



Trident Protect로 애플리케이션을 보호하세요

Trident

NetApp
March 05, 2026

목차

Trident Protect로 애플리케이션을 보호하세요	1
Trident Protect에 대해 알아보세요	1
다음 단계	1
Trident Protect 설치	1
Trident 프로텍트 요구 사항	1
Trident Protect 설치 및 구성	4
Trident Protect CLI 플러그인을 설치하세요	9
Trident Protect 관리	13
Trident Protect 권한 및 액세스 제어 관리	13
Trident Protect 지원 번들 생성	19
Trident 프로텍트 업그레이드	21
애플리케이션을 관리하고 보호합니다	21
Trident Protect AppVault 객체를 사용하여 버킷을 관리합니다	21
Trident Protect를 사용하여 관리를 위한 애플리케이션 정의	29
Trident Protect를 사용하여 애플리케이션 보호	31
Trident Protect를 사용하여 애플리케이션 복원	39
NetApp SnapMirror 및 Trident Protect를 사용하여 애플리케이션 복제	55
Trident Protect를 사용하여 애플리케이션 마이그레이션	67
Trident Protect 실행 후크 관리	71
Trident Protect 제거	76

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.

다음 단계

Trident Protect를 설치하기 전에 요구 사항에 대해 알아볼 수 있습니다.

- ["Trident 프로젝트 요구 사항"](#)

Trident Protect 설치

Trident 프로젝트 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.

- Amazon Elastic Kubernetes Service(EKS)
- Google Kubernetes Engine(GKE)
- Microsoft Azure Kubernetes Service(AKS)
- Red Hat OpenShift
- 수세 목장
- VMware Tanzu 포트폴리오
- 업스트림 Kubernetes



Trident Protect를 설치하는 클러스터가 실행 중인 스냅샷 컨트롤러와 관련 CRD로 구성되어 있는지 확인하세요. 스냅샷 컨트롤러를 설치하려면 다음을 참조하세요. ["참조하십시오"](#).

Trident Protect 스토리지 백엔드 호환성

Trident Protect는 다음과 같은 스토리지 백엔드를 지원합니다.

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

스토리지 백엔드가 다음 요구 사항을 충족하는지 확인합니다.

- 클러스터에 연결된 NetApp 스토리지가 Astra Trident 24.02 이상을 사용하는지 확인합니다(Trident 24.10 권장).
 - Astra Trident가 24.06.1 이전 버전이고 NetApp SnapMirror 재해 복구 기능을 사용하려면 Astra Control Provisioner를 수동으로 활성화해야 합니다.
- 최신 Astra Control Provisioner(Astra Trident 24.06.1 기준으로 기본적으로 설치 및 활성화됨)가 있는지 확인합니다.
- NetApp ONTAP 스토리지 백엔드가 있는지 확인합니다.
- 백업을 저장할 오브젝트 스토리지 버킷을 구성했는지 확인합니다.
- 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 경제 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
```

각 NAS 경제 볼륨에 대해 다음 명령을 실행하여 스냅샷 디렉토리를 설정할 수 있으며, 변경할 볼륨의 UUID로 바꿀 수 <volume-UUID> 있습니다.

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



Trident 백엔드 구성 옵션을 로 true 설정하면 기본적으로 새 볼륨에 대해 스냅샷 디렉토리를 설정할 수 snapshotDir 있습니다. 기존 볼륨은 영향을 받지 않습니다.

KubeVirt VM으로 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.

모든 Trident Protect 버전의 경우 OpenShift 환경에서 자동 동결 기능을 활성화하거나 비활성화하려면 애플리케이션 네임스페이스에 특권 권한을 부여해야 할 수 있습니다. 예를 들어:



```
oc adm policy add-scc-to-user privileged -z default -n
<application-namespace>
```

Trident 프로젝트 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 해제합니다. 필요에 따라 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 복제에 대한 요구 사항

NetApp SnapMirror 는 다음 ONTAP 솔루션의 Trident Protect와 함께 사용할 수 있습니다.

- NetApp ASA
- NetApp AFF
- NetApp FAS
- NetApp ONTAP Select를 참조하십시오
- NetApp Cloud Volumes ONTAP를 참조하십시오
- NetApp ONTAP용 Amazon FSx

SnapMirror 복제를 위한 ONTAP 클러스터 요구 사항

SnapMirror 복제를 사용하려는 경우 ONTAP 클러스터가 다음 요구 사항을 충족하는지 확인하십시오.

- * Astra Control Provisioner 또는 Trident*: Astra Control Provisioner 또는 Trident ONTAP 백엔드로 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
 - ontap-san
- * 라이선스 *: 소스 및 대상 ONTAP 클러스터 모두에서 데이터 보호 번들을 사용하는 ONTAP SnapMirror 비동기 라이선스를 활성화해야 합니다. 자세한 내용은 을 "[ONTAP의 SnapMirror 라이선스 개요](#)" 참조하십시오.

SnapMirror 복제에 대한 피어링 고려 사항

스토리지 백엔드 피어링을 사용하려는 경우 환경이 다음 요구 사항을 충족하는지 확인하십시오.

- * 클러스터 및 SVM *: ONTAP 스토리지 백엔드를 피어링해야 합니다. 자세한 내용은 을 "[클러스터 및 SVM 피어링 개요](#)" 참조하십시오.



두 ONTAP 클러스터 간의 복제 관계에 사용되는 SVM 이름이 고유한지 확인합니다.

- * Astra Control Provisioner 또는 Trident 및 SVM *: 피어링된 원격 SVM을 대상 클러스터의 Astra Control Provisioner 또는 Trident에서 사용할 수 있어야 합니다.
- 관리형 백엔드: 복제 관계를 생성하려면 Trident Protect에서 ONTAP 스토리지 백엔드를 추가하고 관리해야 합니다.
- TCP를 통한 NVMe: Trident Protect는 TCP 프로토콜을 통한 NVMe를 사용하는 스토리지 백엔드에 대한 NetApp SnapMirror 복제를 지원하지 않습니다.

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.

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.



기본적으로 Trident Protect는 클러스터 및 관리형 애플리케이션에 대한 로그, 메트릭, 토폴로지 정보를 포함하여 사용자가 열 수 있는 모든 NetApp 지원 사례에 도움이 되는 지원 정보를 수집합니다. Trident Protect는 이러한 지원 번들을 매일 NetApp 에 보냅니다. Trident Protect를 설치할 때 선택적으로 이 지원 번들 컬렉션을 비활성화할 수 있습니다. 수동으로 할 수 있습니다 "[지원 번들을 생성합니다](#)" 언제든지.

Trident Protect 설치

NetApp 에서 Trident Protect 설치

단계

1. Trident Helm 저장소 추가:

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

2. Trident Protect CRD를 설치하세요:

```
helm install trident-protect-crds netapp-trident-protect/trident-  
protect-crds --version 100.2410.1 --create-namespace --namespace  
trident-protect
```

3. 다음 명령 중 하나를 사용하여 Helm을 사용하여 Trident Protect를 설치합니다. 바꾸다 <name_of_cluster> 클러스터에 할당되고 클러스터의 백업과 스냅샷을 식별하는 데 사용되는 클러스터 이름이 포함됩니다.

- Trident Protect를 정상적으로 설치하세요:

```
helm install trident-protect netapp-trident-protect/trident-  
protect --set clusterName=<name_of_cluster> --version 100.2410.1  
--create-namespace --namespace trident-protect
```

- Trident Protect를 설치하고 예약된 일일 Trident Protect AutoSupport 지원 번들 업로드를 비활성화하세요.

```
helm install trident-protect netapp-trident-protect/trident-  
protect --set autoSupport.enabled=false --set  
clusterName=<name_of_cluster> --version 100.2410.1 --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 컴퓨터로 가져와서 태그를 업데이트한 다음 개인 레지스트리에 푸시합니다.

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exehook:24.10.1
netapp/resourcebackup:24.10.1
netapp/resourcerestore:24.10.1
netapp/resourcedelete:24.10.1
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

예를 들면 다음과 같습니다.

```
docker pull netapp/controller:24.10.1
```

```
docker tag netapp/controller:24.10.1 <private-registry-
url>/controller:24.10.1
```

```
docker push <private-registry-url>/controller:24.10.1
```

2. Trident Protect 시스템 네임스페이스를 만듭니다.

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

3. 레지스트리에 로그인합니다.

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

4. 개인 레지스트리 인증에 사용할 풀 암호를 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 저장소 추가:

```
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. Trident Protect CRD를 설치하세요:

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

8. 다음 명령 중 하나를 사용하여 Helm을 사용하여 Trident Protect를 설치합니다. 바꾸다 `<name_of_cluster>` 클러스터에 할당되고 클러스터의 백업과 스냅샷을 식별하는 데 사용되는 클러스터 이름이 포함됩니다.

- Trident Protect를 정상적으로 설치하세요:

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.1
--create-namespace --namespace trident-protect -f
protectValues.yaml
```

- Trident Protect를 설치하고 예약된 일일 Trident Protect AutoSupport 지원 번들 업로드를 비활성화하세요.

```
helm install trident-protect netapp-trident-protect/trident-protect --set autoSupport.enabled=false --set clusterName=<name_of_cluster> --version 100.2410.1 --create --namespace --namespace trident-protect -f protectValues.yaml
```

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

다음 예제 구성 파일은 사용 가능한 설정을 보여 주며 각 리소스 제한에 대한 기본 values를 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. 파일의 값을 적용합니다 resourceLimits.yaml.

```

helm upgrade trident-protect -n trident-protect -f <resourceLimits.yaml>
--reuse-values

```

Trident Protect CLI 플러그인을 설치하세요

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

Trident Protect CLI 플러그인을 설치하세요

명령줄 유틸리티를 사용하기 전에 클러스터에 액세스하는 데 사용하는 시스템에 설치해야 합니다. 컴퓨터에서 x64 또는 ARM CPU를 사용하는지 여부에 따라 다음 단계를 수행합니다.

Linux AMD64 CPU용 플러그인을 다운로드합니다

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-linux-amd64
```

Linux ARM64 CPU용 플러그인을 다운로드합니다

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-linux-arm64
```

Mac AMD64 CPU용 플러그인을 다운로드합니다

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-macos-amd64
```

Mac ARM64 CPU용 플러그인을 다운로드합니다

단계

1. Trident Protect CLI 플러그인을 다운로드하세요:

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-macos-arm64
```

1. 플러그인 바이너리에 대한 실행 권한 활성화:

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

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

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

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

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



PATH 변수의 위치에 플러그인을 복사하면 또는 `tridentctl protect` 임의의 위치에서 플러그인을 사용할 수 `tridentctl-protect` 있습니다.

Trident CLI 플러그인 도움말을 봅니다

내장 플러그인 도움말 기능을 사용하여 플러그인 기능에 대한 자세한 도움말을 얻을 수 있습니다.

단계

1. 도움말 기능을 사용하여 사용 지침을 봅니다.

```
tridentctl-protect help
```

명령 자동 완성 활성화

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

Bash 셸에 대해 자동 완성 기능을 활성화합니다

단계

1. 완료 스크립트 다운로드:

```
curl -L -O https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-completion.bash
```

2. 스크립트를 포함할 홈 디렉토리에 새 디렉토리를 만듭니다.

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

3. 다운로드한 스크립트를 디렉터리로 이동합니다 ~/.bash/completions.

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

4. 홈 디렉토리의 파일에 다음 줄을 추가합니다 ~/.bashrc.

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

Z 셸에 대한 자동 완성 기능을 활성화합니다

단계

1. 완료 스크립트 다운로드:

```
curl -L -O https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-completion.zsh
```

2. 스크립트를 포함할 홈 디렉토리에 새 디렉토리를 만듭니다.

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

3. 다운로드한 스크립트를 디렉터리로 이동합니다 ~/.zsh/completions.

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

4. 홈 디렉토리의 파일에 다음 줄을 추가합니다 ~/.zprofile.

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

결과

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

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 네임스페이스의 리소스에 액세스할 수 `trident-protect` 있으며 다른 모든 네임스페이스의 리소스에 액세스할 수도 있습니다. 리소스 및 응용 프로그램에 대한 액세스를 제어하려면 추가 네임스페이스를 만들고 해당 네임스페이스에 리소스 및 응용 프로그램을 추가해야 합니다.

사용자는 기본 네임스페이스에 응용 프로그램 데이터 관리 CRS를 만들 수 `trident-protect` 없습니다. 응용 프로그램 네임스페이스에 응용 프로그램 데이터 관리 CRS를 만들어야 합니다(모범 사례로서 연결된 응용 프로그램과 동일한 네임스페이스에 응용 프로그램 데이터 관리 CRS를 만듭니다).

다음은 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.
- *** AutoSupport기술에 따라 로그 수집 일정을 관리할 수 있습니다**

가장 좋은 방법은 RBAC를 사용하여 권한이 있는 객체에 대한 액세스를 관리자에게 제한하는 것입니다.

RBAC가 리소스 및 네임스페이스에 대한 액세스를 규제하는 방법에 대한 자세한 내용은 ["Kubernetes RBAC 설명서"](#)참조하십시오.

서비스 계정에 대한 자세한 내용은 ["Kubernetes 서비스 계정 설명서"](#)참조하십시오.

예: 두 사용자 그룹에 대한 액세스를 관리합니다

예를 들어, 조직에는 클러스터 관리자, 엔지니어링 사용자 그룹 및 마케팅 사용자 그룹이 있습니다. 클러스터 관리자는 엔지니어링 그룹과 마케팅 그룹이 각각 해당 네임스페이스에 할당된 리소스에만 액세스할 수 있는 환경을 만들기 위해 다음 작업을 완료합니다.

1단계: 각 그룹의 리소스를 포함할 네임스페이스를 만듭니다

네임스페이스를 만들면 리소스를 논리적으로 분리하고 해당 리소스에 액세스할 수 있는 사용자를 보다 효율적으로 제어할 수 있습니다.

단계

1. 엔지니어링 그룹의 네임스페이스를 만듭니다.

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

2. 마케팅 그룹의 네임스페이스를 만듭니다.

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

2단계: 각 네임스페이스의 리소스와 상호 작용할 새 서비스 계정을 만듭니다

새로 만드는 각 네임스페이스에는 기본 서비스 계정이 함께 제공되지만, 나중에 필요한 경우 그룹 간에 Privileges를 추가로 나눌 수 있도록 각 사용자 그룹에 대한 서비스 계정을 만들어야 합니다.

단계

1. 엔지니어링 그룹에 대한 서비스 계정을 생성합니다.

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

2. 마케팅 그룹의 서비스 계정 만들기:

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

3단계: 각 새 서비스 계정에 대한 암호를 만듭니다

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

단계

1. 엔지니어링 서비스 계정에 대한 암호를 생성합니다.

```
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 암호 만들기:

```
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
```

4단계: **RoleBinding** 개체를 만들어 **ClusterRole** 개체를 각 새 서비스 계정에 바인딩합니다

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

단계

1. ClusterRole을 엔지니어링 서비스 계정에 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 연결:

```

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

```

5단계: 권한을 테스트합니다

권한이 올바른지 테스트합니다.

단계

1. 엔지니어링 사용자가 엔지니어링 리소스에 액세스할 수 있는지 확인합니다.

```

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 확인합니다.

```

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

```

6단계: AppVault 객체에 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. AppVault에 대한 사용자 액세스 권한을 부여하는 AppVault 및 암호 조합 YAML 파일을 만들고 적용합니다. 예를 들어, 다음 CR은 사용자에게 AppVault에 대한 액세스 권한을 eng-user 부여합니다.

```

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

```

- 클러스터 관리자가 네임스페이스의 특정 리소스에 대한 액세스 권한을 부여할 수 있도록 역할 CR을 생성하고 적용합니다. 예를 들면 다음과 같습니다.

```
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 만들고 적용하여 권한을 사용자 eng-user에 바인딩합니다. 예를 들면 다음과 같습니다.

```
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 확인합니다.

a. 모든 네임스페이스에 대한 AppVault 개체 정보를 검색하려고 합니다.

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

다음과 유사한 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 테스트해 봅니다.

```
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 유사한 출력이 표시됩니다.

```
yes
```

결과

AppVault 권한을 부여한 사용자는 응용 프로그램 데이터 관리 작업에 승인된 AppVault 개체를 사용할 수 있어야 하며, 할당된 네임스페이스 외부의 리소스에 액세스하거나 액세스 권한이 없는 새 리소스를 생성할 수 없습니다.

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.

CR을 사용하여 지원 번들을 만듭니다

단계

1. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: `trident-protect-support-bundle.yaml`).
2. 다음 특성을 구성합니다.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** spec.triggerType *:** *(required)* 지원 번들이 즉시 생성되는지 또는 예약되는지 여부를 결정합니다. 예약된 번들 생성은 UTC 오전 12시에 수행됩니다. 가능한 값:
 - 예약됨
 - 수동
 - **spec.uploadEnabled:** *(Optional)* 지원 번들이 생성된 후 NetApp 지원 사이트에 업로드되어야 하는지 여부를 제어합니다. 지정하지 않으면 기본값으로 `false` 설정됩니다. 가능한 값:
 - 참
 - FALSE(기본값)
 - **spec.dataWindowStart:** *(Optional)* 지원 번들에 포함된 데이터의 창이 시작되는 날짜와 시간을 지정하는 RFC 3339 형식의 날짜 문자열입니다. 지정하지 않을 경우 기본값은 24시간 이전입니다. 지정할 수 있는 가장 빠른 기간 날짜는 7일 이전입니다.

YAML 예:

```
---
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. 파일을 올바른 값으로 채운 후 `astra-support-bundle.yaml` CR:

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

CLI를 사용하여 지원 번들을 생성합니다

단계

1. 괄호 안의 값을 사용자 환경의 정보로 대체하여 지원 번들을 만듭니다. 은 `trigger-type` 번들이 즉시 생성되는지 아니면 생성 시간이 스케줄에 따라 결정되는지, 또는 `Scheduled` 이 될 수 있는지 결정합니다 `Manual`. 기본 설정은 `'Manual'`입니다.

예를 들면 다음과 같습니다.

```
tridentctl-protect create autosupportbundle <my_bundle_name>  
--trigger-type <trigger_type>
```

Trident 프로텍트 업그레이드

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

Trident Protect를 업그레이드하려면 다음 단계를 수행하세요.

단계

1. Trident Helm 리포지토리를 업데이트합니다.

```
helm repo update
```

2. Trident Protect CRD 업그레이드:

```
helm upgrade trident-protect-crds netapp-trident-protect/trident-  
protect-crds --version 100.2410.1 --namespace trident-protect
```

3. Trident 프로텍트 업그레이드:

```
helm upgrade trident-protect netapp-trident-protect/trident-protect  
--version 100.2410.1 --namespace trident-protect
```

애플리케이션을 관리하고 보호합니다

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.

키 생성 및 **AppVault** 정의 예

AppVault CR을 정의할 때 공급자가 호스팅하는 리소스에 액세스하려면 자격 증명을 포함해야 합니다. 자격 증명에 대한 키를 생성하는 방법은 공급자에 따라 다릅니다. 다음은 여러 공급자에 대한 명령줄 키 생성 예제와 각 공급자에

대한 AppVault 정의 예입니다.

키 생성 예

다음 예제를 사용하여 각 클라우드 공급자의 자격 증명에 대한 키를 생성할 수 있습니다.

Google 클라우드

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

Amazon S3(AWS)

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

Microsoft Azure를 참조하십시오

```
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=<generic-s3-  
trident-protect-src-bucket-secret> -n trident-protect
```

StorageGRID 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
```

AppVault CR 예

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

Google 클라우드

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Amazon S3(AWS)

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Microsoft Azure를 참조하십시오

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  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-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

StorageGRID S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-
  971f-ac4a83621922
  namespace: trident-protect
spec:
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Trident Protect CLI를 사용한 AppVault 생성 예

다음 CLI 명령 예제를 사용하여 각 공급자에 대해 AppVault CRS를 만들 수 있습니다.

Google 클라우드

```
tridentctl-protect create vault GCP my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3(AWS)

```
tridentctl-protect create vault AWS <vault-name> --bucket <bucket-name>
--secret <secret-name> --endpoint <s3-endpoint>
```

Microsoft Azure를 참조하십시오

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

일반 S3

```
tridentctl-protect create vault GenericS3 <vault-name> --bucket
<bucket-name> --secret <secret-name> --endpoint <s3-endpoint>
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> --bucket <bucket-
name> --secret <secret-name> --endpoint <s3-endpoint>
```

StorageGRID S3

```
tridentctl-protect create vault StorageGridS3 s3vault --bucket <bucket-
name> --secret <secret-name> --endpoint <s3-endpoint>
```

AppVault 브라우저를 사용하여 AppVault 정보를 봅니다

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

단계

1. AppVault 개체의 내용을 봅니다.

```
tridentctl-protect get appvaultcontent gcp-vault --show-resources all
```

◦ 출력 예 *:

```

+-----+-----+-----+-----+
+-----+
| 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 리소스의 AppVaultPath를 보려면 플래그를 `--show-paths` 사용합니다.

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

AppVault를 제거합니다

언제든지 AppVault 개체를 제거할 수 있습니다.



AppVault 개체를 삭제하기 전에 AppVault CR에서 키를 제거하지 `finalizers` 마십시오. 이렇게 하면 AppVault 버킷의 잔여 데이터와 클러스터의 분리된 리소스가 생성될 수 있습니다.

시작하기 전에

연결된 버킷에 저장된 모든 스냅샷과 백업을 삭제했는지 확인합니다.

Kubernetes CLI를 사용하여 AppVault를 제거합니다

1. AppVault 개체를 제거하고 `appvault_name` 제거할 AppVault 개체의 이름으로 바꿉니다.

```
kubectl delete appvault <appvault_name> -n trident-protect
```

Trident Protect CLI를 사용하여 AppVault 제거

1. AppVault 개체를 제거하고 `appvault_name` 제거할 AppVault 개체의 이름으로 바꿉니다.

```
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 CR을 작성합니다

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 지정 리소스](#)."

응용 프로그램을 정의합니다

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

CR을 사용하여 응용 프로그램을 추가합니다

단계

1. 대상 응용 프로그램 CR 파일을 만듭니다.

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

b. 다음 특성을 구성합니다.

- **metadata.name:** (*required*) 응용 프로그램 사용자 정의 리소스의 이름입니다. 보호 작업에 필요한 다른 CR 파일이 이 값을 참조하기 때문에 선택하는 이름을 확인합니다.
- **spec.includedNamespaces:** (*required*) 네임스페이스 레이블이나 네임스페이스 이름을 사용하여 응용 프로그램 리소스가 있는 네임스페이스를 지정합니다. 응용 프로그램 네임스페이스는 이 목록의 일부여야 합니다.

YAML 예:

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

2. 환경에 맞게 응용 프로그램 CR을 만든 후 CR을 적용합니다. 예를 들면 다음과 같습니다.

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

CLI를 사용하여 애플리케이션을 추가합니다

단계

1. 응용 프로그램 정의를 만들고 적용하여 괄호 안의 값을 사용자 환경의 정보로 바꿉니다. 다음 예제에 표시된 인수를 사용하여 쉼표로 구분된 목록을 사용하여 응용 프로그램 정의에 네임스페이스와 리소스를 포함할 수 있습니다.

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

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."](#)

필요 시 스냅샷을 생성합니다

언제든지 주문형 스냅샷을 생성할 수 있습니다.

CR을 사용하여 스냅샷을 생성합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-snapshot-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** (*required*) 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - * **spec.applicationRef** *: 스냅샷을 생성할 애플리케이션의 Kubernetes 이름입니다.
 - * **spec.appVaultRef** *: (*required*) 스냅샷 내용(메타데이터)을 저장할 AppVault의 이름입니다.
 - **spec.reclaimPolicy:** (*Optional*) 스냅샷 CR이 삭제될 때 스냅샷의 AppArchive에 발생하는 동작을 정의합니다. 즉, 로 설정된 경우에도 Retain 스냅샷이 삭제됩니다. 유효한 옵션:
 - Retain (기본값)
 - 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. 파일을 올바른 값으로 채운 후 trident-protect-snapshot-cr.yaml CR:

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

CLI를 사용하여 스냅샷을 생성합니다

단계

1. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 스냅샷을 생성합니다. 예를 들면 다음과 같습니다.

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

필요 시 백업을 생성합니다

언제든지 앱을 백업할 수 있습니다.

CR을 사용하여 백업을 생성합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-backup-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** spec.applicationRef *:** *(required)* 백업할 응용 프로그램의 Kubernetes 이름입니다.
 - *** spec.appVaultRef *:** *(required)* 백업 내용을 저장할 AppVault의 이름입니다.
 - *** spec.datamover *:** *(Optional)* 백업 작업에 사용할 백업 도구를 나타내는 문자열입니다. 가능한 값 (대소문자 구분):
 - Restic
 - Kopia (기본값)
 - **spec.reclaimPolicy:** *(Optional)* 클레임에서 해제될 때 백업에 어떤 일이 발생하는지를 정의합니다. 가능한 값:
 - Delete
 - Retain (기본값)
 - *** Spec.snapshotRef *:** *(Optional)*: 백업 소스로 사용할 스냅샷의 이름입니다. 제공하지 않으면 임시 스냅샷이 생성되고 백업됩니다.

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

3. 파일을 올바른 값으로 채운 후 `trident-protect-backup-cr.yaml` CR:

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

CLI를 사용하여 백업을 생성합니다

단계

1. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 백업을 만듭니다. 예를 들면 다음과 같습니다.

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

데이터 보호 스케줄을 생성합니다

보호 정책은 정의된 일정에 따라 스냅샷, 백업 또는 둘 다를 생성하여 앱을 보호합니다. 시간별, 일별, 주별 및 월별 스냅샷과 백업을 생성하도록 선택할 수 있으며, 보존할 복제본 수를 지정할 수 있습니다.

CR을 사용하여 일정을 생성합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-schedule-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** (*required*) 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - * **spec.datamover *:** (*Optional*) 백업 작업에 사용할 백업 도구를 나타내는 문자열입니다. 가능한 값 (대소문자 구분):
 - Restic
 - Kopia (기본값)
 - * **spec.applicationRef *:** 백업할 응용 프로그램의 Kubernetes 이름입니다.
 - * **spec.appVaultRef *:** (*required*) 백업 내용을 저장할 AppVault의 이름입니다.
 - * **spec.backupRetention *:** 보존할 백업 수입니다. 0은 백업을 생성하지 않아야 함을 나타냅니다.
 - * **spec.snapshotRetention *:** 보존할 스냅샷 수입니다. 0은 스냅샷을 생성하지 않아야 함을 나타냅니다.
 - **spec.granularity:** 일정이 실행되는 빈도. 가능한 값과 필수 관련 필드:
 - hourly (지정 필요 spec.minute)
 - daily (및 을 지정해야 함 spec.minute spec.hour)
 - weekly (, 및 을 spec.dayOfWeek 지정해야 함 spec.minute, spec.hour)
 - monthly (, 및 을 spec.dayOfMonth 지정해야 함 spec.minute, spec.hour)
 - * **spec.dayOfMonth *:** (*Optional*) 스케줄을 실행할 요일(1-31). 세분화가 로 설정된 경우 이 필드는 `monthly` 필수입니다.
 - **spec.dayOfWeek:** (*Optional*) 일정이 실행되는 요일(0-7)입니다. 0 또는 7의 값은 일요일을 나타냅니다. 세분화가 로 설정된 경우 이 필드는 `weekly` 필수입니다.
 - * **spec.hour *:** (*Optional*) 스케줄을 실행할 시간(0-23)입니다. 세분화가, 또는 로 설정된 경우 이 필드는 daily weekly `monthly` 필수입니다.
 - * **spec.minute *:** (*Optional*) 스케줄을 실행할 분(0-59)입니다. 세분화가, , 또는 로 설정된 경우 이 필드가 hourly daily weekly `monthly` 필요합니다.

```

---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"

```

3. 파일을 올바른 값으로 채운 후 trident-protect-schedule-cr.yaml CR:

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

CLI를 사용하여 일정을 생성합니다

단계

1. 괄호 안의 값을 사용자 환경의 정보로 대체하여 보호 스케줄을 생성합니다. 예를 들면 다음과 같습니다.



을 사용하여 tridentctl-protect create schedule --help 이 명령에 대한 자세한 도움말 정보를 볼 수 있습니다.

```

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>

```

스냅샷을 삭제합니다

더 이상 필요하지 않은 예약된 스냅샷 또는 주문형 스냅샷을 삭제합니다.

단계

1. 스냅샷과 연결된 스냅샷 CR을 제거합니다.

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

백업을 삭제합니다

더 이상 필요하지 않은 예약된 백업 또는 필요 시 백업을 삭제합니다.

단계

1. 백업과 연결된 백업 CR을 제거합니다.

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

백업 작업의 상태를 확인합니다

명령줄을 사용하여 진행 중이거나, 완료되었거나, 실패한 백업 작업의 상태를 확인할 수 있습니다.

단계

1. 다음 명령을 사용하여 백업 작업의 상태를 검색하여 대괄호의 값을 사용자 환경의 정보로 바꿉니다.

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

Azure-NetApp-files(ANF) 작업을 위한 백업 및 복원이 가능합니다

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

다음을 확인합니다.

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

구성 단계를 위해 확장합니다

1. Trident 24.10으로 업그레이드하기 전에 ANF 볼륨을 생성한 경우 Trident에서 다음을 수행하십시오.

a. Azure-NetApp-files 기반이며 애플리케이션과 연결된 각 PV에 대해 스냅샷 디렉토리를 활성화합니다.

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

b. 연결된 각 PV에 대해 스냅샷 디렉토리가 활성화되었는지 확인합니다.

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

응답:

```
snapshotDirectory: "true"
```

+

스냅샷 디렉토리가 활성화되지 않은 경우 Trident Protect는 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.

Trident Protect를 사용하여 애플리케이션 복원

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



Application Restore 시 해당 Application에 설정된 모든 execution hook이 App으로 복구된다. 복원 후 실행 후크가 있는 경우 복구 작업의 일부로 자동으로 실행됩니다.

복원 및 페일오버 작업 중 네임스페이스 주석 및 레이블

복원 및 페일오버 작업 중에 대상 네임스페이스의 레이블과 주석이 소스 네임스페이스의 레이블 및 주석과 일치하도록 만듭니다. 대상 네임스페이스에 없는 소스 네임스페이스의 레이블 또는 주석이 추가되고 이미 존재하는 모든 레이블 또는 주석이 소스 네임스페이스의 값과 일치하도록 덮어쓰여집니다. 대상 네임스페이스에만 있는 레이블이나 주석은 변경되지 않습니다.



RedHat OpenShift를 사용하는 경우 OpenShift 환경에서 네임스페이스 주석의 중요한 역할을 주목해야 합니다. 네임스페이스 주석을 사용하면 복원된 Pod가 OpenShift SCC(Security Context Constraint)에서 정의한 적절한 권한 및 보안 구성을 준수하고 권한 문제 없이 볼륨에 액세스할 수 있습니다. 자세한 내용은 ["OpenShift 보안 컨텍스트 제약 조건 문서"](#)참조하십시오.

복원 또는 페일오버 작업을 수행하기 전에 Kubernetes 환경 변수를 설정하여 타겟 네임스페이스의 특정 주석을 덮어쓰지 않도록 할 수 있습니다 RESTORE_SKIP_NAMESPACE_ANNOTATIONS. 예를 들면 다음과 같습니다.

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

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 보여 줍니다. 작업 후후에 대상 네임스페이스의 상태와 대상 네임스페이스에서 주석과 레이블이 결합되거나 덮어쓰지는 방법을 확인할 수 있습니다.

복구 또는 페일오버 작업 전

다음 표에서는 복구 또는 페일오버 작업 이전의 예제 소스 및 대상 네임스페이스의 상태를 보여 줍니다.

네임스페이스	주석	라벨
네임스페이스 ns-1(소스)	<ul style="list-style-type: none"> • annotation.one/key:"updatedvalue"(주석 1개/키) • Annotation.two/key(주석.two/키): 	<ul style="list-style-type: none"> • 환경 = 운영 • 규정 준수 = HIPAA • 이름 = ns-1
네임스페이스 ns-2(대상)	<ul style="list-style-type: none"> • Annotation.one/key(주석. 하나/키):"true" • Annotation.Three/key:"false" 	<ul style="list-style-type: none"> • 역할 = 데이터베이스

복구 작업 후

다음 표에서는 복구 또는 페일오버 작업 후의 예제 대상 네임스페이스의 상태를 보여 줍니다. 일부 키가 추가되고, 일부 키가 덮어쓰여졌으며, name 대상 네임스페이스와 일치하도록 레이블이 업데이트되었습니다.

네임스페이스	주석	라벨
네임스페이스 ns-2(대상)	<ul style="list-style-type: none"> • annotation.one/key:"updatedvalue"(주석 1개/키) • Annotation.two/key(주석.two/키): • Annotation.Three/key:"false" 	<ul style="list-style-type: none"> • 이름 = ns-2 • 규정 준수 = HIPAA • 환경 = 운영 • 역할 = 데이터베이스

백업에서 다른 네임스페이스로 복원합니다

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 네임스페이스로 백업을 복원해도 백업의 리소스와 이름을 공유하는 리소스는 변경되지 않습니다. 백업의 모든 리소스를 복구하려면 타겟 네임스페이스를 삭제하고 다시 생성하거나 백업을 새 네임스페이스로 복원하십시오.

CR을 사용합니다

단계

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

- **metadata.name:** (*required*) 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
- *** spec.appArchivePath *:** 백업 콘텐츠가 저장되는 AppVault 내부의 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

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

- *** spec.appVaultRef *:** (*required*) 백업 내용이 저장된 AppVault의 이름입니다.
- **spec.namespaceMapping:** 복구 작업의 소스 네임스페이스를 대상 네임스페이스에 매핑하는 것입니다. `my-source-namespace` 및 를 `my-destination-namespace` 사용자 환경의 정보로 바꿉니다.
- **spec.storageClassMapping:** 복원 작업의 소스 스토리지 클래스를 대상 스토리지 클래스로 매핑하는 것입니다. `destinationStorageClass` 및 를 `sourceStorageClass` 사용자 환경의 정보로 바꿉니다.

```
---  
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"}]  
  storageClassMapping:  
    destination: "${destinationStorageClass}"  
    source: "${sourceStorageClass}"
```

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

- **resourceFilter.resourceSelectionCriteria:**(필터링에 필요) Include resourceMatchers에 정의된 리소스를 포함하거나 제외하려면 또는 을 Exclude 사용합니다. 다음 resourceMatchers 매개 변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
 - **resourceFilter.resourceMatchers:** resourceMatcher 개체의 배열입니다. 이 배열에서 여러 요소를 정의하는 경우 해당 요소는 OR 연산으로 일치하고 각 요소(그룹, 종류, 버전) 내의 필드는

AND 연산으로 일치합니다.

- **resourceMatchers[].group:**(*Optional*) 필터링할 리소스의 그룹입니다.
- *** resourceMatchers [].kind *:** (*Optional*) 필터링할 리소스의 종류입니다.
- **resourceMatchers [].version:** (*Optional*) 필터링할 리소스의 버전입니다.
- **resourceMatchers[].names:**(*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- ***resourceMatchers [].namespaces *:** (*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers [].labelSelectors:** (*Optional*) 에 정의된 대로 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "[Kubernetes 문서](#)" 예를 들면 다음과 "trident.netapp.io/os=linux" 같습니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 trident-protect-backup-restore-cr.yaml CR:

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

CLI를 사용합니다

단계

1. 대괄호 안의 값을 환경의 정보로 대체하여 백업을 다른 네임스페이스로 복원합니다. 이 namespace-mapping 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 올바른 대상 네임스페이스에 형식 `source1:dest1,source2:dest2`으로 매핑합니다. 예를 들면 다음과 같습니다.

```
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 복구합니다

언제든지 원래 네임스페이스로 백업을 복원할 수 있습니다.

CR을 사용합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-backup-ipr-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - * **spec.appArchivePath ***: 백업 콘텐츠가 저장되는 AppVault 내부의 경로입니다. 다음 명령을 사용하여 이 경로를 찾을 수 있습니다.

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

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

예를 들면 다음과 같습니다.

```
---  
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 응용 프로그램의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.
 - **resourceFilter.resourceSelectionCriteria:** (필터링에 필요) Include resourceMatchers에 정의된 리소스를 포함하거나 제외하려면 또는 을 Exclude 사용합니다. 다음 resourceMatchers 매개 변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
 - **resourceFilter.resourceMatchers:** resourceMatcher 개체의 배열입니다. 이 배열에서 여러 요소를 정의하는 경우 해당 요소는 OR 연산으로 일치하고 각 요소(그룹, 종류, 버전) 내의 필드는 AND 연산으로 일치합니다.
 - **resourceMatchers[].group:** *(Optional)* 필터링할 리소스의 그룹입니다.
 - * **resourceMatchers [].kind ***: *(Optional)* 필터링할 리소스의 종류입니다.
 - **resourceMatchers [].version:** *(Optional)* 필터링할 리소스의 버전입니다.
 - **resourceMatchers [].names:** *(Optional)* 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
 - ***resourceMatchers [].namespaces ***: *(Optional)* 필터링할 리소스의 Kubernetes

metadata.name 필드에 있는 네임스페이스입니다.

- **resourceMatchers [].labelSelectors:** (*Optional*) 에 정의된 대로 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "[Kubernetes 문서](#)" 예를 들면 다음과 "trident.netapp.io/os=linux" 같습니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 trident-protect-backup-ipr-cr.yaml CR:

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

CLI를 사용합니다

단계

1. 대괄호 안의 값을 환경의 정보로 대체하여 백업을 원래 네임스페이스로 복원합니다. backup`인수에 네임스페이스 및 백업 이름이 형식으로 `/` 사용됩니다. 예를 들면 다음과 같습니다.

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

백업에서 다른 클러스터로 복원합니다

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

- 시작하기 전에 *

다음 필수 구성 요소가 충족되는지 확인합니다.

- 대상 클러스터에 Trident Protect가 설치되어 있습니다.
- 대상 클러스터에는 백업이 저장되는 소스 클러스터와 동일한 AppVault의 버킷 경로에 대한 액세스 권한이 있습니다.

단계

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

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



응용 프로그램 복구용 네임스페이스가 대상 클러스터에 있는지 확인합니다.

2. 대상 클러스터에서 사용 가능한 AppVault의 백업 콘텐츠를 봅니다.

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

이 명령을 실행하면 원래 클러스터, 해당 응용 프로그램 이름, 타임스탬프 및 아카이브 경로를 비롯하여 AppVault에 사용 가능한 백업이 표시됩니다.

◦ 출력 예: *

```
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| 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 이름 및 보관 경로를 사용하여 응용 프로그램을 대상 클러스터로 복원합니다.

CR을 사용합니다

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

- **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
- *** spec.appVaultRef *:** *(required)* 백업 내용이 저장된 AppVault의 이름입니다.
- *** spec.appArchivePath *:** 백업 콘텐츠가 저장되는 AppVault 내부의 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:** 복구 작업의 소스 네임스페이스를 대상 네임스페이스에 매핑하는 것입니다. `my-source-namespace` 및 를 `my-destination-namespace` 사용자 환경의 정보로 바꿉니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 `trident-protect-backup-restore-cr.yaml` CR:

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

CLI를 사용합니다

1. 다음 명령을 사용하여 응용 프로그램을 복원하고 대괄호 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 들면 다음과 같습니다.

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

스냅샷에서 다른 네임스페이스로 복구합니다

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.

CR을 사용합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-snapshot-restore-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** (*required*) 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** spec.appVaultRef *:** (*required*) 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}'
```

- **spec.namespaceMapping:** 복구 작업의 소스 네임스페이스를 대상 네임스페이스에 매핑하는 것입니다. `my-source-namespace` 및 를 `my-destination-namespace` 사용자 환경의 정보로 바꿉니다.
- **spec.storageClassMapping:** 복원 작업의 소스 스토리지 클래스를 대상 스토리지 클래스로 매핑하는 것입니다. `destinationStorageClass` 및 를 `sourceStorageClass` 사용자 환경의 정보로 바꿉니다.

```
---  
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"}]  
  storageClassMapping:  
    destination: "${destinationStorageClass}"  
    source: "${sourceStorageClass}"
```

3. (선택 사항) 복원할 응용 프로그램의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.
 - **resourceFilter.resourceSelectionCriteria:**(필터링에 필요) Include resourceMatchers에 정의된 리소스를 포함하거나 제외하려면 또는 을 Exclude 사용합니다. 다음 resourceMatchers 매개 변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
 - **resourceFilter.resourceMatchers:** resourceMatcher 개체의 배열입니다. 이 배열에서 여러 요소를 정의하는 경우 해당 요소는 OR 연산으로 일치하고 각 요소(그룹, 종류, 버전) 내의 필드는

AND 연산으로 일치합니다.

- **resourceMatchers[].group:**(*Optional*) 필터링할 리소스의 그룹입니다.
- *** resourceMatchers [].kind *:** (*Optional*) 필터링할 리소스의 종류입니다.
- **resourceMatchers [].version:** (*Optional*) 필터링할 리소스의 버전입니다.
- **resourceMatchers[].names:**(*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- ***resourceMatchers [].namespaces *:** (*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers [].labelSelectors:** (*Optional*) 에 정의된 대로 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "[Kubernetes 문서](#)" 예를 들면 다음과 "trident.netapp.io/os=linux" 같습니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 trident-protect-snapshot-restore-cr.yaml CR:

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

CLI를 사용합니다

단계

1. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 스냅샷을 다른 네임스페이스로 복원합니다.

- snapshot `인수에 네임스페이스 및 스냅샷 이름이 형식으로 ``<namespace>/<name>` 사용됩니다.
- 이 namespace-mapping 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를

올바른 대상 네임스페이스에 형식 `source1:dest1,source2:dest2`으로 매핑합니다.

예를 들면 다음과 같습니다.

```
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 복구합니다

언제든지 스냅샷을 원래 네임스페이스로 복구할 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.

CR을 사용합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-snapshot-ipr-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** spec.appVaultRef *:** *(required)* 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 응용 프로그램의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

- **resourceFilter.resourceSelectionCriteria:** (필터링에 필요) Include resourceMatchers에 정의된 리소스를 포함하거나 제외하려면 또는 을 Exclude 사용합니다. 다음 resourceMatchers 매개 변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
 - **resourceFilter.resourceMatchers:** resourceMatcher 개체의 배열입니다. 이 배열에서 여러 요소를 정의하는 경우 해당 요소는 OR 연산으로 일치하고 각 요소(그룹, 종류, 버전) 내의 필드는 AND 연산으로 일치합니다.
 - **resourceMatchers[].group:** *(Optional)* 필터링할 리소스의 그룹입니다.
 - *** resourceMatchers [].kind *:** *(Optional)* 필터링할 리소스의 종류입니다.
 - **resourceMatchers [].version:** *(Optional)* 필터링할 리소스의 버전입니다.
 - **resourceMatchers[].names:** *(Optional)* 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
 - ***resourceMatchers [].namespaces *:** *(Optional)* 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
 - **resourceMatchers [].labelSelectors:** *(Optional)* 에 정의된 대로 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "[Kubernetes 문서](#)" 예를 들면 다음과

"trident.netapp.io/os=linux" 같습니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 trident-protect-snapshot-ipr-cr.yaml CR:

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

CLI를 사용합니다

단계

1. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 스냅샷을 원래 네임스페이스로 복원합니다. 예를 들면 다음과 같습니다.

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

복구 작업의 상태를 확인합니다

명령줄을 사용하여 진행 중이거나, 완료되었거나, 실패한 복구 작업의 상태를 확인할 수 있습니다.

단계

1. 다음 명령을 사용하여 복원 작업의 상태를 검색하여 대괄호의 값을 사용자 환경의 정보로 바꿉니다.

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

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 페일오버 작업 중 네임스페이스 주석 및 레이블

복원 및 페일오버 작업 중에 대상 네임스페이스의 레이블과 주석이 소스 네임스페이스의 레이블 및 주석과 일치하도록 만듭니다. 대상 네임스페이스에 없는 소스 네임스페이스의 레이블 또는 주석이 추가되고 이미 존재하는 모든 레이블 또는 주석이 소스 네임스페이스의 값과 일치하도록 덮어쓰여집니다. 대상 네임스페이스에만 있는 레이블이나 주석은 변경되지 않습니다.



RedHat OpenShift를 사용하는 경우 OpenShift 환경에서 네임스페이스 주석의 중요한 역할을 주목해야 합니다. 네임스페이스 주석을 사용하면 복원된 Pod가 OpenShift SCC(Security Context Constraint)에서 정의한 적절한 권한 및 보안 구성을 준수하고 권한 문제 없이 볼륨에 액세스할 수 있습니다. 자세한 내용은 ["OpenShift 보안 컨텍스트 제약 조건 문서"](#) 참조하십시오.

복원 또는 페일오버 작업을 수행하기 전에 Kubernetes 환경 변수를 설정하여 타겟 네임스페이스의 특정 주석을 덮어쓰지 않도록 할 수 있습니다 RESTORE_SKIP_NAMESPACE_ANNOTATIONS. 예를 들면 다음과 같습니다.

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

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 보여 줍니다. 작업 후후에 대상 네임스페이스의 상태와 대상 네임스페이스에서 주석과 레이블이 결합되거나 덮어쓰지는 방법을 확인할 수 있습니다.

복구 또는 페일오버 작업 전

다음 표에서는 복구 또는 페일오버 작업 이전의 예제 소스 및 대상 네임스페이스의 상태를 보여 줍니다.

네임스페이스	주석	라벨
네임스페이스 ns-1(소스)	<ul style="list-style-type: none"> • annotation.one/key:"updatedvalue"(주석 1개/키) • Annotation.two/key(주석.two/키): 	<ul style="list-style-type: none"> • 환경 = 운영 • 규정 준수 = HIPAA • 이름 = ns-1
네임스페이스 ns-2(대상)	<ul style="list-style-type: none"> • Annotation.one/key(주석. 하나/키):"true" • Annotation.Three/key:"false" 	<ul style="list-style-type: none"> • 역할 = 데이터베이스

복구 작업 후

다음 표에서는 복구 또는 페일오버 작업 후의 예제 대상 네임스페이스의 상태를 보여 줍니다. 일부 키가 추가되고, 일부 키가 덮어쓰여졌으며, name 대상 네임스페이스와 일치하도록 레이블이 업데이트되었습니다.

네임스페이스	주석	라벨
네임스페이스 ns-2(대상)	<ul style="list-style-type: none"> • annotation.one/key:"updatedvalue"(주석 1개/키) • Annotation.two/key(주석.two/키): • Annotation.Three/key:"false" 	<ul style="list-style-type: none"> • 이름 = ns-2 • 규정 준수 = HIPAA • 환경 = 운영 • 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."](#)

복제 관계를 설정합니다

복제 관계를 설정하려면 다음을 수행해야 합니다.

- 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 생성할 시간을 설정합니다

단계

1. 소스 클러스터에서 소스 애플리케이션에 대한 AppVault를 생성합니다. 스토리지 공급자에 따라 의 예를 ["AppVault 사용자 지정 리소스"](#)환경에 맞게 수정합니다.

CR을 사용하여 AppVault를 작성합니다

- a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: trident-protect-appvault-primary-source.yaml).
- b. 다음 특성을 구성합니다.
 - **metadata.name:** (*required*) AppVault 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일은 이 값을 참조하므로 선택한 이름을 기록해 둡니다.
 - **spec.providerConfig:** (*required*) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 구성을 저장합니다. bucketName과 공급자에게 필요한 기타 세부 정보를 선택합니다. 복제 관계에 필요한 다른 CR 파일은 이러한 값을 참조하므로 선택한 값을 기록해 둡니다. 다른 공급자의 AppVault CRS에 대한 예는 ["AppVault 사용자 지정 리소스"](#) 참조하십시오.
 - **spec.providerCredentials:** (*required*) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 자격 증명에 대한 참조를 저장합니다.
 - **spec.providerCredentials.valueFromSecret:** (*required*) 자격 증명 값이 비밀에서 와야 함을 나타냅니다.
 - * key *: (*required*) 선택할 비밀의 유효한 키입니다.
 - * name *: (*required*) 이 필드의 값을 포함하는 비밀의 이름입니다. 같은 네임스페이스에 있어야 합니다.
 - **spec.providerCredentials.secretAccessKey:** (*required*) 제공자에 액세스하는 데 사용되는 액세스 키입니다. name * 은 * spec.providerCredentials.valueFromSecret.name* 과 일치해야 합니다.
 - **spec.providerType:** (*required*) 백업을 제공하는 항목을 결정합니다(예: NetApp ONTAP S3, 일반 S3, Google Cloud 또는 Microsoft Azure). 가능한 값:
 - 설치하고
 - Azure를 지원합니다
 - GCP
 - 일반 - S3
 - ONTAP-S3
 - StorageGRID-S3
- c. 파일을 올바른 값으로 채운 후 trident-protect-appvault-primary-source.yaml CR:

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

CLI를 사용하여 AppVault를 작성합니다

- a. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 AppVault를 작성합니다.

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

2. 소스 응용 프로그램 생성 CR:

CR을 사용하여 소스 응용 프로그램을 만듭니다

- a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: trident-protect-app-source.yaml).
- b. 다음 특성을 구성합니다.
 - **metadata.name:** (required) 응용 프로그램 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일은 이 값을 참조하므로 선택한 이름을 기록해 둡니다.
 - **spec.includedNamespaces:** (required) 네임스페이스 및 관련 레이블의 배열입니다. 네임스페이스 이름을 사용하고 선택적으로 레이블을 사용하여 네임스페이스 범위를 좁혀 여기에 나열된 네임스페이스에 있는 리소스를 지정합니다. 응용 프로그램 네임스페이스는 이 배열의 일부여야 합니다.
 - YAML 예 *:

```
---
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. 파일을 올바른 값으로 채운 후 trident-protect-app-source.yaml CR:

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

CLI를 사용하여 소스 애플리케이션을 생성합니다

- a. 소스 응용 프로그램을 만듭니다. 예를 들면 다음과 같습니다.

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

3. 필요한 경우 소스 애플리케이션의 스냅샷을 생성합니다. 이 스냅샷은 대상 클러스터에서 애플리케이션의 기반으로 사용됩니다. 이 단계를 건너뛸 경우 다음 예약된 스냅샷이 실행될 때까지 기다려야 최신 스냅샷이 생성됩니다.

CR을 사용하여 스냅샷을 촬영합니다

a. 소스 애플리케이션에 대한 복제 스케줄을 생성합니다.

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

ii. 다음 특성을 구성합니다.

- **metadata.name:** *(required)* 일정 사용자 정의 리소스의 이름입니다.
- *** spec.AppVaultRef *:** *(required)* 이 값은 원본 응용 프로그램에 대한 AppVault의 `metadata.name` 필드와 일치해야 합니다.
- **spec.ApplicationRef:** *(required)* 이 값은 소스 응용 프로그램 CR의 `metadata.name` 필드와 일치해야 합니다.
- **spec.backupRetention:** *(required)* 이 필드는 필수 필드이며 값을 0으로 설정해야 합니다.
- *** spec.enabled *:** 반드시 `true`로 설정해야 합니다.
- **spec.granularity:** 으로 설정해야 `Custom` 합니다.
- **spec.recurrenceRule:** UTC 시간과 반복 간격으로 시작 날짜를 정의합니다.
- **spec.snapshotRetention:** 2로 설정해야 합니다.

YAML 예:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

i. 파일을 올바른 값으로 채운 후 `trident-protect-schedule.yaml` CR:

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

CLI를 사용하여 스냅샷을 생성합니다

- a. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 스냅샷을 생성합니다. 예를 들면 다음과 같습니다.

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

4. 소스 클러스터에 적용한 AppVault CR과 동일한 소스 응용 프로그램 AppVault CR을 대상 클러스터에 생성하고 이름을 지정합니다(예: trident-protect-appvault-primary-destination.yaml).

5. CR 적용:

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

6. 대상 클러스터에서 대상 응용 프로그램에 대한 AppVault를 생성합니다. 스토리지 공급자에 따라 의 예를 "AppVault 사용자 지정 리소스"환경에 맞게 수정합니다.

- a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: trident-protect-appvault-secondary-destination.yaml).

- b. 다음 특성을 구성합니다.

- **metadata.name:** (*required*) AppVault 사용자 정의 리소스의 이름입니다. 복제 관계에 필요한 다른 CR 파일은 이 값을 참조하므로 선택한 이름을 기록해 둡니다.
- **spec.providerConfig:** (*required*) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 구성을 저장합니다. `bucketName`와 공급자에게 필요한 기타 세부 정보를 선택합니다. 복제 관계에 필요한 다른 CR 파일은 이러한 값을 참조하므로 선택한 값을 기록해 둡니다. 다른 공급자의 AppVault CRS에 대한 예는 을 "AppVault 사용자 지정 리소스"참조하십시오.
- **spec.providerCredentials:** (*required*) 지정된 공급자를 사용하여 AppVault에 액세스하는 데 필요한 자격 증명에 대한 참조를 저장합니다.
 - **spec.providerCredentials.valueFromSecret:** (*required*) 자격 증명 값이 비밀에서 와야 함을 나타냅니다.
 - *** key *:** (*required*) 선택할 비밀의 유효한 키입니다.
 - *** name *:** (*required*) 이 필드의 값을 포함하는 비밀의 이름입니다. 같은 네임스페이스에 있어야 합니다.
 - **spec.providerCredentials.secretAccessKey:** (*required*) 제공자에 액세스하는 데 사용되는 액세스 키입니다. name * 은 * spec.providerCredentials.valueFromSecret.name* 과 일치해야 합니다.
- **spec.providerType:** (*required*) 백업을 제공하는 항목을 결정합니다(예: NetApp ONTAP S3, 일반 S3, Google Cloud 또는 Microsoft Azure). 가능한 값:
 - 설치하고
 - Azure를 지원합니다
 - GCP

- 일반 - S3
- ONTAP-S3
- StorageGRID-S3

c. 파일을 올바른 값으로 채운 후 `trident-protect-appvault-secondary-destination.yaml` CR:

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

7. AppMirrorRelationship CR 파일 만들기:

CR을 사용하여 AppMirrorRelationship을 생성합니다

- a. 사용자 정의 리소스(CR) 파일을 만들고 이름을 지정합니다(예: trident-protect-relationship.yaml).
- b. 다음 특성을 구성합니다.

- **metadata.name:** (필수) AppMirrorRelationship 사용자 정의 리소스의 이름입니다.
- **spec.destinationAppVaultRef:** (required) 이 값은 대상 클러스터의 대상 응용 프로그램에 대한 AppVault 이름과 일치해야 합니다.
- **spec.namespaceMapping:** (required) 대상 및 소스 네임스페이스는 해당 응용 프로그램 CR에 정의된 응용 프로그램 네임스페이스와 일치해야 합니다.
- *** spec.sourceAppVaultRef *:** (required) 이 값은 소스 응용 프로그램의 AppVault 이름과 일치해야 합니다.
- *** spec.sourceApplicationName *:** (required) 이 값은 소스 응용 프로그램 CR에서 정의한 소스 응용 프로그램의 이름과 일치해야 합니다.
- *** spec.storageClassName *:** (required) 클러스터에서 유효한 스토리지 클래스의 이름을 선택하십시오. 스토리지 클래스는 소스 환경을 피어링하는 ONTAP 스토리지 VM에 연결되어야 한다.
- **spec.recurrenceRule:** UTC 시간과 반복 간격으로 시작 날짜를 정의합니다.

YAML 예:

```
---
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-vsrm-2
```

- c. 파일을 올바른 값으로 채운 후 trident-protect-relationship.yaml CR:

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

CLI를 사용하여 AppMirrorRelationship을 생성한다

- a. AppMirrorRelationship 개체를 만들고 적용하여 대괄호 안의 값을 사용자 환경의 정보로 바꿉니다. 예를 들면 다음과 같습니다.

```
tridentctl-protect create appmirrorrelationship  
<name_of_appmirrorrelationship> --destination-app-vault  
<my_vault_name> --recurrence-rule <rule> --source-app  
<my_source_app> --source-app-vault <my_source_app_vault> -n  
<application_namespace>
```

8. (선택 사항) 복제 관계의 상태 및 상태를 확인합니다.

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

대상 클러스터로 페일오버합니다

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. AppMirrorRelationship CR 파일(예 trident-protect-relationship.yaml:)을 열고 * spec.desiredState* 값을 로 변경합니다 Promoted.
2. CR 파일을 저장합니다.
3. CR 적용:

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

4. (선택 사항) 페일오버된 애플리케이션에 필요한 보호 스케줄을 생성합니다.
5. (선택 사항) 복제 관계의 상태 및 상태를 확인합니다.

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

페일오버된 복제 관계를 다시 동기화합니다

재동기화 작업은 복제 관계를 다시 설정합니다. 재동기화 작업을 수행하면 원래 소스 애플리케이션이 실행 중인 애플리케이션이 되고 대상 클러스터에서 실행 중인 애플리케이션에 대한 변경 내용은 모두 삭제됩니다.

이 프로세스는 복제를 다시 설정하기 전에 대상 클러스터에서 앱을 중지합니다.



페일오버 중에 대상 애플리케이션에 기록된 모든 데이터가 손실됩니다.

단계

1. 소스 애플리케이션의 스냅샷을 생성합니다.
2. AppMirrorRelationship CR 파일(예: trident-protect-relationship.yaml)을 열고 spec.desiredState 값을 Established 로 변경합니다.
3. CR 파일을 저장합니다.
4. CR 적용:

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

5. 대상 클러스터에서 페일오버된 애플리케이션을 보호하기 위해 보호 스케줄을 생성한 경우 제거하십시오. 남아 있는 스케줄은 볼륨 스냅샷에 장애를 일으킵니다.

페일오버된 복제 관계를 역방향으로 재동기화합니다

페일오버된 복제 관계를 역동기화하는 경우 대상 애플리케이션은 소스 애플리케이션이 되고 소스는 대상이 됩니다. 페일오버 중에 대상 애플리케이션에 대한 변경 사항은 유지됩니다.

단계

1. 원래 대상 클러스터에서 AppMirrorRelationship CR을 삭제합니다. 그러면 대상이 원본이 됩니다. 새 대상 클러스터에 남아 있는 보호 스케줄이 있는 경우 제거합니다.
2. 관계를 설정할 때 원래 사용했던 CR 파일을 반대 클러스터에 적용하여 복제 관계를 설정합니다.
3. AppVault CRS가 각 클러스터에 준비되어 있는지 확인합니다.
4. 반대 클러스터에서 복제 관계를 설정하여 반대 방향에 대한 값을 구성합니다.

애플리케이션 복제 방향을 반대로 전환합니다

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 경우 소스와 대상을 스와핑합니다.

단계

1. 종료 스냅샷 생성:

CR을 사용하여 종료 스냅샷을 생성합니다

a. 소스 애플리케이션에 대한 보호 정책 일정을 해제합니다.

b. ShutdownSnapshot CR 파일 생성:

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

ii. 다음 특성을 구성합니다.

- **metadata.name:** *(required)* 사용자 정의 리소스의 이름입니다.
- *** spec.AppVaultRef *:** *(required)* 이 값은 원본 응용 프로그램에 대한 AppVault의 metadata.name 필드와 일치해야 합니다.
- **spec.ApplicationRef:** *(required)* 이 값은 소스 응용 프로그램 CR 파일의 metadata.name 필드와 일치해야 합니다.

YAML 예:

```
---
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. 파일을 올바른 값으로 채운 후 trident-protect-shutdownsnapshot.yaml CR:

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

CLI를 사용하여 종료 스냅샷을 생성합니다

a. 괄호 안의 값을 사용자 환경의 정보로 대체하여 종료 스냅샷을 만듭니다. 예를 들면 다음과 같습니다.

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

2. 스냅샷이 완료되면 스냅샷의 상태를 가져옵니다.

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

3. 다음 명령을 사용하여 * shutdownsnapshot.status.appArchivePath * 의 값을 찾고 파일 경로의 마지막 부분(basename이라고도 함. 이것은 마지막 슬래시 다음에 모두 있음)을 기록합니다.

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

4. 다음과 같이 변경하여 대상 클러스터에서 소스 클러스터로 페일오버를 수행합니다.



페일오버 절차의 2단계에서 AppMirrorRelationship CR 파일에 필드를 포함하고 spec.promotedSnapshot 위의 3단계에서 기록한 기본 이름으로 값을 설정합니다.

5. 의 역동기화 단계를 **페일오버된 복제 관계를 역방향으로 재동기화합니다**수행합니다.

6. 새 소스 클러스터에서 보호 스케줄을 설정합니다.

결과

역방향 복제 때문에 다음 작업이 발생합니다.

- 원본 소스 앱의 Kubernetes 리소스에 대한 스냅샷이 생성됩니다.
- 앱의 Kubernetes 리소스를 삭제하여 원본 소스 앱의 Pod를 정상적으로 중지할 수 있습니다(PVC 및 PVS를 그대로 둡니다).
- 포드가 종료된 후 앱 볼륨의 스냅샷이 촬영되고 복제됩니다.
- SnapMirror 관계가 끊어져 타겟 볼륨이 읽기/쓰기 준비가 되었습니다.
- 앱의 Kubernetes 리소스는 원래 소스 애플리케이션이 종료된 후 복제된 볼륨 데이터를 사용하여 사전 종료 스냅샷에서 복구됩니다.
- 복제는 반대 방향으로 다시 설정됩니다.

애플리케이션을 원래 소스 클러스터로 페일백합니다

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 페일오버를 완료한 관계로부터 시작되며 다음 단계를 포함합니다.

- 페일오버된 상태로 시작합니다.
- 복제 관계를 역방향으로 다시 동기화합니다.



일반 재동기화 작업을 수행하지 마십시오. 그러면 페일오버 절차 중에 대상 클러스터에 기록된 데이터가 삭제됩니다.

- 복제 방향을 반대로 바꿉니다.

단계

1. **페일오버된 복제 관계를 역방향으로 재동기화합니다** 다음 단계를 수행합니다.
2. **애플리케이션 복제 방향을 반대로 전환합니다** 다음 단계를 수행합니다.

복제 관계를 삭제합니다

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

단계

1. AppMirrorRelationship CR 삭제:

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

Trident Protect를 사용하여 애플리케이션 마이그레이션

백업 또는 스냅샷 데이터를 다른 클러스터 또는 스토리지 클래스로 복원하여 클러스터 또는 스토리지 클래스 간에 애플리케이션을 마이그레이션할 수 있습니다.



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

백업 및 복원 작업

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

같은 클러스터에 클론 복제

애플리케이션을 동일한 클러스터에 복제하려면 스냅샷 또는 백업을 생성하고 데이터를 동일한 클러스터에 복구합니다.

단계

1. 다음 중 하나를 수행합니다.
 - a. **"스냅샷을 생성합니다"**..
 - b. **"백업을 생성합니다"**..
2. 동일한 클러스터에서 스냅샷 또는 백업을 생성했는지에 따라 다음 중 하나를 수행합니다.
 - a. **"스냅샷에서 데이터를 복원합니다"**..
 - b. **"백업에서 데이터를 복원합니다"**..

다른 클러스터에 클론 복제

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.



을 사용하여 서로 다른 클러스터 간에 애플리케이션을 복제할 수 "[SnapMirror 복제](#)" 있습니다.

단계

1. "백업을 생성합니다"..
2. 백업이 포함된 객체 스토리지 버킷의 AppVault CR이 대상 클러스터에 구성되어 있는지 확인합니다.
3. 대상 클러스터에서 "[백업에서 데이터를 복원합니다](#)".

한 스토리지 클래스에서 다른 스토리지 클래스로 애플리케이션을 마이그레이션합니다

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

예를 들어 복원 CR에서 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
```

CR을 사용하여 스냅샷을 복원합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-snapshot-restore-cr.yaml` 지정합니다.
2. 생성한 파일에서 다음 속성을 구성합니다.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** 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 *:** *(required)* 스냅샷 내용이 저장된 AppVault의 이름입니다.
- **spec.namespaceMapping:** 복구 작업의 소스 네임스페이스를 대상 네임스페이스에 매핑하는 것입니다. `my-source-namespace` 및 `my-destination-namespace` 사용자 환경의 정보로 바꿉니다.

```
---  
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. 복원할 응용 프로그램의 특정 리소스만 선택해야 하는 경우 특정 레이블로 표시된 리소스를 포함하거나 제외하는 필터링을 추가합니다.

- **resourceFilter.resourceSelectionCriteria:** (필터링에 필요) `include` or `exclude` `resourceMatchers`에 정의된 리소스를 포함하거나 제외하는 데 사용합니다. 다음 `resourceMatchers` 매개 변수를 추가하여 포함하거나 제외할 리소스를 정의합니다.
 - **resourceFilter.resourceMatchers:** `resourceMatcher` 개체의 배열입니다. 이 배열에서 여러 요소를 정의하는 경우 해당 요소는 OR 연산으로 일치하고 각 요소(그룹, 종류, 버전) 내의 필드는 AND 연산으로 일치합니다.
 - **resourceMatchers[].group:** *(Optional)* 필터링할 리소스의 그룹입니다.
 - *** resourceMatchers [].kind *:** *(Optional)* 필터링할 리소스의 종류입니다.
 - **resourceMatchers [].version:** *(Optional)* 필터링할 리소스의 버전입니다.

- **resourceMatchers[].names:** (*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 이름입니다.
- ***resourceMatchers [].namespaces *:** (*Optional*) 필터링할 리소스의 Kubernetes metadata.name 필드에 있는 네임스페이스입니다.
- **resourceMatchers [].labelSelectors:** (*Optional*) 에 정의된 대로 리소스의 Kubernetes metadata.name 필드에 있는 레이블 선택기 문자열입니다. "[Kubernetes 문서](#)" 예를 들면 다음과 "trident.netapp.io/os=linux" 같습니다.

예를 들면 다음과 같습니다.

```
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. 파일을 올바른 값으로 채운 후 trident-protect-snapshot-restore-cr.yaml CR:

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

CLI를 사용하여 스냅샷을 복원합니다

단계

1. 대괄호 안의 값을 사용자 환경의 정보로 대체하여 스냅샷을 다른 네임스페이스로 복원합니다.
 - snapshot `인수에 네임스페이스 및 스냅샷 이름이 형식으로 ``<namespace>/<name>` 사용됩니다.
 - 이 namespace-mapping 인수는 콜론으로 구분된 네임스페이스를 사용하여 소스 네임스페이스를 올바른 대상 네임스페이스에 형식 ``source1:dest1,source2:dest2``으로 매핑합니다.

예를 들면 다음과 같습니다.

```
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. 따라서 애플리케이션 정합성이 보장되는 스냅샷이 보장됩니다.

실행 후크 유형

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

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

실행 순서

데이터 보호 작업이 실행되면 실행 후크 이벤트가 다음 순서로 발생합니다.

1. 해당되는 모든 사용자 정의 사전 작업 실행 후크는 해당 컨테이너에서 실행됩니다. 필요한 만큼 사용자 지정 사전 작업 후크를 만들고 실행할 수 있지만, 이 후크의 실행 순서는 보장되거나 구성할 수 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 시스템은 동결 해제됩니다.
5. 해당되는 모든 사용자 지정 작업 후 실행 후크는 해당 컨테이너에서 실행됩니다. 필요한 만큼 사용자 지정 사후 작업 후크를 만들고 실행할 수 있지만 작업 후 후크의 실행 순서는 보장되거나 구성할 수 없습니다.

같은 유형의 실행 후크를 여러 개 생성하는 경우(예: 사전 스냅샷) 해당 후크의 실행 순서는 보장되지 않습니다. 그러나 다른 유형의 후크를 실행하는 순서는 보장됩니다. 예를 들어, 다음과 같은 순서로 서로 다른 유형의 후크가 모두 있는 구성을 실행할 수 있습니다.

1. 사전 스냅샷 후크가 실행되었습니다
2. 사후 스냅샷 후크가 실행되었습니다
3. 예비 후크가 실행되었습니다

4. 백업 후 후크가 실행되었습니다



위의 순서 예제는 기존 스냅샷을 사용하지 않는 백업을 실행하는 경우에만 적용됩니다.



운영 환경에서 실행 후크 스크립트를 사용하려면 항상 해당 스크립트를 테스트해야 합니다. 'kubbeck exec' 명령을 사용하여 스크립트를 편리하게 테스트할 수 있습니다. 운영 환경에서 실행 후크를 사용하도록 설정한 후 결과 스냅샷과 백업을 테스트하여 적합성이 보장되는지 확인합니다. 앱을 임시 네임스페이스에 클론 복제하고, 스냅샷 또는 백업을 복원한 다음 앱을 테스트하여 이 작업을 수행할 수 있습니다.

사용자 정의 실행 후크에 대한 중요 참고 사항

앱에 대한 실행 후크를 계획할 때 다음 사항을 고려하십시오.

- 실행 후크는 스크립트를 사용하여 작업을 수행해야 합니다. 많은 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 실행 중인 응용 프로그램의 기능을 줄이거나 완전히 비활성화하기 때문에 사용자 지정 실행 후크가 실행되는 시간을 최소화해야 합니다. 연결된 실행 후크와 함께 백업 또는 스냅샷 작업을 시작한 다음 취소하면 백업 또는 스냅샷 작업이 이미 시작된 경우에도 후크를 실행할 수 있습니다. 즉, 백업 후 실행 후크에 사용되는 논리는 백업이 완료된 것으로 가정할 수 없습니다.

실행 후크 필터

응용 프로그램의 실행 후크를 추가하거나 편집할 때 실행 후크에 필터를 추가하여 후크가 일치할 컨테이너를 관리할 수 있습니다. 필터는 모든 컨테이너에서 동일한 컨테이너 이미지를 사용하는 응용 프로그램에 유용하지만 각 이미지를 다른 용도(예: Elasticsearch)로 사용할 수 있습니다. 필터를 사용하면 실행 후크가 실행되는 시나리오를 만들 수 있습니다. 단, 모든 동일한 컨테이너를 실행하는 것은 아닙니다. 단일 실행 후크에 대해 여러 개의 필터를 만들면 논리적 필터 및 연산자와 결합됩니다. 실행 후크당 최대 10개의 활성 필터를 사용할 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. ["정규식 2\(RE2\) 구문 지원"](#).



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

실행 후크 예

를 방문하여 ["NetApp Verda GitHub 프로젝트"](#) Apache Cassandra 및 Elasticsearch와 같은 인기 앱의 실제 실행 후크를 다운로드하십시오. 예제를 보고 사용자 지정 실행 후크를 구조화하는 아이디어를 얻을 수도 있습니다.

실행 후크를 만듭니다

Trident Protect를 사용하여 앱에 대한 사용자 정의 실행 후크를 만들 수 있습니다. 실행 후크를 생성하려면 소유자, 관리자 또는 멤버 권한이 필요합니다.

CR을 사용합니다

단계

1. CR(사용자 정의 리소스) 파일을 만들고 이름을 `trident-protect-hook.yaml` 지정합니다.
2. Trident Protect 환경 및 클러스터 구성에 맞게 다음 속성을 구성하세요.
 - **metadata.name:** *(required)* 이 사용자 정의 리소스의 이름입니다. 사용자 환경에 맞는 고유하고 합리적인 이름을 선택하십시오.
 - *** spec.applicationRef *:** *(required)* 실행 후크를 실행할 응용 프로그램의 Kubernetes 이름입니다.
 - *** spec.stage *:** *(required)* 실행 후크가 실행되어야 하는 작업 중 단계를 나타내는 문자열입니다. 가능한 값:
 - 사전
 - 게시
 - *** spec.action *:** *(required)* 지정된 실행 후크 필터가 일치한다고 가정할 때 실행 후크가 수행할 작업을 나타내는 문자열입니다. 가능한 값:
 - 스냅샷
 - 백업
 - 복원
 - 파일오버
 - **spec.enabled:** *(Optional)* 이 실행 후크의 활성화 여부를 나타냅니다. 지정하지 않으면 기본값은 true 입니다.
 - **spec.hookSource:** *(required)* base64로 인코딩된 후크 스크립트를 포함하는 문자열입니다.
 - **spec.timeout:** *(Optional)* 실행 후크가 실행될 수 있는 시간을 분 단위로 정의하는 숫자입니다. 최소값은 1분이고, 지정하지 않은 경우 기본값은 25분입니다.
 - **spec.arguments:** *(Optional)* 실행 후크에 지정할 수 있는 인수의 YAML 목록입니다.
 - *** spec.matchingCriteria *:** *(Optional)* 각 쌍이 실행 후크 필터를 구성하는 조건 키 값 쌍의 선택적 목록입니다. 실행 후크당 최대 10개의 필터를 추가할 수 있습니다.
 - *** spec.matchingCriteria.type *:** *(Optional)* 실행 후크 필터 유형을 식별하는 문자열입니다. 가능한 값:
 - 컨테이너이미지
 - 컨테이너명
 - PodName을 선택합니다
 - PodLabel을 선택합니다
 - 이름 이름 이름
 - *** spec.matchingCriteria.value *:** *(Optional)* 실행 후크 필터 값을 식별하는 문자열 또는 정규식입니다.

YAML 예:

```

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: IyEvYmluL2Jhc2gKZWNoYAiZXhhbXBsZSBzY3JpcHQiCg==
  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 파일을 올바른 값으로 채운 후 CR:

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

CLI를 사용합니다

단계

1. 괄호 안의 값을 사용자 환경의 정보로 대체하여 실행 후크를 만듭니다. 예를 들면 다음과 같습니다.

```
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>
```

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.

```
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 인쇄된 본 문서의 어떠한 부분도 저작권 소유자의 사전 서면 승인 없이는 어떠한 형식이나 수단(복사, 녹음, 녹화 또는 전자 검색 시스템에 저장하는 것을 비롯한 그래픽, 전자적 또는 기계적 방법)으로도 복제될 수 없습니다.

NetApp이 저작권을 가진 자료에 있는 소프트웨어에는 아래의 라이선스와 고지사항이 적용됩니다.

본 소프트웨어는 NetApp에 의해 '있는 그대로' 제공되며 상품성 및 특정 목적에의 적합성에 대한 명시적 또는 묵시적 보증을 포함하여(이에 제한되지 않음) 어떠한 보증도 하지 않습니다. NetApp은 대체품 또는 대체 서비스의 조달, 사용 불능, 데이터 손실, 이익 손실, 영업 중단을 포함하여(이에 국한되지 않음), 이 소프트웨어의 사용으로 인해 발생하는 모든 직접 및 간접 손해, 우발적 손해, 특별 손해, 징벌적 손해, 결과적 손해의 발생에 대하여 그 발생 이유, 책임론, 계약 여부, 엄격한 책임, 불법 행위(과실 또는 그렇지 않은 경우)와 관계없이 어떠한 책임도 지지 않으며, 이와 같은 손실의 발생 가능성이 통지되었다 하더라도 마찬가지입니다.

NetApp은 본 문서에 설명된 제품을 언제든지 예고 없이 변경할 권리를 보유합니다. NetApp은 NetApp의 명시적인 서면 동의를 받은 경우를 제외하고 본 문서에 설명된 제품을 사용하여 발생하는 어떠한 문제에도 책임을 지지 않습니다. 본 제품의 사용 또는 구매의 경우 NetApp에서는 어떠한 특허권, 상표권 또는 기타 지적 재산권이 적용되는 라이선스도 제공하지 않습니다.

본 설명서에 설명된 제품은 하나 이상의 미국 특허, 해외 특허 또는 출원 중인 특허로 보호됩니다.

제한적 권리 표시: 정부에 의한 사용, 복제 또는 공개에는 DFARS 252.227-7013(2014년 2월) 및 FAR 52.227-19(2007년 12월)의 기술 데이터-비상업적 품목에 대한 권리(Rights in Technical Data -Noncommercial Items) 조항의 하위 조항 (b)(3)에 설명된 제한사항이 적용됩니다.

여기에 포함된 데이터는 상업용 제품 및/또는 상업용 서비스(FAR 2.101에 정의)에 해당하며 NetApp, Inc.의 독점 자산입니다. 본 계약에 따라 제공되는 모든 NetApp 기술 데이터 및 컴퓨터 소프트웨어는 본질적으로 상업용이며 개인 비용만으로 개발되었습니다. 미국 정부는 데이터가 제공된 미국 계약과 관련하여 해당 계약을 지원하는 데에만 데이터에 대한 전 세계적으로 비독점적이고 양도할 수 없으며 재사용이 불가능하며 취소 불가능한 라이선스를 제한적으로 가집니다. 여기에 제공된 경우를 제외하고 NetApp, Inc.의 사전 서면 승인 없이는 이 데이터를 사용, 공개, 재생산, 수정, 수행 또는 표시할 수 없습니다. 미국 국방부에 대한 정부 라이선스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

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