운 기능이나 버그 수정을 활용할 수 있습니다.

> 버전 24.10에서 업그레이드하는 경우 업그레이드 중에 실행 중인 스냅샷이 실패할 수 있습니다. 이 오류로 인해 향후 스냅샷(수동 또는 예약)이 생성되는 것은 방해받지 않습니다. 업그레이드 중에 스냅샷이 실패하면 수동으로 새 스냅샷을 만들어서 애플리케이션을 보호할 수 있습니다.
>
> 잠재적인 실패를 방지하려면 업그레이드 전에 모든 스냅샷 일정을 비활성화한 다음 나중에 다시 활성화할 수 있습니다. 하지만 이로 인해 업그레이드 기간 동안 예약된 스냅샷이 누락되는 문제가 발생합니다.

Trident Protect를 업그레이드하려면 다음 단계를 수행하세요.

단계

1. Trident Helm 저장소를 업데이트하세요:

```
helm repo update
```

2. Trident Protect CRD 업그레이드:

> 25.06 이전 버전에서 업그레이드하는 경우 CRD가 이제 Trident Protect Helm 차트에 포함되었으므로 이 단계가 필요합니다.

a. CRD 관리를 전환하려면 이 명령을 실행하세요. `trident-protect-crds` 에게 `trident-protect` :

```
kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
{"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
```

b. Helm 비밀을 삭제하려면 이 명령을 실행하세요. `trident-protect-crds` 차트:

> 제거하지 마십시오 `trident-protect-crds` Helm을 사용하여 차트를 만들면 CRD와 관련 데이터가 제거될 수 있습니다.

```
kubectl delete secret -n trident-protect -l name=trident-protect-
crds,owner=helm
```

3. Trident 프로텍트 업그레이드:

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2506.0 --namespace trident-protect
```

# 애플리케이션 관리 및 보호

## Trident Protect AppVault 객체를 사용하여 버킷을 관리합니다.

Trident Protect의 버킷 사용자 정의 리소스(CR)는 AppVault로 알려져 있습니다. AppVault 객체는 스토리지 버킷의 선언적 Kubernetes 워크플로 표현입니다. AppVault CR에는 백업, 스냅샷, 복원 작업, SnapMirror 복제와 같은 보호 작업에 버킷을 사용하는 데 필요한 구성이 포함되어 있습니다. 관리자만 AppVault를 만들 수 있습니다.

애플리케이션에서 데이터 보호 작업을 수행할 때는 AppVault CR을 수동으로 또는 명령줄에서 생성해야 합니다. AppVault CR은 사용자 환경에 따라 다르므로, 이 페이지의 예시를 참고하여 AppVault CR을 생성할 수 있습니다.

Trident Protect가 설치된 클러스터에 AppVault CR이 있는지 확인하세요. AppVault CR이 없거나 액세스할 수 없는 경우 명령줄에 오류가 표시됩니다.

**AppVault** 인증 및 비밀번호 구성

AppVault CR을 생성하기 전에 AppVault와 선택한 데이터 이동자가 공급자 및 관련 리소스에 대해 인증할 수 있는지 확인하세요.

데이터 무버 저장소 비밀번호

CR이나 Trident Protect CLI 플러그인을 사용하여 AppVault 객체를 생성하는 경우 Restic 및 Kopia 암호화를 위한 사용자 정의 비밀번호가 포함된 Kubernetes 비밀번호를 지정할 수 있습니다. 비밀번호를 지정하지 않으면 Trident Protect는 기본 비밀번호를 사용합니다.

- AppVault CR을 수동으로 생성할 때 **spec.dataMoverPasswordSecretRef** 필드를 사용하여 비밀번호를 지정합니다.
- Trident Protect CLI를 사용하여 AppVault 객체를 생성할 때 다음을 사용하십시오. `--data-mover-password-secret-ref` 비밀을 지정하는 인수입니다.

데이터 무버 저장소 비밀번호 비밀을 만듭니다**.**

다음 예를 사용하여 비밀번호를 생성하세요. AppVault 객체를 생성할 때 Trident Protect가 이 비밀번호를 사용하여 데이터 이동 저장소를 인증하도록 지시할 수 있습니다.

- 사용하는 데이터 이동 서비스에 따라 해당 데이터 이동 서비스에 해당하는 비밀번호만 포함하면 됩니다. 예를 들어, Restic을 사용하고 있고 앞으로 Kopia를 사용할 계획이 없다면 비밀번호를 생성할 때 Restic 비밀번호만 포함할 수 있습니다.
- 비밀번호를 안전한 곳에 보관하세요. 같은 클러스터나 다른 클러스터에서 데이터를 복원할 때 필요합니다. 클러스터 또는 `trident-protect` 네임스페이스가 삭제되면 비밀번호 없이는 백업이나 스냅샷을 복원할 수 없습니다.

**CR**을 사용하세요

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

**CLI**를 사용하세요

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

#### S3 호환 스토리지 IAM 권한

Amazon S3, Generic S3 등 S3 호환 스토리지에 접속할 경우 "스토리지그리드 S3" , 또는 "ONTAP S3" Trident Protect를 사용하는 경우, 제공하는 사용자 자격 증명에 버킷에 액세스하는 데 필요한 권한이 있는지 확인해야 합니다. 다음은 Trident Protect에 대한 액세스에 필요한 최소한의 권한을 부여하는 정책의 예입니다. 이 정책은 S3 호환 버킷 정책을 관리하는 사용자에게 적용할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon S3 정책에 대한 자세한 내용은 다음 예제를 참조하세요. "Amazon S3 문서" .

#### Amazon S3(AWS) 인증을 위한 EKS Pod ID

Trident Protect는 Kopia 데이터 이동 작업을 위해 EKS Pod Identity를 지원합니다. 이 기능을 사용하면 Kubernetes 비밀에 AWS 자격 증명을 저장하지 않고도 S3 버킷에 안전하게 액세스할 수 있습니다.

- Trident Protect를 사용한 EKS Pod Identity 요구 사항*

Trident Protect와 함께 EKS Pod Identity를 사용하기 전에 다음 사항을 확인하세요.

- EKS 클러스터에 Pod Identity가 활성화되어 있습니다.
- 필요한 S3 버킷 권한이 있는 IAM 역할을 생성했습니다. 자세한 내용은 다음을 참조하세요. "S3 호환 스토리지 IAM 권한".
- IAM 역할은 다음 Trident Protect 서비스 계정과 연결됩니다.
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

Pod Identity를 활성화하고 IAM 역할을 서비스 계정과 연결하는 방법에 대한 자세한 지침은 다음을 참조하세요. "AWS EKS Pod Identity 문서" .

**AppVault** 구성 EKS Pod Identity를 사용하는 경우 AppVault CR을 다음과 같이 구성합니다. `useIAM: true` 명시적 자격 증명 대신 플래그:

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: eks-protect-vault
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-aws
      endpoint: s3.example.com
      useIAM: true
```

#### 클라우드 공급자를 위한 AppVault 키 생성 예시

AppVault CR을 정의할 때 IAM 인증을 사용하지 않는 한 공급자가 호스팅하는 리소스에 액세스하기 위한 자격 증명을 포함해야 합니다. 자격 증명에 대한 키를 생성하는 방법은 공급자에 따라 다릅니다. 다음은 여러 공급자에 대한 명령줄 키 생성 예입니다. 다음 예를 사용하여 각 클라우드 공급자의 자격 증명에 대한 키를 생성할 수 있습니다.

구글 클라우드

```
kubectl create secret generic <secret-name> \
--from-file=credentials=<mycreds-file.json> \
-n trident-protect
```

아마존 **S3(AWS)**

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<amazon-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

마이크로소프트 애저

```
kubectl create secret generic <secret-name> \
--from-literal=accountKey=<secret-name> \
-n trident-protect
```

일반 **S3**

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<generic-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

**ONTAP S3**

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<ontap-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

스토리지그리드 **S3**

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<storagegrid-s3-trident-protect-src
-bucket-secret> \
-n trident-protect
```

### AppVault 생성 예시

각 공급자에 대한 AppVault 정의의 예는 다음과 같습니다.

#### AppVault CR 예시

다음 CR 예제를 사용하여 각 클라우드 공급자에 대한 AppVault 객체를 만들 수 있습니다.

- Restic 및 Kopia 저장소 암호화를 위한 사용자 정의 비밀번호가 포함된 Kubernetes 비밀번호를 선택적으로 지정할 수 있습니다. 참조하다데이터 무버 저장소 비밀번호 자세한 내용은.
- Amazon S3(AWS) AppVault 개체의 경우 선택적으로 sessionToken을 지정할 수 있습니다. 이는 인증을 위해 SSO(Single Sign-On)를 사용하는 경우에 유용합니다. 이 토큰은 공급자에 대한 키를 생성할 때 생성됩니다.클라우드 공급자를 위한 AppVault 키 생성 예시 .
- S3 AppVault 개체의 경우 선택적으로 다음을 사용하여 아웃바운드 S3 트래픽에 대한 송신 프록시 URL을 지정할 수 있습니다. `spec.providerConfig.S3.proxyURL` 열쇠.

구글 클라우드

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

아마존 **S3(AWS)**

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3-secret
```

Kopia 데이터 무버와 함께 Pod Identity를 사용하는 EKS 환경의 경우 다음을 제거할 수 있습니다. `providerCredentials` 섹션을 추가하고 추가하세요 `useIAM: true` 아래에 `s3` 대신 구성을 사용하세요.

마이크로소프트 애저

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

일반 **S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

**ONTAP S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

스토리지그리드 **S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

**Trident Protect CLI**를 사용한 **AppVault** 생성 예

다음 CLI 명령 예를 사용하여 각 공급자에 대한 AppVault CR을 만들 수 있습니다.

- Restic 및 Kopia 저장소 암호화를 위한 사용자 정의 비밀번호가 포함된 Kubernetes 비밀번호를 선택적으로 지정할 수 있습니다. 참조하다데이터 무버 저장소 비밀번호 자세한 내용은.
- S3 AppVault 개체의 경우 선택적으로 다음을 사용하여 아웃바운드 S3 트래픽에 대한 송신 프록시 URL을 지정할 수 있습니다. `--proxy-url <ip_address:port>` 논쟁.

구글 클라우드

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

아마존 **S3(AWS)**

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

마이크로소프트 애저

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

일반 **S3**

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

**ONTAP S3**

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

스토리지그리드 **S3**

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

### **AppVault** 정보 보기

Trident Protect CLI 플러그인을 사용하면 클러스터에서 생성한 AppVault 개체에 대한 정보를 볼 수 있습니다.

단계

1. AppVault 개체의 내용을 봅니다.

```
tridentctl-protect get appvaultcontent gcp-vault \
--show-resources all \
-n trident-protect
```

예시 출력:

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+-----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+-----------------------------
+---------------------------+
```

2. 선택적으로 각 리소스에 대한 AppVaultPath를 보려면 플래그를 사용하세요. `--show-paths` .

   표의 첫 번째 열에 있는 클러스터 이름은 Trident Protect helm 설치 시 클러스터 이름이 지정된 경우에만 사용할 수 있습니다. 예를 들어: `--set clusterName=production1` .

### AppVault 제거

언제든지 AppVault 객체를 제거할 수 있습니다.

> 제거하지 마십시오 `finalizers` AppVault 개체를 삭제하기 전에 AppVault CR에 키를 입력하세요. 그렇게 하면 AppVault 버킷에 잔여 데이터가 생기고 클러스터에 버려진 리소스가 생길 수 있습니다.

시작하기 전에

삭제하려는 AppVault에서 사용 중인 모든 스냅샷 및 백업 CR을 삭제했는지 확인하세요.

**Kubernetes CLI**를 사용하여 **AppVault** 제거

1. AppVault 객체를 제거하고 교체합니다. `appvault-name` 제거할 AppVault 개체의 이름:

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

**Trident Protect CLI**를 사용하여 **AppVault** 제거

1. AppVault 객체를 제거하고 교체합니다. `appvault-name` 제거할 AppVault 개체의 이름:

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

## Trident Protect를 사용하여 관리를 위한 애플리케이션 정의

Trident Protect로 관리하려는 애플리케이션을 정의하려면 애플리케이션 CR과 관련 AppVault CR을 생성하면 됩니다.

### AppVault CR 만들기

애플리케이션에서 데이터 보호 작업을 수행할 때 사용할 AppVault CR을 만들어야 하며, AppVault CR은 Trident Protect가 설치된 클러스터에 있어야 합니다. AppVault CR은 사용자 환경에 따라 다릅니다. AppVault CR의 예는 다음을 참조하세요."AppVault 사용자 정의 리소스."

응용 프로그램 정의

Trident Protect로 관리하려는 각 애플리케이션을 정의해야 합니다. 수동으로 애플리케이션 CR을 생성하거나 Trident Protect CLI를 사용하여 관리할 애플리케이션을 정의할 수 있습니다.

**CR**을 사용하여 애플리케이션 추가

단계

1. 대상 애플리케이션 CR 파일을 만듭니다.
   a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `maria-app.yaml` ).
   b. 다음 속성을 구성합니다.
      - **metadata.name**: (필수) 애플리케이션 사용자 정의 리소스의 이름입니다. 보호 작업에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 이름을 기록해 두세요.
      - **spec.includedNamespaces**: (필수) 네임스페이스와 레이블 선택기를 사용하여 애플리케이션이 사용하는 네임스페이스와 리소스를 지정합니다. 애플리케이션 네임스페이스는 이 목록에 포함되어야 합니다. 레이블 선택기는 선택 사항이며 지정된 각 네임스페이스 내의 리소스를 필터링하는 데 사용할 수 있습니다.
      - **spec.includedClusterScopedResources**: (선택 사항) 이 특성을 사용하여 애플리케이션 정의에 포함될 클러스터 범위 리소스를 지정합니다. 이 속성을 사용하면 그룹, 버전, 종류 및 레이블을 기준으로 리소스를 선택할 수 있습니다.
        - **groupVersionKind**: (필수) 클러스터 범위 리소스의 API 그룹, 버전 및 종류를 지정합니다.
        - **labelSelector**: (선택 사항) 레이블을 기준으로 클러스터 범위 리소스를 필터링합니다.
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (선택 사항) 이 주석은 KubeVirt 환경과 같이 스냅샷 전에 파일 시스템이 정지되는 가상 머신에서 정의된 애플리케이션에만 적용됩니다. 스냅샷 중에 이 애플리케이션이 파일 시스템에 쓸 수 있는지 여부를 지정합니다. true로 설정하면 애플리케이션은 글로벌 설정을 무시하고 스냅샷 중에 파일 시스템에 쓸 수 있습니다. false로 설정하면 애플리케이션은 글로벌 설정을 무시하고 스냅샷을 찍는 동안 파일 시스템이 동결됩니다. 지정되었지만 애플리케이션 정의에 가상 머신이 없는 경우 주석은 무시됩니다. 지정하지 않으면 응용 프로그램은 다음을 따릅니다."글로벌 Trident 프로텍트 동결 설정" .

애플리케이션이 이미 생성된 후에 이 주석을 적용해야 하는 경우 다음 명령을 사용할 수 있습니다.

```
kubectl annotate application -n <application CR namespace> <application CR name> protect.trident.netapp.io/skip-vm-freeze="true"
```

+
YAML 예시:

+

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. (선택 사항) 특정 레이블이 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

    - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `Include` 또는 `Exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.

        - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.
            - **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
            - **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.
            - **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.
            - **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.

- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

> ⓘ 둘 다 `resourceFilter` 그리고 `labelSelector` 사용됩니다, `resourceFilter` 먼저 실행한 다음 `labelSelector` 결과 리소스에 적용됩니다.

예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

2. 환경에 맞게 애플리케이션 CR을 만든 후 CR을 적용합니다. 예를 들어:

```
kubectl apply -f maria-app.yaml
```

단계

1. 다음 예제 중 하나를 사용하여 애플리케이션 정의를 만들고 적용하고, 괄호 안의 값을 사용자 환경의 정보로 바꿉니다. 예제에 표시된 인수와 함께 쉼표로 구분된 목록을 사용하여 애플리케이션 정의에 네임스페이스와 리소스를 포함할 수 있습니다.

   스냅샷 중에 애플리케이션이 파일 시스템에 쓸 수 있는지 여부를 지정하기 위해 앱을 생성할 때 주석을 사용할 수 있습니다. 이는 스냅샷 전에 파일 시스템이 정지되는 KubeVirt 환경과 같은 가상 머신에서 정의된 애플리케이션에만 적용됩니다. 주석을 다음과 같이 설정하면 `true` , 애플리케이션은 글로벌 설정을 무시하고 스냅샷 동안 파일 시스템에 쓸 수 있습니다. 이것을 설정하면 `false` , 애플리케이션은 글로벌 설정을 무시하고 스냅샷을 찍는 동안 파일 시스템이 동결됩니다. 주석을 사용하지만 애플리케이션 정의에 가상 머신이 없는 경우 주석은 무시됩니다. 주석을 사용하지 않으면 응용 프로그램은 다음을 따릅니다."글로벌 Trident 프로텍트 동결 설정" .

CLI를 사용하여 애플리케이션을 생성할 때 주석을 지정하려면 다음을 사용할 수 있습니다. `--annotation` 깃발.

- 애플리케이션을 만들고 파일 시스템 동결 동작에 대한 글로벌 설정을 사용합니다.

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace>
```

- 애플리케이션을 생성하고 파일 시스템 정지 동작에 대한 로컬 애플리케이션 설정을 구성합니다.

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --annotation protect.trident.netapp.io/skip-vm-freeze
=<"true"|"false">
```

사용할 수 있습니다 `--resource-filter-include` 그리고 `--resource-filter-exclude` 리소스를 포함하거나 제외하기 위한 플래그 `resourceSelectionCriteria` 다음 예에서 볼 수 있듯이 그룹, 종류, 버전, 레이블, 이름 및 네임스페이스와 같은 것들이 있습니다.

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-namespace>
--resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my-
deployment"],"Namespaces":["my-
namespace"],"LabelSelectors":["app=my-app"]}]'
```

## Trident Protect를 사용하여 애플리케이션 보호

자동화된 보호 정책이나 임시 기반으로 스냅샷과 백업을 수행하여 Trident Protect가 관리하는 모든 앱을 보호할 수 있습니다.

데이터 보호 작업 중에 파일 시스템을 동결 및 동결 해제하도록 Trident Protect를 구성할 수 있습니다. "Trident Protect를 사용하여 파일 시스템 동결을 구성하는 방법에 대해 자세히 알아보세요.".

주문형 스냅샷 만들기

언제든지 주문형 스냅샷을 만들 수 있습니다.

클러스터 범위 리소스는 애플리케이션 정의에서 명시적으로 참조되거나 애플리케이션 네임스페이스에 대한 참조가 있는 경우 백업, 스냅샷 또는 복제본에 포함됩니다.

**CR**을 사용하여 스냅샷을 만듭니다.

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-snapshot-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.applicationRef**: 스냅샷을 생성할 애플리케이션의 Kubernetes 이름입니다.
   - **spec.appVaultRef**: (필수) 스냅샷 콘텐츠(메타데이터)를 저장해야 하는 AppVault의 이름입니다.
   - **spec.reclaimPolicy**: (선택 사항) 스냅샷 CR이 삭제될 때 스냅샷의 AppArchive에 어떤 일이 일어나는지 정의합니다. 이는 설정된 경우에도 다음을 의미합니다. `Retain` , 스냅샷이 삭제됩니다. 유효한 옵션:
     - `Retain`(기본)
     - `Delete`

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  reclaimPolicy: Delete
```

3. 다음을 채운 후 `trident-protect-snapshot-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-snapshot-cr.yaml
```

**CLI**를 사용하여 스냅샷을 만듭니다.

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔 스냅샷을 만듭니다. 예를 들어:

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

## 주문형 백업 만들기

언제든지 앱을 백업할 수 있습니다.

> 클러스터 범위 리소스는 애플리케이션 정의에서 명시적으로 참조되거나 애플리케이션 네임스페이스에 대한 참조가 있는 경우 백업, 스냅샷 또는 복제본에 포함됩니다.

### 시작하기 전에

장기 실행 S3 백업 작업에 대해 AWS 세션 토큰 만료 기간이 충분한지 확인하세요. 백업 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.

- 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
- 를 참조하세요 "AWS IAM 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

**CR**을 사용하여 백업을 만듭니다**.**

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-backup-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.applicationRef**: (필수) 백업할 애플리케이션의 Kubernetes 이름입니다.
   - **spec.appVaultRef**: (필수) 백업 내용을 저장해야 하는 AppVault의 이름입니다.
   - **spec.dataMover**: (선택 사항) 백업 작업에 사용할 백업 도구를 나타내는 문자열입니다. 가능한 값(대소문자 구분):
     - `Restic`
     - `Kopia`(기본)
   - **spec.reclaimPolicy**: (선택 사항) 백업이 클레임에서 해제될 때 발생하는 작업을 정의합니다. 가능한 값:
     - `Delete`
     - `Retain`(기본)
   - **spec.snapshotRef**: (선택 사항): 백업 소스로 사용할 스냅샷의 이름입니다. 제공되지 않으면 임시 스냅샷이 생성되어 백업됩니다.
   - **metadata.annotations.protect.trident.netapp.io/full-backup** : (선택 사항) 이 주석은 백업이 증분 방식이 아니어야 하는지 여부를 지정하는 데 사용됩니다. 기본적으로 모든 백업은 증분 백업입니다. 그러나 이 주석이 설정된 경우 `true` , 백업이 비증분 백업으로 전환됩니다. 지정하지 않으면 백업은 기본 증분 백업 설정을 따릅니다. 복원과 관련된 위험을 최소화하려면 주기적으로 전체 백업을 수행한 다음 전체 백업 사이에 증분 백업을 수행하는 것이 가장 좋습니다.

     YAML 예시:

```yaml
---
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
  annotations:
    protect.trident.netapp.io/full-backup: "true"
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. 다음을 채운 후 `trident-protect-backup-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-backup-cr.yaml
```

**CLI**를 사용하여 백업을 만듭니다.

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔 백업을 만듭니다. 예를 들어:

```
tridentctl-protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up> --data-mover
<Kopia_or_Restic> -n <application_namespace>
```

선택적으로 사용할 수 있습니다 `--full-backup` 백업이 증분되지 않아야 하는지 여부를 지정하는 플래그입니다. 기본적으로 모든 백업은 증분 백업입니다. 이 플래그를 사용하면 백업이 비증분 방식으로 진행됩니다. 복원과 관련된 위험을 최소화하려면 주기적으로 전체 백업을 수행하고 전체 백업 사이에 증분 백업을 수행하는 것이 가장 좋습니다.

데이터 보호 일정을 만듭니다

보호 정책은 정의된 일정에 따라 스냅샷, 백업 또는 둘 다를 생성하여 앱을 보호합니다. 매시간, 매일, 매주, 매월 스냅샷과 백업을 만들도록 선택할 수 있으며, 보관할 복사본 수를 지정할 수 있습니다. full-backup-rule 주석을 사용하여 비증분 전체 백업을 예약할 수 있습니다. 기본적으로 모든 백업은 증분 백업입니다. 주기적으로 전체 백업을 수행하고 그 사이에 증분 백업을 수행하면 복원과 관련된 위험을 줄이는 데 도움이 됩니다.

- 스냅샷에 대한 일정은 다음을 설정하여 생성할 수 있습니다. `backupRetention` 0으로 그리고 `snapshotRetention` 0보다 큰 값으로. 환경 `snapshotRetention` 0으로 설정하면 예약된 백업은 여전히 스냅샷을 생성하지만, 이는 일시적이며 백업이 완료되면 즉시 삭제됩니다.
- 클러스터 범위 리소스는 애플리케이션 정의에서 명시적으로 참조되거나 애플리케이션 네임스페이스에 대한 참조가 있는 경우 백업, 스냅샷 또는 복제본에 포함됩니다.

**CR**을 사용하여 일정을 만듭니다**.**

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-schedule-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.

   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.dataMover**: (선택 사항) 백업 작업에 사용할 백업 도구를 나타내는 문자열입니다. 가능한 값(대소문자 구분):
     - `Restic`
     - `Kopia`(기본)
   - **spec.applicationRef**: 백업할 애플리케이션의 Kubernetes 이름입니다.
   - **spec.appVaultRef**: (필수) 백업 내용을 저장해야 하는 AppVault의 이름입니다.
   - **spec.backupRetention**: 보관할 백업 수. 0은 백업을 생성하지 않음을 나타냅니다(스냅샷만 생성).
   - **spec.snapshotRetention**: 보관할 스냅샷 수. 0은 스냅샷을 생성하지 않음을 나타냅니다.
   - **spec.granularity**: 일정을 실행해야 하는 빈도입니다. 가능한 값과 필수 연관 필드는 다음과 같습니다.
     - `Hourly`(지정해야 함) `spec.minute` )
     - `Daily`(지정해야 함) `spec.minute` 그리고 `spec.hour` )
     - `Weekly`(지정해야 함) `spec.minute, spec.hour` , 그리고 `spec.dayOfWeek` )
     - `Monthly`(지정해야 함) `spec.minute, spec.hour` , 그리고 `spec.dayOfMonth` )
     - `Custom`
   - **spec.dayOfMonth**: (선택 사항) 일정을 실행해야 하는 날짜(1~31)입니다. 이 필드는 세분성이 설정된 경우 필수입니다. `Monthly` . 값은 문자열로 제공되어야 합니다.
   - **spec.dayOfWeek**: (선택 사항) 일정을 실행해야 하는 요일(0~7). 0 또는 7의 값은 일요일을 나타냅니다. 이 필드는 세분성이 설정된 경우 필수입니다. `Weekly` . 값은 문자열로 제공되어야 합니다.
   - **spec.hour**: (선택 사항) 일정을 실행해야 하는 시간(0~23)입니다. 이 필드는 세분성이 설정된 경우 필수입니다. `Daily` , `Weekly` , 또는 `Monthly` . 값은 문자열로 제공되어야 합니다.
   - **spec.minute**: (선택 사항) 일정을 실행해야 하는 분(0~59)입니다. 이 필드는 세분성이 설정된 경우 필수입니다. `Hourly` , `Daily` , `Weekly` , 또는 `Monthly` . 값은 문자열로 제공되어야 합니다.
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (선택 사항) 이 주석은 전체 백업을 예약하기 위한 규칙을 지정하는 데 사용됩니다. 설정할 수 있습니다 `always` 지속적인 전체 백업을 위해 사용하거나 요구 사항에 맞게 사용자 정의할 수 있습니다. 예를 들어, 일별 단위를 선택하면 전체 백업을 수행할 요일을 지정할 수 있습니다.

     백업 및 스냅샷 일정에 대한 YAML 예:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
  annotations:
    protect.trident.netapp.io/full-backup-rule: "Monday,Thursday"
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: Daily
  hour: "0"
  minute: "0"
```

스냅샷 전용 일정에 대한 YAML 예:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

3. 다음을 채운 후 `trident-protect-schedule-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

**CLI**를 사용하여 일정을 만듭니다.

단계

1. 보호 일정을 만들고 괄호 안의 값을 사용자 환경의 정보로 바꿉니다. 예를 들어:

   사용할 수 있습니다 `tridentctl-protect create schedule --help` 이 명령에 대한 자세한 도움말 정보를 보려면.

   ```
   tridentctl-protect create schedule <my_schedule_name> --appvault
   <my_appvault_name> --app <name_of_app_to_snapshot> --backup
   -retention <how_many_backups_to_retain> --data-mover
   <Kopia_or_Restic> --day-of-month <day_of_month_to_run_schedule>
   --day-of-week <day_of_month_to_run_schedule> --granularity
   <frequency_to_run> --hour <hour_of_day_to_run> --minute
   <minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
   -retention <how_many_snapshots_to_retain> -n <application_namespace>
   --full-backup-rule <string>
   ```

   설정할 수 있습니다 `--full-backup-rule` 플래그를 `always` 지속적인 전체 백업을 위해 사용하거나 요구 사항에 맞게 사용자 정의할 수 있습니다. 예를 들어, 일 단위를 선택하면 전체 백업을 수행할 요일을 지정할 수 있습니다. 예를 들어, 사용 `--full-backup-rule "Monday,Thursday"` 월요일과 목요일에 전체 백업을 예약합니다.

   스냅샷 전용 일정의 경우 다음을 설정합니다. `--backup-retention 0` 0보다 큰 값을 지정합니다. `--snapshot-retention` .

스냅샷 삭제

더 이상 필요하지 않은 예약된 스냅샷이나 주문형 스냅샷을 삭제합니다.

단계

1. 스냅샷과 연관된 스냅샷 CR을 제거합니다.

   ```
   kubectl delete snapshot <snapshot_name> -n my-app-namespace
   ```

백업 삭제

더 이상 필요하지 않은 예약된 백업이나 주문형 백업을 삭제합니다.

회수 정책이 설정되어 있는지 확인하십시오. `Delete` 개체 스토리지에서 모든 백업 데이터를 제거합니다. 정책의 기본 설정은 다음과 같습니다. `Retain` 실수로 데이터가 손실되는 것을 방지합니다. 정책이 변경되지 않으면 `Delete` 백업 데이터는 개체 스토리지에 남아 있으므로 수동으로 삭제해야 합니다.

단계

1. 백업과 관련된 백업 CR을 제거합니다.

```
kubectl delete backup <backup_name> -n my-app-namespace
```

백업 작업 상태 확인

명령줄을 사용하여 진행 중인 백업 작업, 완료된 백업 작업 또는 실패한 백업 작업의 상태를 확인할 수 있습니다.

단계

1. 다음 명령을 사용하여 백업 작업의 상태를 검색하고, 괄호 안의 값을 사용자 환경의 정보로 바꿉니다.

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

**azure-netapp-files(ANF)** 작업에 대한 백업 및 복원 활성화

Trident Protect를 설치한 경우 azure-netapp-files 스토리지 클래스를 사용하고 Trident 24.06 이전에 생성된 스토리지 백엔드에 대해 공간 효율적인 백업 및 복원 기능을 활성화할 수 있습니다. 이 기능은 NFSv4 볼륨에서 작동하며 용량 풀에서 추가 공간을 소모하지 않습니다.

시작하기 전에

다음 사항을 확인하세요.

- Trident Protect를 설치했습니다.
- Trident Protect에서 애플리케이션을 정의했습니다. 이 절차를 완료하기 전까지 이 애플리케이션의 보호 기능은 제한됩니다.
- 당신은 가지고있다 `azure-netapp-files` 스토리지 백엔드의 기본 스토리지 클래스로 선택되었습니다.

구성 단계를 확장하세요

1. Trident 24.10으로 업그레이드하기 전에 ANF 볼륨이 생성된 경우 Trident 에서 다음을 수행하세요.

   a. azure-netapp-files 기반이며 애플리케이션과 연결된 각 PV에 대해 스냅샷 디렉토리를 활성화합니다.

   ```
   tridentctl update volume <pv name> --snapshot-dir=true -n trident
   ```

   b. 연관된 각 PV에 대해 스냅샷 디렉토리가 활성화되었는지 확인하세요.

   ```
   tridentctl get volume <pv name> -n trident -o yaml | grep
   snapshotDir
   ```

   응답:

   ```
   snapshotDirectory: "true"
   ```

   +
   스냅샷 디렉토리가 활성화되어 있지 않으면 정기적인 백업 기능을 선택합니다. 이 기능은 백업 프로세스 중에 용량 풀의 공간을 일시적으로 사용합니다. 이 경우 백업되는 볼륨 크기의 임시 볼륨을 생성할 수 있는 충분한 공간이 용량 풀에 있는지 확인하세요.

결과

Trident Protect를 사용하여 애플리케이션을 백업하고 복원할 준비가 되었습니다. 각 PVC는 다른 애플리케이션에서 백업 및 복원을 위해 사용할 수도 있습니다.

## 응용 프로그램 복원

**Trident Protect**를 사용하여 애플리케이션 복원

Trident Protect를 사용하면 스냅샷이나 백업에서 애플리케이션을 복원할 수 있습니다. 동일한 클러스터로 애플리케이션을 복원할 때 기존 스냅샷에서 복원하는 것이 더 빠릅니다.

- 애플리케이션을 복원하면 해당 애플리케이션에 구성된 모든 실행 후크가 앱과 함께 복원됩니다. 복원 후 실행 후크가 있는 경우 복원 작업의 일부로 자동으로 실행됩니다.
- qtree 볼륨의 경우 백업에서 다른 네임스페이스나 원래 네임스페이스로 복원하는 기능이 지원됩니다. 그러나 qtree 볼륨의 경우 스냅샷에서 다른 네임스페이스나 원래 네임스페이스로 복원하는 작업은 지원되지 않습니다.
- 고급 설정을 사용하여 복원 작업을 사용자 정의할 수 있습니다. 자세한 내용은 다음을 참조하세요. "고급 Trident Protect 복원 설정 사용".

BackupRestore CR을 사용하여 다른 네임스페이스로 백업을 복원하면 Trident Protect는 새 네임스페이스에서 애플리케이션을 복원하고 복원된 애플리케이션에 대한 애플리케이션 CR을 만듭니다. 복구된 애플리케이션을 보호하려면 주문형 백업이나 스냅샷을 만들거나 보호 일정을 설정하세요.

기존 리소스가 있는 다른 네임스페이스에 백업을 복원하더라도 백업에 있는 리소스와 이름을 공유하는 리소스는 변경되지 않습니다. 백업에 있는 모든 리소스를 복원하려면 대상 네임스페이스를 삭제하고 다시 만들거나 백업을 새 네임스페이스로 복원합니다.

### 시작하기 전에

AWS 세션 토큰 만료일이 장기 실행 s3 복원 작업에 충분한지 확인하세요. 복원 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.

- 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
- 를 참조하세요 "AWS IAM 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

Kopia를 데이터 이동자로 사용하여 백업을 복원하는 경우 CR이나 CLI에서 주석을 지정하여 Kopia에서 사용하는 임시 저장소의 동작을 제어할 수 있습니다. 를 참조하세요 "코피아 문서" 구성할 수 있는 옵션에 대한 자세한 내용은 다음을 참조하세요. 사용하다 `tridentctl-protect create --help` Trident Protect CLI로 주석을 지정하는 방법에 대한 자세한 내용은 명령을 참조하세요.

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-backup-restore-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appArchivePath**: 백업 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (필수) 백업 내용이 저장된 AppVault의 이름입니다.
   - **spec.namespaceMapping**: 복원 작업의 소스 네임스페이스를 대상 네임스페이스로 매핑합니다. 바꾸다 `my-source-namespace` 그리고 `my-destination-namespace` 주변 환경의 정보를 활용하여

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (선택 사항) 복원할 애플리케이션의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

   > Trident Protect는 사용자가 선택한 리소스와의 관계로 인해 일부 리소스를 자동으로 선택합니다. 예를 들어, 영구 볼륨 클레임 리소스를 선택하고 연관된 Pod가 있는 경우 Trident Protect는 연관된 Pod도 복원합니다.

   - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `Include` 또는 `Exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
     - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.

- **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
- **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.
- **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.
- **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

  예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 다음을 채운 후 `trident-protect-backup-restore-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 백업을 다른 네임스페이스로 복원합니다. 그만큼 `namespace-mapping` 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 다음 형식의 올바른 대상 네임스페이스에 매핑합니다. `source1:dest1,source2:dest2` . 예를 들어:

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

백업에서 원래 네임스페이스로 복원

언제든지 원래 네임스페이스로 백업을 복원할 수 있습니다.

시작하기 전에

AWS 세션 토큰 만료일이 장기 실행 s3 복원 작업에 충분한지 확인하세요. 복원 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.

- 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
- 를 참조하세요 "AWS IAM 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

Kopia를 데이터 이동자로 사용하여 백업을 복원하는 경우 CR이나 CLI에서 주석을 지정하여 Kopia에서 사용하는 임시 저장소의 동작을 제어할 수 있습니다. 를 참조하세요 "코피아 문서" 구성할 수 있는 옵션에 대한 자세한 내용은 다음을 참조하세요. 사용하다 `tridentctl-protect create --help` Trident Protect CLI로 주석을 지정하는 방법에 대한 자세한 내용은 명령을 참조하세요.

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-backup-ipr-cr.yaml`.
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appArchivePath**: 백업 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (필수) 백업 내용이 저장된 AppVault의 이름입니다.

     예를 들어:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appArchivePath: my-backup-path
  appVaultRef: appvault-name
```

3. (선택 사항) 복원할 애플리케이션의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

   Trident Protect는 사용자가 선택한 리소스와의 관계로 인해 일부 리소스를 자동으로 선택합니다. 예를 들어, 영구 볼륨 클레임 리소스를 선택하고 연관된 Pod가 있는 경우 Trident Protect는 연관된 Pod도 복원합니다.

   - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `Include` 또는 `Exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
     - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.
       - **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
       - **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.

- **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.
- **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

  예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 다음을 채운 후 `trident-protect-backup-ipr-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 백업을 원래 네임스페이스로 복원합니다. 그만큼 `backup` 인수는 형식에서 네임스페이스와 백업 이름을 사용합니다. `<namespace>/<name>` . 예를 들어:

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

원래 클러스터에 문제가 있는 경우 다른 클러스터로 백업을 복원할 수 있습니다.

Kopia를 데이터 이동자로 사용하여 백업을 복원하는 경우 CR이나 CLI에서 주석을 지정하여 Kopia에서 사용하는 임시 저장소의 동작을 제어할 수 있습니다. 를 참조하세요 "코피아 문서" 구성할 수 있는 옵션에 대한 자세한 내용은 다음을 참조하세요. 사용하다 `tridentctl-protect create --help` Trident Protect CLI로 주석을 지정하는 방법에 대한 자세한 내용은 명령을 참조하세요.

시작하기 전에

다음 전제 조건이 충족되는지 확인하세요.

- 대상 클러스터에 Trident Protect가 설치되어 있습니다.
- 대상 클러스터는 백업이 저장된 소스 클러스터와 동일한 AppVault의 버킷 경로에 액세스할 수 있습니다.
- AppVault CR에서 정의된 개체 저장소 버킷에 로컬 환경이 연결할 수 있는지 확인하십시오. `tridentctl-protect get appvaultcontent` 명령. 네트워크 제한으로 인해 액세스가 불가능한 경우 대상 클러스터의 포드 내에서 Trident Protect CLI를 실행하세요.
- AWS 세션 토큰 만료일이 장기 복원 작업에 충분한지 확인하세요. 복원 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.
  - 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
  - 를 참조하세요 "AWS 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

단계

1. Trident Protect CLI 플러그인을 사용하여 대상 클러스터에서 AppVault CR의 가용성을 확인하세요.

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

애플리케이션 복원에 사용되는 네임스페이스가 대상 클러스터에 있는지 확인하세요.

2. 대상 클러스터에서 사용 가능한 AppVault의 백업 내용을 확인합니다.

```
tridentctl-protect get appvaultcontent <appvault_name> \
--show-resources backup \
--show-paths \
--context <destination_cluster_name>
```

이 명령을 실행하면 AppVault에서 사용 가능한 백업이 표시됩니다. 여기에는 원래 클러스터, 해당 애플리케이션 이름, 타임스탬프, 보관 경로가 포함됩니다.

예시 출력:

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |        TIMESTAMP
|    PATH     |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
```

3. AppVault 이름과 보관 경로를 사용하여 대상 클러스터에 애플리케이션을 복원합니다.

**CR**을 사용하세요

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-backup-restore-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appVaultRef**: (필수) 백업 내용이 저장된 AppVault의 이름입니다.
   - **spec.appArchivePath**: 백업 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     BackupRestore CR을 사용할 수 없는 경우 2단계에서 언급한 명령을 사용하여 백업 내용을 볼 수 있습니다.

   - **spec.namespaceMapping**: 복원 작업의 소스 네임스페이스를 대상 네임스페이스로 매핑합니다. 바꾸다 `my-source-namespace` 그리고 `my-destination-namespace` 주변 환경의 정보를 활용하여

     예를 들어:

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-backup-path
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. 다음을 채운 후 `trident-protect-backup-restore-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

**CLI**를 사용하세요

1. 다음 명령을 사용하여 애플리케이션을 복원하고 괄호 안의 값을 사용자 환경의 정보로 바꿉니다. 네임스페이스 매핑 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 source1:dest1,source2:dest2

형식으로 올바른 대상 네임스페이스에 매핑합니다. 예를 들어:

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--context <destination_cluster_name> \
-n <application_namespace>
```

스냅샷에서 다른 네임스페이스로 복원

사용자 정의 리소스(CR) 파일을 사용하여 스냅샷에서 데이터를 다른 네임스페이스나 원래 소스 네임스페이스로 복원할 수 있습니다. SnapshotRestore CR을 사용하여 다른 네임스페이스로 스냅샷을 복원하면 Trident Protect는 새 네임스페이스에서 애플리케이션을 복원하고 복원된 애플리케이션에 대한 애플리케이션 CR을 만듭니다. 복구된 애플리케이션을 보호하려면 주문형 백업이나 스냅샷을 만들거나 보호 일정을 설정하세요.

SnapshotRestore는 다음을 지원합니다. `spec.storageClassMapping` 속성이지만 소스 및 대상 저장소 클래스가 동일한 저장소 백엔드를 사용하는 경우에만 해당됩니다. 복원을 시도하는 경우 `StorageClass` 다른 스토리지 백엔드를 사용하는 경우 복원 작업이 실패합니다.

시작하기 전에

AWS 세션 토큰 만료일이 장기 실행 s3 복원 작업에 충분한지 확인하세요. 복원 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.

- 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
- 를 참조하세요 "AWS IAM 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-snapshot-restore-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appVaultRef**: (필수) 스냅샷 내용이 저장된 AppVault의 이름입니다.
   - **spec.appArchivePath**: 스냅샷 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.namespaceMapping**: 복원 작업의 소스 네임스페이스를 대상 네임스페이스로 매핑합니다. 바꾸다 `my-source-namespace` 그리고 `my-destination-namespace` 주변 환경의 정보를 활용하여

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (선택 사항) 복원할 애플리케이션의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

   > Trident Protect는 사용자가 선택한 리소스와의 관계로 인해 일부 리소스를 자동으로 선택합니다. 예를 들어, 영구 볼륨 클레임 리소스를 선택하고 연관된 Pod가 있는 경우 Trident Protect는 연관된 Pod도 복원합니다.

   - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `Include` 또는 `Exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
     - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.

- **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
- **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.
- **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.
- **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

  예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 다음을 채운 후 `trident-protect-snapshot-restore-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 스냅샷을 다른 네임스페이스로 복원합니다.
   - 그만큼 `snapshot` 인수는 형식에 네임스페이스와 스냅샷 이름을 사용합니다. `<namespace>/<name>` .
   - 그만큼 `namespace-mapping` 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 형식에 맞는 올바른 대상 네임스페이스에 매핑합니다. `source1:dest1,source2:dest2` .

예를 들어:

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

### 스냅샷에서 원래 네임스페이스로 복원

언제든지 원래 네임스페이스로 스냅샷을 복원할 수 있습니다.

> 애플리케이션에서 여러 네임스페이스를 사용하고 이러한 네임스페이스에 동일한 이름의 PVC가 있는 경우 스냅샷 복원 작업(제자리 복원 및 새 네임스페이스로의 복원 모두)이 올바르게 작동하지 않습니다. 복원된 모든 볼륨에 각 네임스페이스에 맞는 올바른 데이터가 아닌 동일한 데이터가 저장됩니다. 스냅샷 복원 대신 백업 복원을 사용하거나 이 문제가 해결된 버전 26.02 이상으로 업그레이드하십시오.

#### 시작하기 전에

AWS 세션 토큰 만료일이 장기 실행 s3 복원 작업에 충분한지 확인하세요. 복원 작업 중에 토큰이 만료되면 작업이 실패할 수 있습니다.

- 를 참조하세요 "AWS API 문서" 현재 세션 토큰 만료일을 확인하는 방법에 대한 자세한 내용은 다음을 참조하세요.
- 를 참조하세요 "AWS IAM 문서" AWS 리소스에 대한 자격 증명에 대한 자세한 내용은 다음을 참조하세요.

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-snapshot-ipr-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appVaultRef**: (필수) 스냅샷 내용이 저장된 AppVault의 이름입니다.
   - **spec.appArchivePath**: 스냅샷 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

```
kubectl get snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

```
---
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appVaultRef: appvault-name
    appArchivePath: my-snapshot-path
```

3. (선택 사항) 복원할 애플리케이션의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

> ⓘ Trident Protect는 사용자가 선택한 리소스와의 관계로 인해 일부 리소스를 자동으로 선택합니다. 예를 들어, 영구 볼륨 클레임 리소스를 선택하고 연관된 Pod가 있는 경우 Trident Protect는 연관된 Pod도 복원합니다.

   - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `Include` 또는 `Exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
     - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.
       - **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
       - **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.
       - **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.
       - **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name

필드에 있는 이름입니다.

- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

  예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 다음을 채운 후 `trident-protect-snapshot-ipr-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용합니다.

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 스냅샷을 원래 네임스페이스로 복원합니다. 예를 들어:

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <snapshot_to_restore> \
-n <application_namespace>
```

### 복원 작업 상태 확인

명령줄을 사용하여 진행 중인 복원 작업, 완료된 복원 작업 또는 실패한 복원 작업의 상태를 확인할 수 있습니다.

단계

1. 다음 명령을 사용하여 복원 작업의 상태를 검색하고, 괄호 안의 값을 사용자 환경의 정보로 바꿉니다.

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

### 고급 **Trident Protect** 복원 설정 사용

## 주석, 네임스페이스 설정, 스토리지 옵션 등의 고급 설정을 사용하여 복원 작업을 사용자 정의하여 특정 요구 사항을 충족할 수 있습니다.

#### 복원 및 장애 조치 작업 중 네임스페이스 주석 및 레이블

복원 및 장애 조치 작업 동안 대상 네임스페이스의 레이블과 주석은 소스 네임스페이스의 레이블과 주석과 일치하도록 만들어집니다. 대상 네임스페이스에 없는 소스 네임스페이스의 레이블이나 주석이 추가되고, 이미 존재하는 레이블이나 주석은 소스 네임스페이스의 값과 일치하도록 덮어씁니다. 대상 네임스페이스에만 존재하는 레이블이나 주석은 변경되지 않습니다.

> Red Hat OpenShift를 사용하는 경우 OpenShift 환경에서 네임스페이스 주석의 중요한 역할을 알아두는 것이 중요합니다. 네임스페이스 주석은 복원된 포드가 OpenShift 보안 컨텍스트 제약(SCC)에 의해 정의된 적절한 권한과 보안 구성을 준수하고 권한 문제 없이 볼륨에 액세스할 수 있도록 보장합니다. 자세한 내용은 다음을 참조하세요. "OpenShift 보안 컨텍스트 제약 조건 문서" .

Kubernetes 환경 변수를 설정하여 대상 네임스페이스의 특정 주석이 덮어쓰여지는 것을 방지할 수 있습니다. `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` 복원이나 장애 조치 작업을 수행하기 전에. 예를 들어:

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

> 복원 또는 장애 조치 작업을 수행할 때 지정된 모든 네임스페이스 주석 및 레이블 `restoreSkipNamespaceAnnotations` 그리고 `restoreSkipNamespaceLabels` 복구 또는 장애 조치 작업에서 제외됩니다. 이러한 설정은 Helm을 처음 설치할 때 구성되어야 합니다. 자세한 내용은 다음을 참조하세요. "AutoSupport 및 네임스페이스 필터링 옵션 구성".

Helm을 사용하여 소스 애플리케이션을 설치한 경우 `--create-namespace` 깃발에는 특별한 대우가 주어집니다. `name` 라벨 키. 복구 또는 장애 조치 프로세스 중에 Trident Protect는 이 레이블을 대상 네임스페이스에 복사하지만 소스의 값이 소스 네임스페이스와 일치하는 경우 값을 대상 네임스페이스 값으로 업데이트합니다. 이 값이 소스 네임스페이스와 일치하지 않으면 변경 사항 없이 대상 네임스페이스에 복사됩니다.

예

다음 예제에서는 각각 다른 주석과 레이블이 있는 소스 및 대상 네임스페이스를 보여줍니다. 작업 전후의 대상 네임스페이스 상태를 볼 수 있으며, 대상 네임스페이스에서 주석과 레이블이 어떻게 결합되거나 덮어쓰여지는지 확인할 수 있습니다.

복구 또는 장애 조치 작업 전

다음 표는 복원 또는 장애 조치 작업 전의 예제 소스 및 대상 네임스페이스의 상태를 보여줍니다.

| 네임스페이스 | 주석 | 라벨 |
| --- | --- | --- |
| 네임스페이스 ns-1(소스) | • annotation.one/key: "업데이트된 값"<br>• annotation.two/key: "true" | • 환경=생산<br>• 규정 준수=HIPAA<br>• 이름=ns-1 |
| 네임스페이스 ns-2(대상) | • annotation.one/key: "true"<br>• annotation.three/key: "false" | • 역할=데이터베이스 |

복원 작업 후

다음 표는 복원 또는 장애 조치 작업 후의 대상 네임스페이스의 상태를 보여줍니다. 일부 키가 추가되었고 일부는 덮어쓰여졌으며 `name` 레이블이 대상 네임스페이스와 일치하도록 업데이트되었습니다.

| 네임스페이스 | 주석 | 라벨 |
| --- | --- | --- |
| 네임스페이스 ns-2(대상) | • annotation.one/key: "업데이트된 값"<br>• annotation.two/key: "true"<br>• annotation.three/key: "false" | • 이름=ns-2<br>• 규정 준수=HIPAA<br>• 환경=생산<br>• 역할=데이터베이스 |

지원되는 필드

이 섹션에서는 복원 작업에 사용할 수 있는 추가 필드에 대해 설명합니다.

스토리지 클래스 매핑

그만큼 `spec.storageClassMapping` 속성은 소스 애플리케이션에 있는 스토리지 클래스에서 대상 클러스터의 새 스토리지 클래스로의 매핑을 정의합니다. 서로 다른 스토리지 클래스를 사용하는 클러스터 간에 애플리케이션을 마이그레이션하거나 BackupRestore 작업에 대한 스토리지 백엔드를 변경할 때 이 기능을 사용할 수 있습니다.

예:

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

지원되는 주석

이 섹션에서는 시스템의 다양한 동작을 구성하는 데 지원되는 주석을 나열합니다. 사용자가 주석을 명시적으로 설정하지 않으면 시스템은 기본값을 사용합니다.

| 주석 | 유형 | 설명 | 기본값 |
|---|---|---|---|
| protect.trident.netapp.io/데이터 이동 시간 초과 초 | 끈 | 데이터 이동 작업이 중단될 수 있는 최대 시간(초)입니다. | "300" |
| protect.trident.netapp.io/kopia-content-cache-size-limit-mb | 끈 | Kopia 콘텐츠 캐시의 최대 크기 제한(메가바이트)입니다. | "1000" |

## NetApp SnapMirror 및 Trident Protect를 사용하여 애플리케이션 복제

Trident Protect를 사용하면 NetApp SnapMirror 기술의 비동기 복제 기능을 사용하여 동일한 클러스터 또는 서로 다른 클러스터 간에 데이터 및 애플리케이션 변경 사항을 한 스토리지 백엔드에서 다른 스토리지 백엔드로 복제할 수 있습니다.

복원 및 장애 조치 작업 중 네임스페이스 주석 및 레이블

복원 및 장애 조치 작업 동안 대상 네임스페이스의 레이블과 주석은 소스 네임스페이스의 레이블과 주석과 일치하도록 만들어집니다. 대상 네임스페이스에 없는 소스 네임스페이스의 레이블이나 주석이 추가되고, 이미 존재하는 레이블이나 주석은 소스 네임스페이스의 값과 일치하도록 덮어씁니다. 대상 네임스페이스에만 존재하는 레이블이나 주석은 변경되지 않습니다.

> Red Hat OpenShift를 사용하는 경우 OpenShift 환경에서 네임스페이스 주석의 중요한 역할을 알아두는 것이 중요합니다. 네임스페이스 주석은 복원된 포드가 OpenShift 보안 컨텍스트 제약(SCC)에 의해 정의된 적절한 권한과 보안 구성을 준수하고 권한 문제 없이 볼륨에 액세스할 수 있도록 보장합니다. 자세한 내용은 다음을 참조하세요. "OpenShift 보안 컨텍스트 제약 조건 문서" .

Kubernetes 환경 변수를 설정하여 대상 네임스페이스의 특정 주석이 덮어쓰여지는 것을 방지할 수 있습니다. `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` 복원이나 장애 조치 작업을 수행하기 전에. 예를 들어:

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

복원 또는 장애 조치 작업을 수행할 때 지정된 모든 네임스페이스 주석 및 레이블 `restoreSkipNamespaceAnnotations` 그리고 `restoreSkipNamespaceLabels` 복구 또는 장애 조치 작업에서 제외됩니다. 이러한 설정은 Helm을 처음 설치할 때 구성되어야 합니다. 자세한 내용은 다음을 참조하세요. "AutoSupport 및 네임스페이스 필터링 옵션 구성".

Helm을 사용하여 소스 애플리케이션을 설치한 경우 `--create-namespace` 깃발에는 특별한 대우가 주어집니다. `name` 라벨 키. 복구 또는 장애 조치 프로세스 중에 Trident Protect는 이 레이블을 대상 네임스페이스에 복사하지만 소스의 값이 소스 네임스페이스와 일치하는 경우 값을 대상 네임스페이스 값으로 업데이트합니다. 이 값이 소스 네임스페이스와 일치하지 않으면 변경 사항 없이 대상 네임스페이스에 복사됩니다.

예

다음 예제에서는 각각 다른 주석과 레이블이 있는 소스 및 대상 네임스페이스를 보여줍니다. 작업 전후의 대상 네임스페이스 상태를 볼 수 있으며, 대상 네임스페이스에서 주석과 레이블이 어떻게 결합되거나 덮어쓰여지는지 확인할 수 있습니다.

복구 또는 장애 조치 작업 전

다음 표는 복원 또는 장애 조치 작업 전의 예제 소스 및 대상 네임스페이스의 상태를 보여줍니다.

| 네임스페이스 | 주석 | 라벨 |
|---|---|---|
| 네임스페이스 ns-1(소스) | • annotation.one/key: "업데이트된 값"<br>• annotation.two/key: "true" | • 환경=생산<br>• 규정 준수=HIPAA<br>• 이름=ns-1 |
| 네임스페이스 ns-2(대상) | • annotation.one/key: "true"<br>• annotation.three/key: "false" | • 역할=데이터베이스 |

복원 작업 후

다음 표는 복원 또는 장애 조치 작업 후의 대상 네임스페이스의 상태를 보여줍니다. 일부 키가 추가되었고 일부는 덮어쓰여졌으며 `name` 레이블이 대상 네임스페이스와 일치하도록 업데이트되었습니다.

| 네임스페이스 | 주석 | 라벨 |
|---|---|---|
| 네임스페이스 ns-2(대상) | • annotation.one/key: "업데이트된 값"<br>• annotation.two/key: "true"<br>• annotation.three/key: "false" | • 이름=ns-2<br>• 규정 준수=HIPAA<br>• 환경=생산<br>• 역할=데이터베이스 |

데이터 보호 작업 중에 파일 시스템을 동결 및 동결 해제하도록 Trident Protect를 구성할 수 있습니다. "Trident Protect를 사용하여 파일 시스템 동결을 구성하는 방법에 대해 자세히 알아보세요.".

### 장애 조치 및 역방향 작업 중 실행 후크

AppMirror 관계를 사용하여 애플리케이션을 보호하는 경우 장애 조치 및 역방향 작업 중에 알아야 할 실행 후크와 관련된 특정 동작이 있습니다.

- 장애 조치 중에 실행 후크는 소스 클러스터에서 대상 클러스터로 자동으로 복사됩니다. 수동으로 다시 만들 필요는 없습니다. 장애 조치 후에는 애플리케이션에 실행 후크가 존재하며 관련 작업 중에 실행됩니다.
- 역방향 또는 역방향 재동기화 중에 애플리케이션의 기존 실행 후크는 모두 제거됩니다. 소스 애플리케이션이 대상 애플리케이션이 되면 이러한 실행 후크는 유효하지 않으며 실행을 방지하기 위해 삭제됩니다.

실행 후크에 대해 자세히 알아보려면 다음을 참조하세요."Trident Protect 실행 후크 관리" .

### 복제 관계 설정

복제 관계를 설정하는 데는 다음이 포함됩니다.

- Trident Protect가 앱 스냅샷을 얼마나 자주 찍을지 선택합니다(여기에는 앱의 Kubernetes 리소스와 각 앱 볼륨에 대한 볼륨 스냅샷이 포함됨)
- 복제 일정 선택(Kubernetes 리소스 및 영구 볼륨 데이터 포함)
- 스냅샷을 찍을 시간 설정

단계

1. 소스 클러스터에서 소스 애플리케이션에 대한 AppVault를 만듭니다. 저장소 공급자에 따라 다음 예제를 수정하세요."AppVault 사용자 정의 리소스" 귀하의 환경에 맞게:

**CR**을 사용하여 **AppVault** 만들기

a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-appvault-primary-source.yaml` ).

b. 다음 속성을 구성합니다.

- **metadata.name**: (필수) AppVault 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 이름을 기록해 두세요.
- **spec.providerConfig**: (필수) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 구성을 저장합니다. 공급자에 대한 버킷 이름과 기타 필요한 세부 정보를 선택하세요. 복제 관계에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 값을 기록해 두세요. 참조하다"AppVault 사용자 정의 리소스" 다른 공급업체와의 AppVault CR 사례는 다음과 같습니다.
- **spec.providerCredentials**: (필수) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 자격 증명에 대한 참조를 저장합니다.
  - **spec.providerCredentials.valueFromSecret**: (필수) 자격 증명 값이 비밀에서 나와야 함을 나타냅니다.
    - 키: (필수) 선택할 비밀번호의 유효한 키입니다.
    - **name**: (필수) 이 필드의 값을 포함하는 비밀의 이름입니다. 같은 네임스페이스에 있어야 합니다.
  - **spec.providerCredentials.secretAccessKey**: (필수) 공급자에 액세스하는 데 사용되는 액세스 키입니다. *name*은 *spec.providerCredentials.valueFromSecret.name*과 일치해야 합니다.
- **spec.providerType**: (필수) 백업을 제공하는 항목을 결정합니다. 예: NetApp ONTAP S3, 일반 S3, Google Cloud 또는 Microsoft Azure. 가능한 값:
  - AWS
  - 하늘빛
  - 지씨피
  - 제네릭-s3
  - 온탭-s3
  - 스토리지그리드-s3

c. 다음을 채운 후 `trident-protect-appvault-primary-source.yaml` 올바른 값으로 파일을 만들고 CR을 적용합니다.

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

**CLI**를 사용하여 **AppVault** 만들기

a. 괄호 안의 값을 사용자 환경의 정보로 바꿔 AppVault를 만듭니다.

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. 소스 클러스터에서 소스 애플리케이션 CR을 만듭니다.

**CR**을 사용하여 소스 애플리케이션을 만듭니다.

a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-app-source.yaml` ).

b. 다음 속성을 구성합니다.

- **metadata.name**: (필수) 애플리케이션 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 이름을 기록해 두세요.
- **spec.includedNamespaces**: (필수) 네임스페이스와 관련 레이블의 배열입니다. 네임스페이스 이름을 사용하고 필요에 따라 레이블을 사용하여 네임스페이스의 범위를 좁혀 여기에 나열된 네임스페이스에 있는 리소스를 지정합니다. 애플리케이션 네임스페이스는 이 배열의 일부여야 합니다.

  **YAML** 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. 다음을 채운 후 `trident-protect-app-source.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-namespace
```

**CLI**를 사용하여 소스 애플리케이션 만들기

a. 소스 애플리케이션을 생성합니다. 예를 들어:

```
tridentctl-protect create app <my-app-name> --namespaces
<namespaces-to-be-included> -n <my-app-namespace>
```

3. 선택적으로 소스 클러스터에서 소스 애플리케이션의 스냅샷을 만듭니다. 이 스냅샷은 대상 클러스터의 애플리케이션의 기반으로 사용됩니다. 이 단계를 건너뛰면 다음 예약된 스냅샷이 실행될 때까지 기다려야 최신 스냅샷을 얻을 수 있습니다. 주문형 스냅샷을 생성하려면 다음을 참조하세요. "주문형 스냅샷 만들기".
4. 소스 클러스터에서 복제 일정 CR을 만듭니다.

아래에 제공된 일정과 함께, 피어링된 ONTAP 클러스터 간에 공통 스냅샷을 유지하려면 7일의 보존 기간을 갖는 별도의 일일 스냅샷 일정을 만드는 것이 좋습니다. 이를 통해 스냅샷을 최대 7일 동안 사용할 수 있지만, 보존 기간은 사용자 요구 사항에 따라 사용자 정의할 수 있습니다.

장애 조치가 발생하면 시스템은 이러한 스냅샷을 최대 7일 동안 역방향 작업에 사용할 수 있습니다. 이 접근 방식을 사용하면 마지막 스냅샷 이후에 변경된 내용만 전송되고 모든 데이터는 전송되지 않으므로 역방향 프로세스가 더 빠르고 효율적입니다.

해당 애플리케이션의 기존 일정이 이미 원하는 보존 요구 사항을 충족하는 경우 추가 일정은 필요하지 않습니다.

**CR**을 사용하여 복제 일정을 만듭니다.

a. 소스 애플리케이션에 대한 복제 일정을 만듭니다.

i. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-schedule.yaml` ).

ii. 다음 속성을 구성합니다.

- **metadata.name**: (필수) 일정 사용자 정의 리소스의 이름입니다.
- **spec.appVaultRef**: (필수) 이 값은 소스 애플리케이션의 AppVault의 metadata.name 필드와 일치해야 합니다.
- **spec.applicationRef**: (필수) 이 값은 소스 애플리케이션 CR의 metadata.name 필드와 일치해야 합니다.
- **spec.backupRetention**: (필수) 이 필드는 필수이며, 값은 0으로 설정해야 합니다.
- **spec.enabled**: true로 설정해야 합니다.
- **spec.granularity**: 설정해야 함 `Custom` .
- **spec.recurrenceRule**: UTC 시간으로 시작 날짜와 반복 간격을 정의합니다.
- **spec.snapshotRetention**: 2로 설정해야 합니다.

YAML 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

i. 다음을 채운 후 `trident-protect-schedule.yaml` 올바른 값으로 파일을 만들고 CR을 적용합니다.

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-namespace
```

**CLI**를 사용하여 복제 일정을 만듭니다.

a. 괄호 안의 값을 사용자 환경의 정보로 바꿔 복제 일정을 만듭니다.

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

예:

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. 대상 클러스터에서 소스 클러스터에 적용한 AppVault CR과 동일한 소스 애플리케이션 AppVault CR을 만들고 이름을 지정합니다(예: `trident-protect-appvault-primary-destination.yaml` ).
6. CR을 적용하세요:

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
trident-protect
```

7. 대상 클러스터의 대상 애플리케이션에 대한 대상 AppVault CR을 만듭니다. 저장소 공급자에 따라 다음 예제를 수정하세요."AppVault 사용자 정의 리소스" 귀하의 환경에 맞게:
   a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-appvault-secondary-destination.yaml` ).
   b. 다음 속성을 구성합니다.
      - **metadata.name**: (필수) AppVault 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 이름을 기록해 두세요.
      - **spec.providerConfig**: (필수) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 구성을 저장합니다. 선택하다 `bucketName` 그리고 귀하의 공급업체에 대한 기타 필요한 세부 정보입니다. 복제 관계에 필요한 다른 CR 파일이 이 값을 참조하므로 선택한 값을 기록해 두세요. 참조하다"AppVault 사용자 정의 리소스" 다른 공급업체와의 AppVault CR 사례는 다음과 같습니다.
      - **spec.providerCredentials**: (필수) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 자격 증명에 대한 참조를 저장합니다.
        - **spec.providerCredentials.valueFromSecret**: (필수) 자격 증명 값이 비밀에서 나와야 함을 나타냅니다.
          - 키: (필수) 선택할 비밀번호의 유효한 키입니다.
          - **name**: (필수) 이 필드의 값을 포함하는 비밀의 이름입니다. 같은 네임스페이스에 있어야 합니다.

        - **spec.providerCredentials.secretAccessKey**: (필수) 공급자에 액세스하는 데 사용되는 액세스 키입니다. *name*은 *spec.providerCredentials.valueFromSecret.name*과 일치해야 합니다.
    - **spec.providerType**: (필수) 백업을 제공하는 항목을 결정합니다. 예: NetApp ONTAP S3, 일반 S3, Google Cloud 또는 Microsoft Azure. 가능한 값:
        - AWS
        - 하늘빛
        - 지씨피
        - 제네릭-s3
        - 온탭-s3
        - 스토리지그리드-s3

c. 다음을 채운 후 `trident-protect-appvault-secondary-destination.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n trident-protect
```

8. 대상 클러스터에서 AppMirrorRelationship CR 파일을 만듭니다.

**CR**을 사용하여 **AppMirrorRelationship**을 만듭니다**.**

a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-relationship.yaml` ).

b. 다음 속성을 구성합니다.

- **metadata.name:** (필수) AppMirrorRelationship 사용자 정의 리소스의 이름입니다.
- **spec.destinationAppVaultRef**: (필수) 이 값은 대상 클러스터의 대상 애플리케이션에 대한 AppVault 이름과 일치해야 합니다.
- **spec.namespaceMapping**: (필수) 대상 및 소스 네임스페이스는 해당 애플리케이션 CR에 정의된 애플리케이션 네임스페이스와 일치해야 합니다.
- **spec.sourceAppVaultRef**: (필수) 이 값은 소스 애플리케이션의 AppVault 이름과 일치해야 합니다.
- **spec.sourceApplicationName**: (필수) 이 값은 소스 애플리케이션 CR에 정의한 소스 애플리케이션의 이름과 일치해야 합니다.
- **spec.sourceApplicationUID**: (필수) 이 값은 소스 애플리케이션 CR에서 정의한 소스 애플리케이션의 UID와 일치해야 합니다.
- **spec.storageClassName**: (선택 사항) 클러스터에서 유효한 스토리지 클래스의 이름을 선택합니다. 스토리지 클래스는 소스 환경과 피어링된 ONTAP 스토리지 VM에 연결되어야 합니다. 스토리지 클래스가 제공되지 않으면 기본적으로 클러스터의 기본 스토리지 클래스가 사용됩니다.
- **spec.recurrenceRule**: UTC 시간으로 시작 날짜와 반복 간격을 정의합니다.

  YAML 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. 다음을 채운 후 `trident-protect-relationship.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

**CLI**를 사용하여 **AppMirrorRelationship**을 만듭니다.

a. AppMirrorRelationship 객체를 만들고 적용하며, 괄호 안의 값을 사용자 환경의 정보로 바꿉니다.

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --source-app-vault <my_vault_name> --recurrence
-rule <rule> --namespace-mapping <ns_mapping> --source-app-id
<source_app_UID> --source-app <my_source_app_name> --storage
-class <storage_class_name> -n <application_namespace>
```

예:

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. (선택 사항) 대상 클러스터에서 복제 관계의 상태를 확인합니다.

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

대상 클러스터로 장애 조치

Trident Protect를 사용하면 복제된 애플리케이션을 대상 클러스터로 장애 조치할 수 있습니다. 이 절차는 복제 관계를 중지하고 대상 클러스터에서 앱을 온라인으로 전환합니다. Trident Protect는 소스 클러스터에서 앱이 작동 중이면 앱을 중지하지 않습니다.

단계

1. 대상 클러스터에서 AppMirrorRelationship CR 파일을 편집합니다(예: `trident-protect-relationship.yaml` ) 및 **spec.desiredState** 값을 다음으로 변경합니다. `Promoted` .

2. CR 파일을 저장합니다.
3. CR을 적용하세요:

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. (선택 사항) 장애 조치된 애플리케이션에 필요한 보호 일정을 만듭니다.
5. (선택 사항) 복제 관계의 상태를 확인합니다.

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

장애가 발생한 복제 관계 재동기화

재동기화 작업은 복제 관계를 재설정합니다. 재동기화 작업을 수행한 후에는 원래 소스 애플리케이션이 실행 중인 애플리케이션이 되고 대상 클러스터에서 실행 중인 애플리케이션에 적용된 모든 변경 사항은 삭제됩니다.

이 프로세스는 복제를 재설정하기 전에 대상 클러스터에서 앱을 중지합니다.

장애 조치 중에 대상 애플리케이션에 기록된 모든 데이터는 손실됩니다.

단계

1. 선택 사항: 소스 클러스터에서 소스 애플리케이션의 스냅샷을 만듭니다. 이렇게 하면 소스 클러스터의 최신 변경 사항이 캡처됩니다.
2. 대상 클러스터에서 AppMirrorRelationship CR 파일을 편집합니다(예: `trident-protect-relationship.yaml` ) 및 spec.desiredState의 값을 다음으로 변경합니다. `Established` .
3. CR 파일을 저장합니다.
4. CR을 적용하세요:

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. 장애 조치된 애플리케이션을 보호하기 위해 대상 클러스터에 보호 일정을 만든 경우 해당 일정을 제거합니다. 남아 있는 일정으로 인해 볼륨 스냅샷 오류가 발생합니다.

장애가 발생한 복제 관계의 역방향 재동기화

장애 조치된 복제 관계를 역방향 재동기화하면 대상 애플리케이션이 소스 애플리케이션이 되고, 소스가 대상이 됩니다. 장애 조치 중에 대상 애플리케이션에 적용된 변경 사항은 유지됩니다.

단계

1. 원래 대상 클러스터에서 AppMirrorRelationship CR을 삭제합니다. 이로 인해 목적지가 소스가 됩니다. 새 대상 클러스터에 보호 일정이 남아 있으면 제거합니다.
2. 원래 관계를 설정하는 데 사용했던 CR 파일을 반대 클러스터에 적용하여 복제 관계를 설정합니다.

3. 새로운 대상(원래 소스 클러스터)이 두 AppVault CR로 구성되었는지 확인하세요.
4. 반대쪽 클러스터에 복제 관계를 설정하고 역방향에 대한 값을 구성합니다.

역방향 애플리케이션 복제 방향

복제 방향을 반대로 하면 Trident Protect는 원래 소스 스토리지 백엔드로 복제를 계속 진행하면서 애플리케이션을 대상 스토리지 백엔드로 이동합니다. Trident Protect는 소스 애플리케이션을 중지하고 대상 앱으로 장애 조치하기 전에 대상에 데이터를 복제합니다.

이 상황에서는 소스와 목적지가 바뀌게 됩니다.

단계

1. 소스 클러스터에서 종료 스냅샷을 만듭니다.

**CR**을 사용하여 종료 스냅샷을 만듭니다.

a. 소스 애플리케이션에 대한 보호 정책 일정을 비활성화합니다.

b. ShutdownSnapshot CR 파일을 만듭니다.

i. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-shutdownsnapshot.yaml` ).

ii. 다음 속성을 구성합니다.

- **metadata.name**: (필수) 사용자 정의 리소스의 이름입니다.
- **spec.AppVaultRef**: (필수) 이 값은 소스 애플리케이션의 AppVault의 metadata.name 필드와 일치해야 합니다.
- **spec.ApplicationRef**: (필수) 이 값은 소스 애플리케이션 CR 파일의 metadata.name 필드와 일치해야 합니다.

YAML 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-
c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-
46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. 다음을 채운 후 `trident-protect-shutdownsnapshot.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-
namespace
```

**CLI**를 사용하여 종료 스냅샷을 만듭니다.

a. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 종료 스냅샷을 만듭니다. 예를 들어:

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot>
--appvault <my_vault> --app <app_to_snapshot> -n
<application_namespace>
```

2. 소스 클러스터에서 종료 스냅샷이 완료된 후 종료 스냅샷의 상태를 가져옵니다.

```
kubectl get shutdownsnapshot -n my-app-namespace
<shutdown_snapshot_name> -o yaml
```

3. 소스 클러스터에서 다음 명령을 사용하여 *shutdownsnapshot.status.appArchivePath*의 값을 찾고 파일 경로의 마지막 부분(기본 이름이라고도 함. 마지막 슬래시 뒤의 모든 내용이 됩니다)을 기록합니다.

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 다음 변경 사항을 적용하여 새 대상 클러스터에서 새 소스 클러스터로 장애 조치를 수행합니다.

> 장애 조치 절차의 2단계에서 다음을 포함합니다. `spec.promotedSnapshot` AppMirrorRelationship CR 파일의 필드를 설정하고 해당 값을 위 3단계에서 기록한 기본 이름으로 설정합니다.

5. 역방향 재동기화 단계를 수행하십시오.장애가 발생한 복제 관계의 역방향 재동기화 .
6. 새 소스 클러스터에서 보호 일정을 활성화합니다.

결과

다음 동작은 역방향 복제로 인해 발생합니다.

- 원본 소스 앱의 Kubernetes 리소스에 대한 스냅샷이 생성됩니다.
- 원래 소스 앱의 Pod는 앱의 Kubernetes 리소스를 삭제하여 정상적으로 중지됩니다(PVC와 PV는 그대로 유지).
- 포드가 종료된 후 앱 볼륨의 스냅샷이 촬영되어 복제됩니다.
- SnapMirror 관계가 끊어져 대상 볼륨을 읽기/쓰기할 수 있게 되었습니다.
- 앱의 Kubernetes 리소스는 원래 소스 앱이 종료된 후 복제된 볼륨 데이터를 사용하여 종료 전 스냅샷에서 복원됩니다.
- 복제는 역방향으로 재설정됩니다.

원래 소스 클러스터로 애플리케이션을 장애 복구합니다.

Trident Protect를 사용하면 다음과 같은 작업 순서를 통해 장애 조치 작업 후에 "장애 복구"를 달성할 수 있습니다. 원래 복제 방향을 복원하기 위한 이 워크플로에서 Trident Protect는 복제 방향을 반전하기 전에 모든 애플리케이션 변경 사항을 원래 소스 애플리케이션으로 복제(재동기화)합니다.

이 프로세스는 대상에 대한 장애 조치를 완료한 관계에서 시작되며 다음 단계를 포함합니다.

- 실패한 상태로 시작합니다.
- 복제 관계를 역방향으로 재동기화합니다.

장애 조치 절차 중에 대상 클러스터에 기록된 데이터가 삭제되므로 일반적인 재동기화 작업을 수행하지 마세요.

- 복제 방향을 반대로 합니다.

단계

1. 수행하다장애가 발생한 복제 관계의 역방향 재동기화 단계.
2. 수행하다역방향 애플리케이션 복제 방향 단계.

복제 관계 삭제

언제든지 복제 관계를 삭제할 수 있습니다. 애플리케이션 복제 관계를 삭제하면 서로 관계가 없는 두 개의 별도 애플리케이션이 생성됩니다.

단계

1. 현재 대상 클러스터에서 AppMirrorRelationship CR을 삭제합니다.

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

## Trident Protect를 사용하여 애플리케이션 마이그레이션

### 백업 데이터를 복원하여 클러스터 간이나 다른 스토리지 클래스로 애플리케이션을 마이그레이션할 수 있습니다.

애플리케이션을 마이그레이션하면 해당 애플리케이션에 구성된 모든 실행 후크가 앱과 함께 마이그레이션됩니다. 복원 후 실행 후크가 있는 경우 복원 작업의 일부로 자동으로 실행됩니다.

백업 및 복원 작업

다음 시나리오에 대한 백업 및 복원 작업을 수행하려면 특정 백업 및 복원 작업을 자동화할 수 있습니다.

동일한 클러스터에 복제

동일한 클러스터에 애플리케이션을 복제하려면 스냅샷이나 백업을 생성하고 동일한 클러스터에 데이터를 복원합니다.

단계

1. 다음 중 하나를 수행하세요.
   a. "스냅샷 만들기".
   b. "백업을 만듭니다".
2. 동일한 클러스터에서 스냅샷이나 백업을 생성했는지에 따라 다음 중 하나를 수행합니다.
   a. "스냅샷에서 데이터 복원".
   b. "백업에서 데이터를 복원하세요".

다른 클러스터에 애플리케이션을 복제하려면(클러스터 간 복제 수행) 소스 클러스터에 백업을 만든 다음, 해당 백업을 다른 클러스터로 복원합니다. 대상 클러스터에 Trident Protect가 설치되어 있는지 확인하세요.

ⓘ 다음을 사용하여 다른 클러스터 간에 애플리케이션을 복제할 수 있습니다."SnapMirror 복제" .

단계

1. "백업을 만듭니다".
2. 백업이 포함된 개체 스토리지 버킷에 대한 AppVault CR이 대상 클러스터에 구성되었는지 확인하세요.
3. 대상 클러스터에서"백업에서 데이터를 복원합니다" .

한 스토리지 클래스에서 다른 스토리지 클래스로 애플리케이션 마이그레이션

대상 스토리지 클래스에 백업을 복원하여 애플리케이션을 한 스토리지 클래스에서 다른 스토리지 클래스로 마이그레이션할 수 있습니다.

예를 들어 (복원 CR의 비밀 제외):

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    - destination: "${destinationNamespace}"
      source: "${sourceNamespace}"
  storageClassMapping:
    - destination: "${destinationStorageClass}"
      source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

**CR**을 사용하여 스냅샷을 복원합니다.

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-snapshot-restore-cr.yaml` .
2. 생성한 파일에서 다음 속성을 구성합니다.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.appArchivePath**: 스냅샷 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (필수) 스냅샷 내용이 저장된 AppVault의 이름입니다.
   - **spec.namespaceMapping**: 복원 작업의 소스 네임스페이스를 대상 네임스페이스로 매핑합니다. 바꾸다 `my-source-namespace` 그리고 `my-destination-namespace` 주변 환경의 정보를 활용하여

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
       appArchivePath: my-snapshot-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. 선택적으로, 복원할 애플리케이션의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.
   - **resourceFilter.resourceSelectionCriteria**: (필터링에 필요) 사용 `include or exclude` resourceMatchers에 정의된 리소스를 포함하거나 제외합니다. 다음 resourceMatchers 매개변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
     - **resourceFilter.resourceMatchers**: resourceMatcher 객체의 배열입니다. 이 배열에 여러 요소를 정의하는 경우 OR 연산으로 일치하고, 각 요소 내부의 필드(그룹, 종류, 버전)는 AND 연산으로 일치합니다.
       - **resourceMatchers[].group**: (선택 사항) 필터링할 리소스 그룹입니다.
       - **resourceMatchers[].kind**: (선택 사항) 필터링할 리소스의 종류입니다.
       - **resourceMatchers[].version**: (선택 사항) 필터링할 리소스의 버전입니다.

- **resourceMatchers[].names**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- **resourceMatchers[].namespaces**: (선택 사항) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers[].labelSelectors**: (선택 사항) 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "쿠버네티스 문서" . 예를 들어: `"trident.netapp.io/os=linux"` .

  예를 들어:

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 다음을 채운 후 `trident-protect-snapshot-restore-cr.yaml` 올바른 값으로 파일을 만들고 CR을 적용하세요.

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

**CLI**를 사용하여 스냅샷 복원

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔서 스냅샷을 다른 네임스페이스로 복원합니다.
   - 그만큼 `snapshot` 인수는 형식에 네임스페이스와 스냅샷 이름을 사용합니다. `<namespace>/<name>` .
   - 그만큼 `namespace-mapping` 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 형식에 맞는 올바른 대상 네임스페이스에 매핑합니다. `source1:dest1,source2:dest2` .

     예를 들어:

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

## Trident Protect 실행 후크 관리

실행 후크는 관리되는 앱의 데이터 보호 작업과 함께 실행되도록 구성할 수 있는 사용자 지정 작업입니다. 예를 들어, 데이터베이스 앱이 있는 경우 실행 후크를 사용하여 스냅샷 전에 모든 데이터베이스 트랜잭션을 일시 중지하고 스냅샷이 완료된 후 트랜잭션을 다시 시작할 수 있습니다. 이를 통해 애플리케이션과 일관된 스냅샷이 보장됩니다.

### 실행 후크의 유형

Trident Protect는 실행 가능 시기에 따라 다음 유형의 실행 후크를 지원합니다.

- 사전 스냅샷
- 스냅샷 이후
- 사전 백업
- 백업 후
- 복원 후
- 장애 조치 후

### 실행 순서

데이터 보호 작업이 실행되면 실행 후크 이벤트는 다음 순서로 발생합니다.

1. 적용 가능한 사용자 정의 사전 작업 실행 후크는 해당 컨테이너에서 실행됩니다. 필요한 만큼 사용자 정의 사전 작업 후크를 만들고 실행할 수 있지만, 작업 전에 이러한 후크를 실행하는 순서는 보장되지 않으며 구성할 수도 없습니다.
2. 해당되는 경우 파일 시스템이 정지됩니다. "Trident Protect를 사용하여 파일 시스템 동결을 구성하는 방법에 대해 자세히 알아보세요.".
3. 데이터 보호 작업이 수행됩니다.
4. 해당되는 경우 동결된 파일 시스템이 동결 해제됩니다.
5. 적용 가능한 모든 사용자 정의 사후 작업 실행 후크는 해당 컨테이너에서 실행됩니다. 필요한 만큼 사용자 정의 사후 작업 후크를 만들고 실행할 수 있지만, 작업 후 이러한 후크의 실행 순서는 보장되지 않으며 구성할 수도 없습니다.

동일한 유형의 실행 후크를 여러 개 생성하는 경우(예: 사전 스냅샷), 해당 후크의 실행 순서는 보장되지 않습니다. 하지만 서로 다른 유형의 후크 실행 순서는 보장됩니다. 예를 들어, 다양한 유형의 후크를 모두 포함하는 구성을 실행하는 순서는 다음과 같습니다.

1. 스냅샷 전 후크 실행됨
2. 스냅샷 후 후크 실행됨
3. 사전 백업 후크 실행됨

4. 백업 후 후크 실행됨

이전 주문 예시는 기존 스냅샷을 사용하지 않는 백업을 실행할 때만 적용됩니다.

프로덕션 환경에서 실행 후크 스크립트를 활성화하기 전에 항상 테스트해야 합니다. 'kubectl exec' 명령을 사용하면 편리하게 스크립트를 테스트할 수 있습니다. 프로덕션 환경에서 실행 후크를 활성화한 후 결과 스냅샷과 백업을 테스트하여 일관성이 있는지 확인하세요. 앱을 임시 네임스페이스에 복제하고 스냅샷이나 백업을 복원한 다음 앱을 테스트하면 됩니다.

스냅샷 이전 실행 후크가 Kubernetes 리소스를 추가, 변경 또는 제거하는 경우 해당 변경 사항은 스냅샷이나 백업 및 이후의 모든 복원 작업에 포함됩니다.

사용자 정의 실행 후크에 대한 중요 참고 사항

앱의 실행 후크를 계획할 때 다음 사항을 고려하세요.

- 실행 후크는 스크립트를 사용하여 작업을 수행해야 합니다. 여러 개의 실행 후크가 동일한 스크립트를 참조할 수 있습니다.
- Trident Protect에서는 실행 후크가 사용하는 스크립트가 실행 가능한 셸 스크립트 형식으로 작성되어야 합니다.
- 스크립트 크기는 96KB로 제한됩니다.
- Trident Protect는 실행 후크 설정과 일치 기준을 사용하여 스냅샷, 백업 또는 복원 작업에 적용할 수 있는 후크를 결정합니다.

실행 후크는 종종 실행 중인 애플리케이션의 기능을 감소시키거나 완전히 비활성화하기 때문에 사용자 정의 실행 후크가 실행되는 데 걸리는 시간을 최소화하려고 항상 노력해야 합니다. 연관된 실행 후크로 백업 또는 스냅샷 작업을 시작한 후 취소하더라도 백업 또는 스냅샷 작업이 이미 시작된 경우 후크는 계속 실행될 수 있습니다. 즉, 백업 후 실행 후크에 사용된 로직은 백업이 완료되었다고 가정할 수 없습니다.

실행 후크 필터

애플리케이션에 대한 실행 후크를 추가하거나 편집할 때 실행 후크에 필터를 추가하여 후크가 일치할 컨테이너를 관리할 수 있습니다. 필터는 모든 컨테이너에서 동일한 컨테이너 이미지를 사용하지만 각 이미지를 다른 목적으로 사용할 수 있는 애플리케이션(예: Elasticsearch)에 유용합니다. 필터를 사용하면 실행 후크가 일부 컨테이너에서만 실행되는 시나리오를 만들 수 있지만 반드시 모든 컨테이너에서 실행되는 것은 아닙니다. 단일 실행 후크에 대해 여러 필터를 만드는 경우 논리적 AND 연산자를 사용하여 결합됩니다. 실행 후크당 최대 10개의 활성 필터를 가질 수 있습니다.

실행 후크에 추가하는 각 필터는 정규 표현식을 사용하여 클러스터의 컨테이너와 일치시킵니다. 후크가 컨테이너와 일치하면 후크는 해당 컨테이너에서 연관된 스크립트를 실행합니다. 필터에 대한 정규 표현식은 RE2(Regular Expression 2) 구문을 사용하는데, 이 구문은 일치 항목 목록에서 컨테이너를 제외하는 필터를 만드는 것을 지원하지 않습니다. Trident Protect가 실행 후크 필터의 정규 표현식에 대해 지원하는 구문에 대한 정보는 다음을 참조하세요. "정규 표현식 2(RE2) 구문 지원" .

복원 또는 복제 작업 후에 실행되는 실행 후크에 네임스페이스 필터를 추가하고 복원 또는 복제 소스와 대상이 다른 네임스페이스에 있는 경우, 네임스페이스 필터는 대상 네임스페이스에만 적용됩니다.

## 실행 후크 예제

방문하세요 "NetApp Verda GitHub 프로젝트" Apache Cassandra 및 Elasticsearch와 같은 인기 있는 앱에 대한 실제 실행 후크를 다운로드합니다. 또한 사용자 정의 실행 후크를 구성하는 데 필요한 예제를 보고 아이디어를 얻을 수 있습니다.

## 실행 후크 만들기

예 **1.** 을 사용하여 앱에 대한 사용자 정의 실행 후크를 만들 수 있습니다**.** 실행 후크를 생성하려면 소유자**,** 관리자 또는 멤버 권한이 필요합니다**.**

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-hook.yaml` .
2. Trident Protect 환경 및 클러스터 구성에 맞게 다음 속성을 구성하세요.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.applicationRef**: (필수) 실행 후크를 실행할 애플리케이션의 Kubernetes 이름입니다.
   - **spec.stage**: (필수) 실행 후크가 동작 중 어느 단계에서 실행되어야 하는지를 나타내는 문자열입니다. 가능한 값:
     - 사전
     - 우편
   - **spec.action**: (필수) 지정된 실행 후크 필터가 모두 일치한다고 가정할 때 실행 후크가 수행할 작업을 나타내는 문자열입니다. 가능한 값:
     - 스냅샷
     - 지원
     - 복원하다
     - 장애 조치
   - **spec.enabled**: (선택 사항) 이 실행 후크가 활성화되어 있는지 비활성화되어 있는지를 나타냅니다. 지정하지 않으면 기본값은 true입니다.
   - **spec.hookSource**: (필수) base64로 인코딩된 후크 스크립트가 포함된 문자열입니다.
   - **spec.timeout**: (선택 사항) 실행 후크가 실행될 수 있는 시간을 분 단위로 정의하는 숫자입니다. 최소값은 1분이며, 지정하지 않으면 기본값은 25분입니다.
   - **spec.arguments**: (선택 사항) 실행 후크에 지정할 수 있는 인수의 YAML 목록입니다.
   - **spec.matchingCriteria**: (선택 사항) 조건 키 값 쌍의 선택적 목록으로, 각 쌍은 실행 후크 필터를 구성합니다. 실행 후크당 최대 10개의 필터를 추가할 수 있습니다.
   - **spec.matchingCriteria.type**: (선택 사항) 실행 후크 필터 유형을 식별하는 문자열입니다. 가능한 값:
     - 컨테이너이미지
     - 컨테이너 이름
     - 포드이름
     - 포드레이블
     - 네임스페이스 이름
   - **spec.matchingCriteria.value**: (선택 사항) 실행 후크 필터 값을 식별하는 문자열 또는 정규 표현식입니다.

     YAML 예시:

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. CR 파일에 올바른 값을 채운 후 CR을 적용합니다.

```
kubectl apply -f trident-protect-hook.yaml
```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 환경의 정보로 바꿔서 실행 후크를 만듭니다. 예를 들어:

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

수동으로 실행 후크 실행

테스트 목적으로 실행 후크를 수동으로 실행할 수 있으며, 실패 후 후크를 수동으로 다시 실행해야 하는 경우에도 사용할 수 있습니다. 실행 후크를 수동으로 실행하려면 소유자, 관리자 또는 멤버 권한이 필요합니다.

실행 후크를 수동으로 실행하는 것은 두 가지 기본 단계로 구성됩니다.

1. 리소스를 수집하고 백업을 생성하여 후크가 실행될 위치를 결정하는 리소스 백업을 생성합니다.
2. 백업에 대해 실행 후크를 실행합니다.

**1**단계: 리소스 백업 만들기

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-resource-backup.yaml` .
2. Trident Protect 환경 및 클러스터 구성에 맞게 다음 속성을 구성하세요.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.applicationRef**: (필수) 리소스 백업을 생성할 애플리케이션의 Kubernetes 이름입니다.
   - **spec.appVaultRef**: (필수) 백업 내용이 저장된 AppVault의 이름입니다.
   - **spec.appArchivePath**: 백업 내용이 저장되는 AppVault 내부 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     YAML 예시:

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: ResourceBackup
     metadata:
       name: example-resource-backup
     spec:
       applicationRef: my-app-name
       appVaultRef: my-appvault-name
       appArchivePath: example-resource-backup
     ```

3. CR 파일에 올바른 값을 채운 후 CR을 적용합니다.

   ```
   kubectl apply -f trident-protect-resource-backup.yaml
   ```

**CLI**를 사용하세요

단계

1. 괄호 안의 값을 사용자 환경의 정보로 바꿔 백업을 만듭니다. 예를 들어:

   ```
   tridentctl protect create resourcebackup <my_backup_name> --app
   <my_app_name> --appvault <my_appvault_name> -n
   <my_app_namespace> --app-archive-path <app_archive_path>
   ```

2. 백업 상태를 확인합니다. 작업이 완료될 때까지 이 예제 명령을 반복해서 사용할 수 있습니다.

```
tridentctl protect get resourcebackup -n <my_app_namespace>
<my_backup_name>
```

3. 백업이 성공했는지 확인하세요.

```
kubectl describe resourcebackup <my_backup_name>
```

**2**단계: 실행 후크 실행

**CR**을 사용하세요

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다. `trident-protect-hook-run.yaml` .
2. Trident Protect 환경 및 클러스터 구성에 맞게 다음 속성을 구성하세요.
   - **metadata.name**: (필수) 이 사용자 지정 리소스의 이름입니다. 사용자 환경에 맞게 고유하고 적절한 이름을 선택하세요.
   - **spec.applicationRef**: (필수) 이 값이 1단계에서 만든 ResourceBackup CR의 애플리케이션 이름과 일치하는지 확인하세요.
   - **spec.appVaultRef**: (필수) 이 값이 1단계에서 생성한 ResourceBackup CR의 appVaultRef와 일치하는지 확인하세요.
   - **spec.appArchivePath**: 이 값이 1단계에서 만든 ResourceBackup CR의 appArchivePath와 일치하는지 확인합니다.

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.action**: (필수) 지정된 실행 후크 필터가 모두 일치한다고 가정할 때 실행 후크가 수행할 작업을 나타내는 문자열입니다. 가능한 값:
     - 스냅샷
     - 지원
     - 복원하다
     - 장애 조치
   - **spec.stage**: (필수) 실행 후크가 동작 중 어느 단계에서 실행되어야 하는지를 나타내는 문자열입니다. 이 후크 실행은 다른 단계에서 후크를 실행하지 않습니다. 가능한 값:
     - 사전
     - 우편

     YAML 예시:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

3. CR 파일에 올바른 값을 채운 후 CR을 적용합니다.

```
kubectl apply -f trident-protect-hook-run.yaml
```

**CLI**를 사용하세요

단계

1. 수동 실행 후크 실행 요청을 만듭니다.

```
tridentctl protect create exechooksrun <my_exec_hook_run_name>
-n <my_app_namespace> --action snapshot --stage <pre_or_post>
--app <my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. 실행 후크 실행 상태를 확인합니다. 작업이 완료될 때까지 이 명령을 반복해서 실행할 수 있습니다.

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. 최종 세부 정보와 상태를 확인하려면 exechooksrun 객체를 설명합니다.

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

# Trident Protect 제거

평가판에서 정식 버전으로 업그레이드하는 경우 Trident Protect 구성 요소를 제거해야 할 수도 있습니다.

Trident Protect를 제거하려면 다음 단계를 수행하세요.

단계

1. Trident Protect CR 파일을 제거합니다.

   이 단계는 25.06 이상 버전에서는 필요하지 않습니다.

   ```
   helm uninstall -n trident-protect trident-protect-crds
   ```

2. Trident Protect 제거:

   ```
   helm uninstall -n trident-protect trident-protect
   ```

3. Trident Protect 네임스페이스를 제거합니다.

   ```
   kubectl delete ns trident-protect
   ```

저작권 정보

Copyright © 2026 NetApp, Inc. All Rights Reserved. 미국에서 인쇄됨 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 라이센스와 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 라이센스도 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 라이센스를 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 라이센스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

NETAPP, NETAPP 로고 및 http://www.netapp.com/TM에 나열된 마크는 NetApp, Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.