使用 **Trident Protect** 保護應用程式

Trident

NetApp
March 13, 2025

目錄

使用 Trident Protect 保護應用程式

瞭解 Trident Protect

NetApp Trident Protect 提供進階的應用程式資料管理功能，可強化由 NetApp ONTAP 儲存系統和 NetApp Trident CSI 儲存資源配置程式所支援的狀態化 Kubernetes 應用程式的功能與可用度。Trident Protect 可簡化跨公有雲和內部部署環境的容器化工作負載管理，保護和移動。它也透過 API 和 CLI 提供自動化功能。

您可以 Trident 建立自訂資源（ CRS ），或使用 Trident Protect CLI 來保護應用程式。

接下來呢？

您可以在安裝 Trident Protect 要求之前先瞭解相關資訊：

- "Trident 保護需求"

安裝 Trident Protect

Trident 保護需求

首先，請確認您的營運環境，應用程式叢集，應用程式和授權是否準備就緒。確保您的環境符合這些需求，以部署及操作 Trident Protect 。

Trident 保護 Kubernetes 叢集相容性

Trident Protect 可與多種完全託管且自我管理的 Kubernetes 產品相容，包括：

- Amazon Elastic Kubernetes Service （ EKS ）
- Google Kubernetes Engine（GKE）
- Microsoft Azure Kubernetes服務（英文）
- Red Hat OpenShift
- SUSE Rancher
- VMware Tanzu產品組合
- 上游Kubernetes

請確定您安裝 Trident Protect 的叢集已設定執行中的快照控制器和相關的 CRD 。若要安裝快照控制器，請 "這些指示"參閱。

Trident 保護儲存後端相容性

Trident Protect 支援下列儲存設備後端：

- Amazon FSX for NetApp ONTAP 產品

- Cloud Volumes ONTAP
- ONTAP 儲存陣列
- Google Cloud NetApp Volumes
- Azure NetApp Files

確保您的儲存後端符合下列需求：

- 確保連接至叢集的 NetApp 儲存設備使用 Astra Trident 24.02 或更新版本（建議使用 Trident 24.10 ）。
  - 如果 Astra Trident 早於 24.06.1 版，且您計畫使用 NetApp SnapMirror 災難恢復功能，則需要手動啟用 Astra 控制項資源配置程式。
- 確保您擁有最新的 Astra 控制備份程式（預設為 Astra Trident 24.06.1 ）。
- 確保您擁有 NetApp ONTAP 儲存後端。
- 請確定您已設定物件儲存貯體以儲存備份。
- 建立您計畫用於應用程式或應用程式資料管理作業的任何應用程式命名空間。Trident Protect 不會為您建立這些命名空間；如果您在自訂資源中指定不存在的命名空間，則作業將會失敗。

NAS 經濟容量需求

Trident Protect 支援 NAS 經濟型磁碟區的備份與還原作業。目前不支援快照，複製和 SnapMirror 複寫至 NAS 經濟型磁碟區。您需要為打算搭配 Trident Protect 使用的每個 NAS 經濟型磁碟區啟用快照目錄。

某些應用程式與使用 Snapshot 目錄的磁碟區不相容。對於這些應用程式，您需要在 ONTAP 儲存系統上執行下列命令，以隱藏快照目錄：

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

您可以針對每個 NAS 經濟型磁碟區執行下列命令，以您要變更的磁碟區 UUID 取代，來啟用 Snapshot 目錄 `<volume-UUID>`：

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

您可以將 Trident 後端組態選項設定為，為 `true`**新的磁碟區預設啟用快照目錄** `snapshotDir`。現有的磁碟區不受影響。

使用 **KubeVirt VM** 保護資料

當您保護在 KubeVirt VM 上執行的應用程式時， Trident Protect 24.10 和 24.10.1 及更新版本的行為會有所不同。對於這兩個版本，您可以在資料保護作業期間啟用或停用檔案系統凍結和解除凍結。

Trident Protect 24.10

Trident Protect 24.10 無法在資料保護作業期間，自動確保 KubeVirt VM 檔案系統的狀態一致。如果您想要使用 Trident Protect 24.10 來保護 KubeVirt VM 資料，則必須在資料保護作業之前，手動啟用檔案系統的凍結 / 取消凍結功能。如此可確保檔案系統處於一致的狀態。

您可以將 Trident Protect 24.10 設定為在資料保護作業期間管理 VM 檔案系統的凍結和取消凍結"設定虛擬化"，然後使用下列命令：

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

Trident Protect 24.10.1 及更新版本

從 Trident Protect 24.10.1 開始， Trident Protect 會在資料保護作業期間，自動凍結和取消凍結 KubeVirt 檔案系統。您也可以使用下列命令停用此自動行為：

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

SnapMirror 複寫需求

NetApp SnapMirror 複寫可與 Trident Protect 搭配使用，適用於下列 ONTAP 解決方案：

- 內部部署 NetApp FAS ， AFF 和 ASA 叢集
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSX for NetApp ONTAP 產品

SnapMirror 複寫的 ONTAP 叢集需求

如果您打算使用 SnapMirror 複寫，請確保 ONTAP 叢集符合下列需求：

- * Astra 控制備份程式或 Trident * ： Astra 控制備份程式或 Trident 必須同時存在於使用 ONTAP 作為後端的來源叢集和目的地 Kubernetes 叢集上。Trident Protect 使用下列驅動程式所支援的儲存類別，以 NetApp SnapMirror 技術支援複寫：
  - 「ONTAP-NAS」
  - 「ONTAP-SAN」
- * 授權 * ：使用資料保護套件的 ONTAP SnapMirror 非同步授權必須同時在來源和目的地 ONTAP 叢集上啟用。如需詳細資訊、請參閱 "SnapMirror授權概述ONTAP" 。

SnapMirror 複寫的對等考量

如果您計畫使用儲存後端對等，請確保您的環境符合下列需求：

- * 叢集與 SVM* ：必須對 ONTAP 儲存設備的後端進行對等處理。如需詳細資訊、請參閱 "叢集與SVM對等概觀" 。

  確保兩個 ONTAP 叢集之間複寫關係中使用的 SVM 名稱是唯一的。

- **Astra** 控制資源配置程式或 **Trident** 和 **SVM** ：對等的遠端 SVM 必須可用於目的地叢集上的 Astra 控制資源配置程式或 Trident 。

- * 託管後端 * ：您需要在 Trident Protect 中新增及管理 ONTAP 儲存後端，才能建立複寫關係。
- **NVMe over TCP** ： Trident Protect 不支援 NetApp SnapMirror 複寫，用於使用 NVMe over TCP 傳輸協定的儲存後端。

用於 **SnapMirror** 複寫的 **Trident / ONTAP** 組態

Trident Protect 要求您至少設定一個儲存後端，以支援來源叢集和目的地叢集的複寫。如果來源叢集和目的地叢集相同、則目的地應用程式應使用不同於來源應用程式的儲存後端、以獲得最佳恢復能力。

安裝及設定 **Trident Protect**

如果您的環境符合 Trident Protect 的要求，您可以依照下列步驟在叢集上安裝 Trident Protect 。您可以從 NetApp 取得 Trident Protect ，或從您自己的私有登錄安裝。如果您的叢集無法存取網際網路，從私有登錄安裝會很有幫助。

安裝 **Trident Protect**

安裝 **Trident Protect from NetApp**

步驟

1. 新增Trident Helm儲存庫：

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

2. 安裝 Trident Protect 客戶需求日：

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2502.0 --create-namespace --namespace
trident-protect
```

3. 使用 Helm 安裝 Trident Protect 。以叢集名稱取代 `<name_of_cluster>`，該名稱將指派給叢集，用於識別叢集的備份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2502.0 --create
-namespace --namespace trident-protect
```

從私有登錄安裝 **Trident Protect**

如果 Kubernetes 叢集無法存取網際網路，您可以從私有映像登錄安裝 Trident Protect 。在這些範例中，請將方括號中的值取代為環境中的資訊：

步驟

1. 將下列影像拉到您的本機電腦，更新標記，然後將它們推送到您的私人登錄：

```
netapp/controller:25.02.0
netapp/restic:25.02.0
netapp/kopia:25.02.0
netapp/trident-autosupport:25.02.0
netapp/exechook:25.02.0
netapp/resourcebackup:25.02.0
netapp/resourcerestore:25.02.0
netapp/resourcedelete:25.02.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例如：

```
docker pull netapp/controller:25.02.0
```

```
docker tag netapp/controller:25.02.0 <private-registry-
url>/controller:25.02.0
```

```
docker push <private-registry-url>/controller:25.02.0
```

2. 建立 Trident Protect 系統命名空間：

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

3. 登入登錄：

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

4. 建立用於私人登錄驗證的拉出密碼：

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

5. 新增Trident Helm儲存庫：

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

6. 建立名為的檔案 `protectValues.yaml`。請確定其中包含下列 Trident Protect 設定：

```
---
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. 安裝 Trident Protect 客戶需求日：

```
helm install trident-protect-crds netapp-trident-protect/trident-protect-crds --version 100.2502.0 --create-namespace --namespace trident-protect
```

8. 使用 Helm 安裝 Trident Protect 。以叢集名稱取代 `<name_of_cluster>`，該名稱將指派給叢集，用於識別叢集的備份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name_of_cluster> --version 100.2502.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

安裝 **Trident Protect CLI** 外掛程式

您可以使用 Trident Protect 命令列外掛程式（ Trident 公用程式的延伸 `tridentctl`）來建立自訂資源（ CRS ），並與 Trident 互動。

安裝 **Trident Protect CLI** 外掛程式

在使用命令列公用程式之前，您必須先將其安裝在用來存取叢集的機器上。根據您的機器使用的是 x64 或 ARM CPU ，請遵循下列步驟。

下載適用於 **Linux AMD64 CPU** 的外掛程式

步驟

1. 下載 Trident Protect CLI 外掛程式：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.02.0/tridentctl-protect-linux-amd64
```

下載適用於 **Linux ARM64 CPU** 的外掛程式

步驟

1. 下載 Trident Protect CLI 外掛程式：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.02.0/tridentctl-protect-linux-arm64
```

下載適用於 **Mac AMD64 CPU** 的外掛程式

步驟

1. 下載 Trident Protect CLI 外掛程式：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.02.0/tridentctl-protect-macos-amd64
```

下載 **Mac ARM64 CPU** 的外掛程式

步驟

1. 下載 Trident Protect CLI 外掛程式：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.02.0/tridentctl-protect-macos-arm64
```

1. 啟用外掛程式二進位檔的執行權限：

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

2. 將外掛程式二進位檔複製到路徑變數中定義的位置。例如， `/usr/bin`或 `/usr/local/bin`（您可能需要提升的 Privileges ）：

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

3. 您也可以選擇將外掛程式二進位檔複製到主目錄中的某個位置。在這種情況下，建議您確保位置是 PATH 變數的一部分：

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

將外掛程式複製到 PATH 變數中的某個位置，可讓您輸入或 `tridentctl protect`從任何位置使用外掛程式 `tridentctl-protect`。

檢視 **Trident CLI** 外掛程式說明

您可以使用內建的外掛程式說明功能，取得外掛程式功能的詳細說明：

步驟

1. 使用說明功能檢視使用指南：

```
tridentctl-protect help
```

啟用命令自動完成

安裝 Trident Protect CLI 外掛程式之後，您可以啟用某些命令的自動完成功能。

啟用 **Bash Shell** 的自動完成功能

步驟

1. 下載完成指令碼：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.02.0/tridentctl-completion.bash
```

2. 在主目錄中建立新目錄以包含指令碼：

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

3. 將下載的指令碼移至 `~/.bash/completions`目錄：

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

4. 將下列行新增至 `~/.bashrc`主目錄中的檔案：

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

啟用 **Z Shell** 的自動完成功能

步驟

1. 下載完成指令碼：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.02.0/tridentctl-completion.zsh
```

2. 在主目錄中建立新目錄以包含指令碼：

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

3. 將下載的指令碼移至 `~/.zsh/completions`目錄：

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

4. 將下列行新增至 `~/.zprofile`主目錄中的檔案：

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

結果

下次登入 Shell 時，您可以將命令自動完成功能與 tridentctl-Protect 外掛程式搭配使用。

自訂 Trident Protect 安裝

您可以自訂 Trident Protect 的預設組態，以符合您環境的特定需求。

指定 Trident Protect 容器資源限制

安裝 Trident Protect 之後，您可以使用組態檔來指定 Trident Protect 容器的資源限制。設定資源限制可讓您控制 Trident Protect 作業消耗多少叢集資源。

步驟

1. 建立名為的檔案 `resourceLimits.yaml`。
2. 根據您的環境需求，在檔案中填入 Trident Protect 容器的資源限制選項。

   以下範例組態檔顯示可用的設定，並包含每個資源限制的預設值：

```
---
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. 套用檔案中的值 `resourceLimits.yaml`：

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

自訂安全性內容限制

安裝 Trident Protect 之後，您可以使用組態檔來修改 Trident Protect 容器的 OpenShift 安全性內容限制（ SCC ）。這些限制定義了 Red Hat OpenShift 叢集中 Pod 的安全性限制。

步驟

1. 建立名為的檔案 `sccconfig.yaml`。
2. 將 SCC 選項新增至檔案，並根據環境需求修改參數。

   以下範例顯示 SCC 選項參數的預設值：

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

下表說明 SCC 選項的參數：

參數	說明	預設
建立	決定是否可以建立 SCC 資源。只有在設定為 `true`且 `Helm` 安裝程序識別 `OpenShift` 環境時，才會建立 `SCC` 資源 `scc.create`。如果未在 OpenShift 上操作，或如果設為 `false`，則 `scc.create`不會建立任何 SCC 資源。	是的
名稱	指定SCC的名稱。	Trident 保護工作
優先順序	定義 SCC 的優先順序。優先順序值較高的 SCC 會在值較低之前進行評估。	1

3. 套用檔案中的值 `sccconfig.yaml`：

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

這會將預設值取代為檔案中指定的值 `sccconfig.yaml`。

設定 **Trident Protect** 的 **NetApp AutoSupport** 連線

您可以設定連線的 Proxy ，變更 Trident Protect 連線至 NetApp 支援的方式，以上傳支援套件。您可以根據需要將 Proxy 設定為使用安全連線或不安全連線。

設定安全 **Proxy** 連線

步驟

1. 設定安全的 Proxy 連線以上傳 Trident Protect 支援服務包：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect --set autoSupport.proxy=http://my.proxy.url
--reuse-values
```

設定不安全的 **Proxy** 連線

步驟

1. 設定不安全的 Proxy 連線，以進行 Trident Protect 支援服務套件上傳，以略過 TLS 驗證：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect --set autoSupport.proxy=http://my.proxy.url
--set autoSupport.insecure=true --reuse-values
```

將 **Trident** 保護 **Pod** 限制在特定節點

您可以使用 Kubernetes nodeSelector 節點選擇限制，根據節點標籤來控制哪些節點符合執行 Trident Protect Pod 的資格。根據預設， Trident Protect 僅限於執行 Linux 的節點。您可以根據自己的需求，進一步自訂這些限制。

步驟

1. 建立名為的檔案 `nodeSelectorConfig.yaml`。
2. 將 nodeSelector 選項新增至檔案，並修改檔案以新增或變更節點標籤，以根據環境需求加以限制。例如，下列檔案包含預設的作業系統限制，但也針對特定區域和應用程式名稱：

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

3. 套用檔案中的值 `nodeSelectorConfig.yaml`：

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

這會將預設限制取代為您在檔案中指定的限制 `nodeSelectorConfig.yaml`。

停用每日 **Trident Protect AutoSupport** 套件上傳

您也可以停用排定的每日 Trident Protect AutoSupport 支援服務套件上傳。

根據預設， Trident Protect 會收集支援資訊，協助處理您可能開啟的任何 NetApp 支援案例，包括叢集和託管應用程式的記錄，度量和拓撲資訊。Trident Protect 會根據每日排程將這些支援套裝組合傳送至 NetApp 。您可以隨時手動"產生支援服務組合"進行。

步驟

1. 建立名為的檔案 `autosupportconfig.yaml`。
2. 將 AutoSupport 選項新增至檔案，並根據環境需求修改參數。

   下列範例顯示 AutoSupport 選項參數的預設值：

   ```
   autoSupport:
     enabled: true
   ```

   當 `autoSupport.enabled`設為 `false`時， AutoSupport 支援套裝組合的每日上傳會停用。

3. 套用檔案中的值 `autosupportconfig.yaml`：

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

管理 Trident Protect

管理 **Trident** 保護授權與存取控制

Trident Protect 採用 Kubernetes 角色型存取控制（ RBAC ）模式。根據預設， Trident Protect 提供單一系統命名空間及其相關的預設服務帳戶。如果組織有許多使用者或特定的安全需求，您可以使用 Trident Protect 的 RBAC 功能，更精細地控制對資源和命名空間的存取。

叢集管理員一律可以存取預設命名空間中的資源 `trident-protect`，也可以存取所有其他命名空間中的資源。若要控制對資源和應用程式的存取，您需要建立額外的命名空間，並將資源和應用程式新增至這些命名空間。

請注意，沒有使用者可以在預設命名空間中建立應用程式資料管理 CRS `trident-protect` 。您需要在應用程式命名空間中建立應用程式資料管理 CRS （最佳做法是在與其相關應用程式相同的命名空間中建立應用程式資料管理 CRS ）。

只有系統管理員才能存取授權的 Trident 保護自訂資源物件，包括：

- * AppVault* ：需要儲存庫認證資料
- * AutoSupportBundle * ：收集指標，記錄及其他敏感的 Trident Protect 資料
- * AutoSupportBundleSchedule* ：管理記錄收集排程

最佳做法是使用 RBAC 來限制系統管理員存取權限物件。

如需 RBAC 如何規範資源和命名空間存取的詳細資訊，請參閱 "Kubernetes RBAC 文件"。

如需服務帳戶的相關資訊，請參閱 "Kubernetes 服務帳戶文件"。

範例：管理兩組使用者的存取權

例如，組織有叢集管理員，一組工程設計使用者，以及一組行銷使用者。叢集管理員將完成下列工作，以建立一個環境，其中工程群組和行銷群組各自只能存取指派給各自命名空間的資源。

步驟 **1** ：建立命名空間以包含每個群組的資源

建立命名空間可讓您以邏輯方式分隔資源，並更有效地控制誰有權存取這些資源。

步驟

1. 為工程群組建立命名空間：

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

2. 為行銷群組建立命名空間：

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

步驟 **2** ：建立新的服務帳戶，與每個命名空間中的資源互動

您所建立的每個新命名空間都有預設服務帳戶，但您應該為每個使用者群組建立服務帳戶，以便日後在必要時在群組之間進一步分割 Privileges 。

步驟

1. 為工程群組建立服務帳戶：

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

2. 為行銷群組建立服務帳戶：

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

步驟 **3** ：為每個新的服務帳戶建立秘密

服務帳戶密碼是用來驗證服務帳戶，如果受到入侵，也可以輕鬆刪除和重新建立。

步驟

1. 為工程服務帳戶建立秘密：

```
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. 為行銷服務帳戶建立秘密：

```
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
```

步驟 **4** ：建立 **RoleBinding** 物件，將 **ClusterRole** 物件繫結至每個新的服務帳戶

安裝 Trident Protect 時會建立預設的 ClusterRole 物件。您可以建立並套用角色繫結物件，將此 ClusterRole 繫結至服務帳戶。

步驟

1. 將 ClusterRole 繫結至工程服務帳戶：

```
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. 將 ClusterRole 連結至行銷服務帳戶：

```
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
```

步驟 **5** ：測試權限

測試權限是否正確。

步驟

1. 確認工程使用者可以存取工程資源：

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

2. 確認工程使用者無法存取行銷資源：

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

步驟 6 ：授予對 AppVault 物件的存取權

若要執行資料管理工作，例如備份和快照，叢集管理員必須將 AppVault 物件的存取權授予個別使用者。

步驟

1. 建立並套用 AppVault 和加密組合 YAML 檔案，以授予使用者存取 AppVault 的權限。例如，下列 CR 將 AppVault 的存取權授予使用者 `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. 建立並套用角色 CR ，讓叢集管理員能夠授與對命名空間中特定資源的存取權。例如：

```
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. 建立並套用 RoleBinding CR ，將權限繫結至使用者 eng-user 。例如：

```
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. 確認權限正確。

   a. 嘗試擷取所有命名空間的 AppVault 物件資訊：

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

   您應該會看到類似下列的輸出：

```
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. 測試以查看使用者是否能取得他們現在有權存取的 AppVault 資訊：

```
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
```

您應該會看到類似下列的輸出：

```
yes
```

結果

您已授予 AppVault 權限的使用者應該能夠使用授權的 AppVault 物件來執行應用程式資料管理作業，而且不應能夠存取指派命名空間以外的任何資源，或建立他們無法存取的新資源。

監控 **Trident Protect** 資源

您可以使用 kube-state 度量， Prometheus 和 Alertmanager 開放原始碼工具來監控受 Trident Protect 保護的資源健全狀況。

kube-state 度量服務會從 Kubernetes API 通訊產生度量。搭配 Trident Protect 使用可提供環境中資源狀態的實用資訊。

Prometheus 是一個工具組，可擷取由 kube 狀態度量所產生的資料，並將其呈現為這些物件的易讀資訊。kube 狀態指標和 Prometheus 共同提供一種方法，讓您使用 Trident Protect 監控所管理資源的健全狀況和狀態。

AlertManager 是一項服務，可擷取 Prometheus 等工具所傳送的警示，並將其路由至您設定的目的地。

> 這些步驟所包含的組態和指南僅為範例，您需要自訂以符合您的環境。請參閱下列正式文件，以取得特定指示與支援：
>
> - "Kube-state指標文件"
> - "Prometheus 文件"
> - "AlertManager 文件"

步驟 **1** ：安裝監控工具

若要在 Trident Protect 中啟用資源監控，您必須安裝及設定 kube-st狀態 度量， Promethus 和 Alertmanager 。

安裝 **kube** 狀態度量

您可以使用 Helm 來安裝 kube 狀態度量。

步驟

1. 新增 kube 狀態指標 Helm 圖表。例如：

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

2. 為 Helm 圖表建立組態檔（例如 `metrics-config.yaml`）。您可以自訂下列範例組態，以符合您的環境：

測量數據 **-config.yaml** ： **Kube-state** 度量 **Helm** 圖表組態

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

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: astra.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: ["backups.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
```

3. 部署 Helm 圖表以安裝 kube 狀態度量。例如：

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

4. 依照中的指示，設定 kube 狀態度量，以產生 Trident Protect 所使用之自訂資源的度量 "Kube-state 度量自訂資源文件"。

安裝Prometheus

您可以依照中的指示來安裝 Prometheus "Prometheus 文件" 。

安裝 AlertManager

您可以依照中的指示安裝 AlertManager "AlertManager 文件" 。

步驟 2 ：設定監控工具以共同作業

安裝監控工具之後，您需要將它們設定為一起運作。

步驟

1. 將 kube 狀態指標與 Prometheus 整合。編輯 Prometheus 配置文件(`prometheus.yml`）並添加 kube 狀態指標服務信息。例如：

Prometheus.yml ：與 Prometheus 整合的 Kube-state 度量服務

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

2. 設定 Prometheus 將警示路由至 AlertManager 。編輯 Prometheus 配置文件(`prometheus.yml`）並添加以下部分：

Prometheus.yml ：傳送警示給 **Alertmanager**

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.astra-connector.svc:9093
```

結果

現在， Prometheus 可以從 kube-state 度量收集度量，並可傳送警示給 Alertmanager 。您現在已準備好設定觸發警示的條件，以及應傳送警示的位置。

步驟 **3** ：設定警示和警示目的地

設定工具以共同作業之後，您需要設定觸發警示的資訊類型，以及應傳送警示的位置。

警示範例：備份失敗

以下範例定義當備份自訂資源的狀態設定為 5 秒或更長時間時觸發的關鍵警示 `Error`。您可以自訂此範例以符合您的環境，並將此 YAML 片段包含在組態檔案中 `prometheus.yml`：

rules.yml ：定義失敗備份的 **Prometheus** 警示

```
rules.yml: |
  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."
```

設定 **AlertManager** 以傳送警示至其他頻道

您可以將 AlertManager 設定為傳送通知給其他通道，例如電子郵件， PagerDuty ， Microsoft 團隊或其他通知服務，方法是在檔案中指定個別的組態 `alertmanager.yml`。

以下範例將警示管理員設定為傳送通知至 Slack 頻道。若要根據您的環境自訂此範例，請將金鑰的值取代為 `api_url`您環境中使用的 Slack Webhook URL ：

alertmanager.yml ：傳送警示至 **Slack** 頻道

```
data:
  alertmanager.yml: |
    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
```

產生 **Trident Protect** 支援服務組合

Trident Protect 可讓系統管理員產生套件組合，其中包含 NetApp 支援所需的資訊，包括所管理叢集和應用程式的記錄，度量和拓撲資訊。如果您已連線至網際網路，則可以使用自訂資源（ CR ）檔案，將支援套件上傳至 NetApp 支援網站（ NSS ）。

使用 **CR** 建立支援服務組合

步驟

1. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-support-bundle.yaml`）。
2. 設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.triggerType** ：（ _required _ ）決定是立即產生支援套件，還是排程產生。排定的套件產生時間為上午 12 點， UTC 。可能值：
     - 已排程
     - 手冊
   - **SPEC.uploadEnabled** ：（ *Optional* ）控制是否應在支援服務組合產生後，將其上傳至 NetApp 支援網站。如果未指定，則默認爲 `false`。可能值：
     - 是的
     - 否（預設）
   - **spec.daWindowStart** ：（ *Optional* ） RFC 3339 格式的日期字串，指定支援套件中所包含資料的視窗應開始的日期與時間。如果未指定，則預設為 24 小時前。您可以指定的最早時間是 7 天前。

     YAML 範例：

```
---
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. 在您以正確的值填入檔案之後 `astra-support-bundle.yaml` 、請套用 CR ：

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

使用 **CLI** 建立支援服務包

步驟

1. 建立支援服務組合，以環境資訊取代括號中的值。 `trigger-type`決定套件是立即建立，還是建立時間取決於排程，可以是 `Manual`或 `Scheduled`。預設設定為 `Manual`。

   例如：

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

升級 **Trident Protect**

您可以將 Trident Protect 升級至最新版本，以利用新功能或錯誤修正。

若要升級 Trident Protect ，請執行下列步驟。

步驟

1. 更新 Trident Helm 儲存庫：

```
helm repo update
```

2. 升級 Trident Protect 客戶需求日：

```
helm upgrade trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2502.0  --namespace trident-protect
```

3. 升級 Trident Protect ：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2502.0 --namespace trident-protect
```

管理及保護應用程式

使用 **Trident Protect AppVault** 物件來管理貯體

Trident Protect 的貯體自訂資源（ CR ）稱為 AppVault 。AppVault 物件是儲存貯體的宣告性 Kubernetes 工作流程表示。AppVault CR 包含用於保護作業（例如備份，快照，還原作業和 SnapMirror 複寫）的儲存庫所需的組態。只有管理員可以建立 AppVaults 。

設定 **AppVault** 驗證和密碼

在建立 AppVault 物件之前，您必須確保 AppVault 和您選擇的資料移動器可以向提供者和任何相關資源進行驗證。

資料移動器儲存庫密碼

當您使用 CRS 或 Trident Protect CLI 外掛程式建立 AppVault 物件時，您可以選擇性地指示 Trident Protect 使用 Kubernetes 機密，其中包含 Restic 和 Kopia 儲存庫加密的自訂密碼。如果您未指定密碼， Trident Protect

會使用預設密碼。

- 當手動建立 AppVault CRS 時，您可以使用 **spec.dataMoverPasswordSecretRef** 欄位來指定密碼。
- 使用 Trident Protect CLI 建立 AppVault 物件時，您可以使用 `--data-mover-password-secret-ref`引數來指定機密。

建立資料移動者儲存庫密碼機密

請使用下列範例建立密碼機密。當您建立 AppVault 物件時，可以指示 Trident Protect 使用此密碼來驗證資料移動器儲存庫。

視您使用的資料移動器而定，您只需要加入該資料移動器的對應密碼。例如，如果您使用 Restic ，而且不打算在未來使用 Kopia ，則在建立機密時，只能包含 Restic 密碼。

使用 **CR**

```
---
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
```

使用**CLI**

```
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
```

適用於雲端供應商的 **AppVault** 主要世代範例

定義 AppVault CR 時，您需要加入認證，才能存取供應商託管的資源。您產生認證金鑰的方式會因供應商而異。以下是多個提供者的命令列金鑰產生範例。您可以使用下列範例來建立每個雲端供應商認證的金鑰。

Google Cloud

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

Microsoft Azure

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

一般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
```

StorageGRID 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 建立範例

以下是每個提供者的 AppVault 定義範例。

AppVault CR 範例

您可以使用下列 CR 範例，為每個雲端供應商建立 AppVault 物件。

- 您可以選擇性地指定 Kubernetes 機密，其中包含 Restic 和 Kopia 儲存庫加密的自訂密碼。如需詳細資訊、請參閱 [資料移動器儲存庫密碼] 。
- 對於 S3 AppVault 物件，您可以選擇性地指定一個工作區權杖，如果您使用單一登入（ SSO ）進行驗證，這個功能就很有用。當您在中為提供者產生金鑰時適用於雲端供應商的 AppVault 主要世代範例，就會建立此權杖。
- 對於 S3 AppVault 物件，您可以選擇使用金鑰來指定傳出 S3 流量的外傳 Proxy URL `spec.providerConfig.S3.proxyURL` 。

Google Cloud

```
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
```

Amazon 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
```

Microsoft Azure

```
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
```

一般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
    sessionToken:
      valueFromSecret:
        key: sessionToken
        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
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3_secret
```

StorageGRID S3

```yaml
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
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3_secret
```

使用 **Trident Protect CLI** 建立 **AppVault** 範例

您可以使用下列 CLI 命令範例，為每個供應商建立 AppVault CRS 。

- 您可以選擇性地指定 Kubernetes 機密，其中包含 Restic 和 Kopia 儲存庫加密的自訂密碼。如需詳細資訊、請參閱 [資料移動器儲存庫密碼] 。
- 對於 S3 AppVault 物件，您可以選擇使用引數，為輸出 S3 流量指定外傳 Proxy URL `--proxy-url <ip_address:port>` 。

Google Cloud

```
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
```

Amazon 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
```

Microsoft Azure

```
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
```

一般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
```

StorageGRID 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
```

檢視 **AppVault** 資訊

您可以使用 Trident Protect CLI 外掛程式來檢視您在叢集上建立的 AppVault 物件相關資訊。

步驟

1. 檢視 AppVault 物件的內容：

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

   - 輸出範例 * ：

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   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. （可選）要查看每個資源的 AppVaultPath ，請使用標誌 `--show-paths`。

   只有在 Trident Protect helm 安裝中指定叢集名稱時，表格第一欄中的叢集名稱才能使用。例如 `--set clusterName=production1`：。

移除 **AppVault**

您可以隨時移除 AppVault 物件。

> ⓘ 刪除 AppVault 物件之前，請勿移除 `finalizers` AppVault CR 中的機碼。如果您這麼做，可能會導致 AppVault 貯體中的剩餘資料，以及叢集中的孤立資源。

開始之前

請確定您已刪除要刪除的 AppVault 所使用的所有快照和備份 CRS 。

使用 **Kubernetes CLI** 移除 **AppVault**

1. 移除 AppVault 物件，以要移除的 AppVault 物件名稱取代 `appvault_name`：

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

使用 **Trident Protect CLI** 移除 **AppVault**

1. 移除 AppVault 物件，以要移除的 AppVault 物件名稱取代 `appvault_name`：

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

使用 **Trident Protect** 定義管理應用程式

您可以建立應用程式 CR 和相關的 AppVault CR ，以定義您想要使用 Trident Protect 管理的應用程式。

建立 **AppVault CR**

您需要建立 AppVault CR ，以便在應用程式上執行資料保護作業時使用，而 AppVault CR 必須位於安裝 Trident Protect 的叢集上。AppVault CR 專屬於您的環境，如需 AppVault CRS 的範例，請參閱"AppVault 自訂資源。"

定義應用程式

您需要定義每個要使用 Trident Protect 管理的應用程式。您可以手動建立應用程式 CR 或使用 Trident Protect CLI 來定義應用程式以進行管理。

使用 **CR** 新增應用程式

步驟

1. 建立目的地應用程式 CR 檔案：
   a. 建立自訂資源（ CR ）檔案並命名（例如 `maria-app.yaml`）。
   b. 設定下列屬性：
      - * metadata.name*: （ _required ）應用程式自訂資源的名稱。請注意您選擇的名稱，因為保護作業所需的其他 CR 檔案都會參照此值。
      - * spec.includedNamespaces*: （ _required _ ）使用命名空間標籤或命名空間名稱來指定應用程式資源所在的命名空間。應用程式命名空間必須是此清單的一部分。
      - * metadata.annotations.protect.trident.netapp.io/skip-vm-freeze*: （ *Optional* ）此註釋僅適用於從虛擬機器定義的應用程式，例如 KubeVirt 環境，檔案系統會在快照之前凍結。指定此應用程式是否可以在快照期間寫入檔案系統。如果設為 true ，應用程式會忽略全域設定，並在快照期間寫入檔案系統。如果設置爲 false ，則應用程序將忽略全局設置，並在快照期間凍結文件系統。如果指定，但應用程式在應用程式定義中沒有虛擬機器，則會忽略附註。如果未指定，則應用程式會遵循"全域 Trident 保護凍結設定"。

如果您需要在建立應用程式之後套用此註釋，可以使用下列命令：

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

+
YAML 範例：

+

```
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:
  - labelSelector: {}
    namespace: namespace-1
  - labelSelector: {}
    namespace: namespace-2
```

1. 建立應用程式 CR 以符合您的環境之後，請套用 CR 。例如：

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

步驟

1. 使用下列其中一個範例建立及套用應用程式定義，以環境中的資訊取代方括號中的值。您可以在應用程式定義中加入命名空間和資源，使用以逗號分隔的清單，以及範例中所示的引數。

   您可以選擇在建立應用程式時使用註釋，以指定應用程式是否可以在快照期間寫入檔案系統。這僅適用於從虛擬機器定義的應用程式，例如 KubeVirt 環境，檔案系統會在快照之前凍結。如果您將註釋設為 `true`，應用程式會忽略全域設定，並在快照期間寫入檔案系統。如果將其設置爲 `false`，則應用程序將忽略全局設置，並在快照期間凍結文件系統。如果您使用附註，但應用程式在應用程式定義中沒有虛擬機器，則會忽略附註。如果您不使用註釋，應用程式會遵循"全域 Trident 保護凍結設定"。

   若要在使用 CLI 建立應用程式時指定評註，您可以使用此 `--annotation`旗標。

   - 建立應用程式，並使用通用設定來執行檔案系統凍結行為：

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

   - 建立應用程式並設定檔案系統凍結行為的本機應用程式設定：

```
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">
```

使用 **Trident Protect** 保護應用程式

您可以使用自動保護原則或臨機操作方式，拍攝快照和備份，以保護由 Trident Protect 管理的所有應用程式。

您可以將 Trident Protect 設定為在資料保護作業期間凍結和取消凍結檔案系統。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。

建立隨需快照

您可以隨時建立隨需快照。

如果叢集範圍的資源在應用程式定義中明確參照，或是具有任何應用程式命名空間的參照，則這些資源會包含在備份，快照或複製中。

使用 **CR** 建立快照

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-snapshot-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **SPEC.applicationRef** ：要快照的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef** ：（ _required _ ）應儲存快照內容（中繼資料）的 AppVault 名稱。
   - **spec.relaimersPolicy** ：（ *Optional* ）定義刪除快照 CR 時，應用程式歸檔會發生什麼情況。這表示即使設定為，快照也 `Retain`會被刪除。有效選項：
     - `Retain` （預設）
     - `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. 在您以正確的值填入檔案之後 `trident-protect-snapshot-cr.yaml` 、請套用 CR ：

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

使用 **CLI** 建立快照

步驟

1. 建立快照，以您環境的資訊取代方括號中的值。例如：

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

建立隨選備份

您可以隨時備份應用程式。

如果叢集範圍的資源在應用程式定義中明確參照，或是具有任何應用程式命名空間的參照，則這些資源會包含在備份，快照或複製中。

開始之前

確保 AWS 工作階段權杖到期時間足以應付任何長期執行的 S3 備份作業。如果 Token 在備份作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR** 建立備份

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-backup-cr.yaml` 。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **SPEC.applicationRef** ：（ _required _ ）要備份的應用程式 Kubernetes 名稱。
   - **spec.appVaultRef** ：（ _required _ ）應儲存備份內容的 AppVault 名稱。
   - *spec.dataMover * ：（ *Optional* ）字串，指出備份作業所使用的備份工具。可能的值（區分大小寫）：
     - `Restic`
     - `Kopia` （預設）
   - **spec.reClaimPolicy** ：（ *Optional* ）定義備份從宣告中釋出時會發生什麼情況。可能值：
     - `Delete`
     - `Retain` （預設）
   - **Spec.snapshotRef** ：（ *Optional* ）：用於備份來源的快照名稱。如果未提供，將會建立並備份暫存快照。

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-backup-cr.yaml` 、請套用 CR ：

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

使用 **CLI** 建立備份

步驟

1. 建立備份，以您環境的資訊取代括號中的值。例如：

```
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>
```

建立資料保護排程

保護原則可在已定義的排程中建立快照、備份或兩者、以保護應用程式。您可以選擇每小時、每天、每週和每月建立快照和備份、也可以指定要保留的複本數量。

如果叢集範圍的資源在應用程式定義中明確參照，或是具有任何應用程式命名空間的參照，則這些資源會包含在備份，快照或複製中。

開始之前

確保 AWS 工作階段權杖到期時間足以應付任何長期執行的 S3 備份作業。如果 Token 在備份作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR** 建立排程

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-schedule-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - *spec.dataMover * ：（ *Optional* ）字串，指出備份作業所使用的備份工具。可能的值（區分大小寫）：
     - `Restic`
     - `Kopia` （預設）
   - **SPEC.applicationRef** ：要備份之應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef** ：（ _required _ ）應儲存備份內容的 AppVault 名稱。
   - *SPEC.BackupRetention * ：要保留的備份數量。零表示不應建立備份。
   - *spec.snapshotRetention * ：要保留的快照數。零表示不應建立任何快照。
   - * spec.granularity*: 執行排程的頻率。可能的值、以及必要的相關欄位：
     - `hourly` （要求您指定 `spec.minute`）
     - `daily` （要求您指定 `spec.minute` 和 `spec.hour`）
     - `weekly`（要求您指定 `spec.minute, spec.hour`，和 `spec.dayOfWeek`）
     - `monthly`（要求您指定 `spec.minute, spec.hour`，和 `spec.dayOfMonth`）
   - **spec.dayOfMontth** ：（ *Optional* ）排程應執行的月份日期（ 1 - 31 ）。如果精細度設為、則此欄位為必 `monthly`填。
   - *spec.dayOfWeek * ：（ _Optional _ ）排程應執行的一週中的一天（ 0 - 7 ）。0 或 7 的值表示星期日。如果精細度設為、則此欄位為必 `weekly`填。
   - *spec.hour * ：（ _Optional _ ）排程應執行的一天中的小時（ 0 - 23 ）。如果精細度設置爲、或，則此字段爲必填字段 `daily` `weekly` `monthly`。
   - * 規格分鐘 * ：（ _ 選用 _ ）排程應執行的小時（ 0 - 59 ）分鐘。如果精細度設置爲、、或，則此字段爲必填字段 `hourly` `daily` `weekly` `monthly`。

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-schedule-cr.yaml` 、請套用 CR ：

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

使用 **CLI** 建立排程

步驟

1. 建立保護排程，以環境資訊取代方括號中的值。例如：

> 您可以使用 `tridentctl-protect create schedule --help`來檢視此命令的詳細說明資訊。

```
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>
```

刪除快照

刪除不再需要的排程或隨需快照。

步驟

1. 移除與快照相關的 Snapshot CR ：

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

刪除備份

刪除不再需要的排程或隨需備份。

步驟

1. 移除與備份相關的備份 CR ：

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

檢查備份作業的狀態

您可以使用命令列來檢查正在進行，已完成或已失敗的備份作業狀態。

步驟

1. 使用下列命令可擷取備份作業的狀態，以環境中的資訊取代方括號中的值：

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

啟用 **NetApp** 檔案（ **anf** ）作業的備份與還原

如果您已安裝 Trident Protect ，您可以啟用節省空間的備份與還原功能，以供使用 NetApp 檔案儲存類別的儲存後端使用，並在 Trident 24.06 之前建立。此功能可與 NFSv4 磁碟區搭配使用，不會佔用容量集區的額外空間。

開始之前

請確認下列事項：

- 您已安裝 Trident Protect 。
- 您已在 Trident Protect 中定義應用程式。在您完成此程序之前、此應用程式的保護功能有限。
- 您已 `azure-netapp-files` 選擇儲存後端的預設儲存類別。

展開以進行組態步驟

1. 如果 anf Volume 是在升級至 Trident 24.10 之前建立的，請在 Trident 中執行下列動作：

   a. 針對每個以 NetApp 檔案為基礎且與應用程式相關的 PV ，啟用快照目錄：

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

   b. 確認已為每個相關的 PV 啟用快照目錄：

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

   回應：

   ```
   snapshotDirectory: "true"
   ```

   +
   未啟用 Snapshot 目錄時， Trident Protect 會選擇一般備份功能，在備份程序期間會暫時佔用容量集區中的空間。在這種情況下，請確保容量集區中有足夠的可用空間，以建立備份磁碟區大小的暫存磁碟區。

結果

應用程式已準備好使用 Trident Protect 進行備份與還原。每個 PVC 也可供其他應用程式用於備份和還原。

使用 **Trident Protect** 還原應用程式

您可以使用 Trident Protect 從快照或備份還原應用程式。將應用程式還原至同一個叢集時、從現有的快照還原速度會更快。

當您還原應用程式時，為應用程式設定的所有執行掛鉤都會隨應用程式一起還原。如果存在還原後執行掛鉤，則會在還原作業中自動執行。

還原和容錯移轉作業期間的命名空間註釋和標籤

在還原和容錯移轉作業期間，目的地命名空間中的標籤和註釋會與來源命名空間中的標籤和註釋相符。會新增來源命名空間中不存在的標籤或註釋，並覆寫已存在的任何標籤或註釋，以符合來源命名空間的值。只存在於目的地命名空間上的標籤或註釋會保持不變。

如果您使用 RedHat OpenShift ，請務必注意 OpenShift 環境中命名空間註釋的關鍵作用。命名空間註釋可確保還原的 Pod 符合 OpenShift 安全性內容限制（ SCC ）所定義的適當權限和安全性組態，而且無需權限問題即可存取磁碟區。如需詳細資訊，請 "OpenShift 安全性內容限制文件"參閱。

您可以在執行還原或容錯移轉作業之前，先設定 Kubernetes 環境變數，以避免覆寫目的地命名空間中的特定註釋 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。例如：

```
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>
```

如果您使用帶有標誌的 Helm 來安裝來源應用程式 `--create-namespace`，則會對標籤機碼給予特殊處理 `name`。在還原或容錯移轉程序期間， Trident Protect 會將此標籤複製到目的地命名空間，但如果來源的值符合來源命名空間，則會將值更新到目的地命名空間值。如果此值與來源命名空間不相符，則會將其複製到目的地命名空間，而不會有任何變更。

範例

以下範例提供來源和目的地命名空間，每個命名空間都有不同的註釋和標籤。您可以查看作業前後目的地命名空間的狀態，以及註釋和標籤在目的地命名空間中的組合或覆寫方式。

還原或容錯移轉作業之前

下表說明還原或容錯移轉作業之前的範例來源和目的地命名空間狀態：

命名空間	註釋	標籤
命名空間 nS-1 （來源）	• annotation.one / 機碼：「 updatedvalue 」 • annotation.b2/key ：「 true 」	• 環境 = 正式作業 • Compliance = HIPAA • NAME=ns-1
命名空間 nS-2 （目的地）	• annotation.one / 機碼：「 true 」 • annotation.the/key ：「 FALSE 」	• role = 資料庫

還原作業之後

下表說明還原或容錯移轉作業之後範例目的地命名空間的狀態。某些金鑰已新增，部分已覆寫， `name`標籤已更新以符合目的地命名空間：

命名空間	註釋	標籤
命名空間 nS-2 （目的地）	• annotation.one / 機碼：「 updatedvalue 」 • annotation.b2/key ：「 true 」 • annotation.the/key ：「 FALSE 」	• NAME=nS-2 • Compliance = HIPAA • 環境 = 正式作業 • role = 資料庫

從備份還原至不同的命名空間

當您使用 BackupRestore CR 將備份還原至不同的命名空間時， Trident Protect 會將應用程式還原至新命名空間，並為還原的應用程式建立應用程式 CR 。若要保護還原的應用程式，請建立隨需備份或快照，或建立保護排程。

> 將備份還原至具有現有資源的不同命名空間，並不會改變任何與備份中共用名稱的資源。若要還原備份中的所有資源，請刪除並重新建立目標命名空間，或將備份還原至新的命名空間。

開始之前

確保 AWS 工作階段權杖到期時間足以執行任何長時間執行的 S3 還原作業。如果 Token 在還原作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-backup-restore-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appArchivePath** ：儲存備份內容的 AppVault 內部路徑。您可以使用下列命令來尋找此路徑：

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

   - **spec.appVaultRef** ：（ _required _ ）儲存備份內容的 AppVault 名稱。
   - **spec.namespaceMapping**: 將還原作業的來源命名空間對應至目的地命名空間。以環境中的資訊取代 `my-source-namespace`和 `my-destination-namespace`。
   - **spec.storageClassMapping** ：將還原作業的來源儲存類別對應至目的地儲存類別。以環境中的資訊取代 `destinationStorageClass`和 `sourceStorageClass`。

     ```
     ---
     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. （ *Optional* ）如果您只需要選取應用程式的某些資源來還原，請新增篩選功能，以包含或排除標記有特定標籤的資源：

   Trident Protect 會自動選取部分資源，因為這些資源與您選取的資源之間的關係。例如，如果您選取持續磁碟區宣告資源，而且該資源有相關聯的 Pod ，則 Trident Protect 也會還原相關聯的 Pod 。

   - **resourceFilter.resourceSelectionCriteria** ：（篩選所需）使用 `Include`或包含或 `Exclude`排除在 resourceMatchers 中定義的資源。新增下列資源配置工具參數、以定義要納入或排除的資源：
     - **resourceFilter.resourceMatchers** ：一組 resourceMatcher 物件。如果您在此陣列中定義多

個元素，它們會比對為 OR 作業，而每個元素（群組，種類，版本）內的欄位會比對為 AND 作業。

- **resourceMatchers[].group** ：（ *Optional* ）要篩選的資源群組。
- **resourceMatchers[].cher** ：（ *Optional* ）要篩選的資源種類。
- **resourceMatchers[].version** ：（ *Optional* ）要篩選的資源版本。
- 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）名稱。
- 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）命名空間。
- 資源的 Kubernetes metadata.name 欄位中的 *resourceMatchers[].labelSelectors * ：（ *Optional* ） Label 選取器字串，如中所定義 "Kubernetes文件"。例如 `"trident.netapp.io/os=linux"`：。

  例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-backup-restore-cr.yaml` 、請套用 CR ：

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

使用CLI

步驟

1. 將備份還原至不同的命名空間，以環境中的資訊取代括弧中的值。此 `namespace-mapping`引數使用以冒號分隔的命名空間，以格式將來源命名空間對應至正確的目的地命名空間 `source1:dest1,source2:dest2`。例如：

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

從備份還原至原始命名空間

您可以隨時將備份還原至原始命名空間。

開始之前

確保 AWS 工作階段權杖到期時間足以執行任何長時間執行的 S3 還原作業。如果 Token 在還原作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-backup-ipr-cr.yaml` 。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appArchivePath** ：儲存備份內容的 AppVault 內部路徑。您可以使用下列命令來尋找此路徑：

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

   - **spec.appVaultRef** ：（ _required _ ）儲存備份內容的 AppVault 名稱。

     例如：

```
---
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. （ *Optional* ）如果您只需要選取應用程式的某些資源來還原，請新增篩選功能，以包含或排除標記有特定標籤的資源：

   Trident Protect 會自動選取部分資源，因為這些資源與您選取的資源之間的關係。例如，如果您選取持續磁碟區宣告資源，而且該資源有相關聯的 Pod ，則 Trident Protect 也會還原相關聯的 Pod 。

   - **resourceFilter.resourceSelectionCriteria** ：（篩選所需）使用 `Include`或包含或 `Exclude`排除在 resourceMatchers 中定義的資源。新增下列資源配置工具參數、以定義要納入或排除的資源：
     - **resourceFilter.resourceMatchers** ：一組 resourceMatcher 物件。如果您在此陣列中定義多個元素，它們會比對為 OR 作業，而每個元素（群組，種類，版本）內的欄位會比對為 AND 作業。
       - **resourceMatchers[].group** ：（ *Optional* ）要篩選的資源群組。
       - **resourceMatchers[].cher** ：（ *Optional* ）要篩選的資源種類。
       - **resourceMatchers[].version** ：（ *Optional* ）要篩選的資源版本。
       - 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（

Optional ）名稱。

- 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）命名空間。
- 資源的 Kubernetes metadata.name 欄位中的 *resourceMatchers[].labelSelectors * ：（ *Optional* ） Label 選取器字串，如中所定義 "Kubernetes文件"。例如 `"trident.netapp.io/os=linux"`：。

  例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-backup-ipr-cr.yaml` 、請套用 CR ：

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

使用CLI

步驟

1. 將備份還原至原始命名空間，以環境中的資訊取代括弧中的值。 `backup`引數使用的名稱空間和備份名稱格式為 `<namespace>/<name>`。例如：

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

從備份還原至不同的叢集

如果原始叢集發生問題，您可以將備份還原至不同的叢集。

開始之前

確保符合下列先決條件：

- 目的地叢集已安裝 Trident Protect 。
- 目的地叢集可存取與儲存備份的來源叢集相同 AppVault 的儲存區路徑。
- 確保 AWS 工作階段權杖到期時間足以執行任何長時間執行的還原作業。如果 Token 在還原作業期間過期，作業可能會失敗。
  - 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
  - 如需 AWS 資源認證的詳細資訊，請參閱 "AWS 文件"。

步驟

1. 使用 Trident Protect CLI 外掛程式檢查目的地叢集上的 AppVault CR 可用度：

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

確保目的地叢集上存在用於應用程式還原的命名空間。

2. 從目的地叢集檢視可用 AppVault 的備份內容：

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

執行此命令會顯示 AppVault 中的可用備份，包括其原始叢集，對應的應用程式名稱，時間戳記和歸檔路徑。

- 輸出範例： *

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   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. 使用 AppVault 名稱和歸檔路徑將應用程式還原至目的地叢集：

使用 **CR**

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-backup-restore-cr.yaml` 。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appVaultRef** ：（ _required _ ）儲存備份內容的 AppVault 名稱。
   - **spec.appArchivePath** ：儲存備份內容的 AppVault 內部路徑。您可以使用下列命令來尋找此路徑：

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

     如果無法使用 BackupRestore CR ，您可以使用步驟 2 所述的命令來檢視備份內容。

   - **spec.namespaceMapping**: 將還原作業的來源命名空間對應至目的地命名空間。以環境中的資訊取代 `my-source-namespace`和 `my-destination-namespace` 。

     例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-backup-restore-cr.yaml` 、請套用 CR ：

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

使用**CLI**

1. 使用下列命令還原應用程式，將方括號中的值取代為您環境中的資訊。命名空間對應引數使用以冒號分隔的命名空間，將來源命名空間對應到正確的目的地命名空間，格式為 source1:dest1 ，source2:dest2 。例如：

```
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>
```

從快照還原至不同的命名空間

您可以使用自訂資源（ CR ）檔案、將資料從快照還原至不同的命名空間或原始來源命名空間。當您使用 SnapshotRestore CR 將快照還原至不同的命名空間時， Trident Protect 會在新命名空間中還原應用程式，並為還原的應用程式建立應用程式 CR 。若要保護還原的應用程式，請建立隨需備份或快照，或建立保護排程。

開始之前

確保 AWS 工作階段權杖到期時間足以執行任何長時間執行的 S3 還原作業。如果 Token 在還原作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appVaultRef** ：（ _required _ ）儲存快照內容的 AppVault 名稱。
   - **spec.appArchivePath** ：在 AppVault 中儲存快照內容的路徑。您可以使用下列命令來尋找此路徑：

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

   - **spec.namespaceMapping**: 將還原作業的來源命名空間對應至目的地命名空間。以環境中的資訊取代 `my-source-namespace`和 `my-destination-namespace`。
   - **spec.storageClassMapping** ：將還原作業的來源儲存類別對應至目的地儲存類別。以環境中的資訊取代 `destinationStorageClass`和 `sourceStorageClass`。

     ```
     ---
     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. （ *Optional* ）如果您只需要選取應用程式的某些資源來還原，請新增篩選功能，以包含或排除標記有特定標籤的資源：

   Trident Protect 會自動選取部分資源，因為這些資源與您選取的資源之間的關係。例如，如果您選取持續磁碟區宣告資源，而且該資源有相關聯的 Pod ，則 Trident Protect 也會還原相關聯的 Pod 。

   - **resourceFilter.resourceSelectionCriteria** ：（篩選所需）使用 `Include`或包含或 `Exclude`排除在 resourceMatchers 中定義的資源。新增下列資源配置工具參數、以定義要納入或排除的資源：
     - **resourceFilter.resourceMatchers** ：一組 resourceMatcher 物件。如果您在此陣列中定義多

個元素，它們會比對為 OR 作業，而每個元素（群組，種類，版本）內的欄位會比對為 AND 作業。

- **resourceMatchers[].group** ：（ *Optional* ）要篩選的資源群組。
- **resourceMatchers[].cher** ：（ *Optional* ）要篩選的資源種類。
- **resourceMatchers[].version** ：（ *Optional* ）要篩選的資源版本。
- 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）名稱。
- 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）命名空間。
- 資源的 Kubernetes metadata.name 欄位中的 *resourceMatchers[].labelSelectors * ：（ *Optional* ） Label 選取器字串，如中所定義 "Kubernetes文件"。例如 `"trident.netapp.io/os=linux"`：。

  例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-snapshot-restore-cr.yaml` 、請套用 CR ：

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

使用CLI

步驟

1. 將快照還原至不同的命名空間，以環境中的資訊取代方括號中的值。
   - `snapshot`引數使用格式的命名空間和快照名稱 `<namespace>/<name>`。
   - 此 `namespace-mapping`引數使用以冒號分隔的命名空間，以格式將來源命名空間對應至正確的

目的地命名空間 `source1:dest1,source2:dest2。

例如：

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

從快照還原至原始命名空間

您可以隨時將快照還原至原始命名空間。

開始之前

確保 AWS 工作階段權杖到期時間足以執行任何長時間執行的 S3 還原作業。如果 Token 在還原作業期間過期，作業可能會失敗。

- 如需檢查目前工作階段權杖到期時間的詳細資訊，請參閱 "AWS API 文件"。
- 如需 AWS 資源認證的詳細資訊，請參閱 "AWS IAM 文件"。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-snapshot-ipr-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appVaultRef** ：（ _required _ ）儲存快照內容的 AppVault 名稱。
   - **spec.appArchivePath** ：在 AppVault 中儲存快照內容的路徑。您可以使用下列命令來尋找此路徑：

```
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. （ *Optional* ）如果您只需要選取應用程式的某些資源來還原，請新增篩選功能，以包含或排除標記有特定標籤的資源：

> Trident Protect 會自動選取部分資源，因為這些資源與您選取的資源之間的關係。例如，如果您選取持續磁碟區宣告資源，而且該資源有相關聯的 Pod ，則 Trident Protect 也會還原相關聯的 Pod 。

   - **resourceFilter.resourceSelectionCriteria** ：（篩選所需）使用 `Include`或包含或 `Exclude`排除在 resourceMatchers 中定義的資源。新增下列資源配置工具參數、以定義要納入或排除的資源：
     - **resourceFilter.resourceMatchers** ：一組 resourceMatcher 物件。如果您在此陣列中定義多個元素，它們會比對為 OR 作業，而每個元素（群組，種類，版本）內的欄位會比對為 AND 作業。
       - **resourceMatchers[].group** ：（ *Optional* ）要篩選的資源群組。
       - **resourceMatchers[].cher** ：（ *Optional* ）要篩選的資源種類。
       - **resourceMatchers[].version** ：（ *Optional* ）要篩選的資源版本。
       - 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）名稱。
       - 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）命名空間。

- 資源的 Kubernetes metadata.name 欄位中的 *resourceMatchers[].labelSelectors * ：（ *Optional* ） Label 選取器字串，如中所定義 "Kubernetes文件"。例如 `"trident.netapp.io/os=linux"`：。

  例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-snapshot-ipr-cr.yaml` 、請套用 CR ：

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

使用CLI

步驟

1. 將快照還原至原始命名空間，以環境中的資訊取代方括號中的值。例如：

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

檢查還原作業的狀態

您可以使用命令列來檢查進行中，已完成或已失敗的還原作業狀態。

步驟

1. 使用下列命令可擷取還原作業的狀態，以環境中的資訊取代方括號中的值：

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

使用 NetApp SnapMirror 和 Trident Protect 複寫應用程式

使用 Trident Protect ，您可以使用 NetApp SnapMirror 技術的非同步複寫功能，將資料和應用程式變更從一個儲存後端複寫到另一個儲存後端，在同一個叢集或不同叢集之間複寫。

還原和容錯移轉作業期間的命名空間註釋和標籤

在還原和容錯移轉作業期間，目的地命名空間中的標籤和註釋會與來源命名空間中的標籤和註釋相符。會新增來源命名空間中不存在的標籤或註釋，並覆寫已存在的任何標籤或註釋，以符合來源命名空間的值。只存在於目的地命名空間上的標籤或註釋會保持不變。

> ⓘ 如果您使用 RedHat OpenShift ，請務必注意 OpenShift 環境中命名空間註釋的關鍵作用。命名空間註釋可確保還原的 Pod 符合 OpenShift 安全性內容限制（ SCC ）所定義的適當權限和安全性組態，而且無需權限問題即可存取磁碟區。如需詳細資訊，請 "OpenShift 安全性內容限制文件"參閱。

您可以在執行還原或容錯移轉作業之前，先設定 Kubernetes 環境變數，以避免覆寫目的地命名空間中的特定註釋 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。例如：

```
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>
```

如果您使用帶有標誌的 Helm 來安裝來源應用程式 `--create-namespace`，則會對標籤機碼給予特殊處理 `name`。在還原或容錯移轉程序期間， Trident Protect 會將此標籤複製到目的地命名空間，但如果來源的值符合來源命名空間，則會將值更新到目的地命名空間值。如果此值與來源命名空間不相符，則會將其複製到目的地命名空間，而不會有任何變更。

範例

以下範例提供來源和目的地命名空間，每個命名空間都有不同的註釋和標籤。您可以查看作業前後目的地命名空間的狀態，以及註釋和標籤在目的地命名空間中的組合或覆寫方式。

還原或容錯移轉作業之前

下表說明還原或容錯移轉作業之前的範例來源和目的地命名空間狀態：

命名空間	註釋	標籤
命名空間 nS-1 （來源）	• annotation.one / 機碼：「 updatedvalue 」 • annotation.b2/key ：「 true 」	• 環境 = 正式作業 • Compliance = HIPAA • NAME=ns-1
命名空間 nS-2 （目的地）	• annotation.one / 機碼：「 true 」 • annotation.the/key ：「 FALSE 」	• role = 資料庫

還原作業之後

下表說明還原或容錯移轉作業之後範例目的地命名空間的狀態。某些金鑰已新增，部分已覆寫， `name`標籤已更新以符合目的地命名空間：

命名空間	註釋	標籤
命名空間 nS-2 （目的地）	• annotation.one / 機碼：「 updatedvalue 」 • annotation.b2/key ：「 true 」 • annotation.the/key ：「 FALSE 」	• NAME=nS-2 • Compliance = HIPAA • 環境 = 正式作業 • role = 資料庫

您可以將 Trident Protect 設定為在資料保護作業期間凍結和取消凍結檔案系統。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。

設定複寫關係

設定複寫關係涉及下列事項：

- 選擇 Trident Protect 拍攝應用程式快照的頻率（包括應用程式的 Kubernetes 資源，以及每個應用程式磁碟區的磁碟區快照）
- 選擇複寫排程（包括 Kubernetes 資源及持續磁碟區資料）
- 設定拍攝快照的時間

步驟

1. 為來源叢集上的來源應用程式建立 AppVault 。視您的儲存供應商而定，請修改中的範例"AppVault 自訂資源"以符合您的環境：

使用 **CR** 建立 **AppVault**

a. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-appvault-primary-source.yaml`）。

b. 設定下列屬性：

- * metadata.name*: （ _required ） AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 CR 檔案會參照此值。
- * spec.providerConfig*: （ _required ）儲存使用指定供應商存取 AppVault 所需的組態。請為您的供應商選擇一個「鎖釦名稱」和任何其他必要的詳細資料。請記下您選擇的值，因為複寫關係所需的其他 CR 檔案會參照這些值。如需 AppVault CRS 與其他供應商的範例，請參閱"AppVault 自訂資源"。
- * spec.providerCredentials*: （ _required _ ）會儲存使用指定提供者存取 AppVault 所需之任何認證的參考資料。
  - * spec.providerCredentials.valueFromSecret*: （ _required _ ）表示認證值應來自機密。
    - **key** ：（ _required _ ）要從中選擇的密碼的有效金鑰。
    - * 名稱 * ：（ _ 必要 _ ）包含此欄位值的機密名稱。必須位於相同的命名空間中。
  - * spec.providerCredentials.secretAccessKey*: （ _required _ ）存取提供者所用的存取金鑰。* 名稱 * 應與 * spec.providerCredentials.valueFromSecret.name* 相符。
- * spec.providerType*: （ _required _ ）決定提供備份的內容，例如 NetApp ONTAP S3 ，一般 S3 ， Google Cloud 或 Microsoft Azure 。可能值：
  - AWS
  - Azure
  - GCP
  - generic-S3
  - ONTAP S3
  - StorageGRID S3

c. 在您以正確的值填入檔案之後 `trident-protect-appvault-primary-source.yaml` 、請套用 CR ：

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

使用 **CLI** 建立 **AppVault**

a. 建立 AppVault ，以環境資訊取代方括號中的值：

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

2. 建立來源應用程式 CR ：

使用 **CR** 建立來源應用程式

a. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-app-source.yaml`）。

b. 設定下列屬性：

- * metadata.name*: （ _required ）應用程式自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 CR 檔案會參照此值。
- * spec.includedNamespaces*: （ _required ）一組命名空間和相關標籤。使用命名空間名稱，並選擇性地使用標籤來縮小命名空間的範圍，以指定此處列出的命名空間中存在的資源。應用程式命名空間必須是此陣列的一部分。
  - YAML* 範例：

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-app-source.yaml` 、請套用 CR ：

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

使用 **CLI** 建立來源應用程式

a. 建立來源應用程式。例如：

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

3. 您也可以選擇拍攝來源應用程式的快照。此快照是作為目的地叢集上應用程式的基礎。如果您略過此步驟，則需要等待下一個排定的快照執行，以便擁有最近的快照。

使用 **CR** 拍攝快照

a. 建立來源應用程式的複寫排程：

i. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-schedule.yaml`）。

ii. 設定下列屬性：

- * metadata.name*: （ _required ）排程自訂資源的名稱。
- **spec.AppVaultRef** ：（ _required _ ）此值必須符合來源應用程式的 AppVault metadata.name 欄位。
- **spec.ApplicationRef** ：（ _required _ ）此值必須符合來源應用程式 CR 的 metadata.name 欄位。
- *spec.backupRetention * ：（ _required _ ）此欄位為必填欄位，且值必須設為 0 。
- **spec.enabled** ：必須設置爲 true 。
- * spec.granularity*: 必須設定為 `Custom`。
- **spec.recurrenceRule** ：以 UTC 時間和循環時間間隔定義開始日期。
- *spec.snapshotRetention * ：必須設定為 2 。

YAML 範例：

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-schedule.yaml` 、請套用 CR ：

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

使用 **CLI** 拍攝快照

a. 建立快照，以您環境的資訊取代方括號中的值。例如：

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

4. 在目的地叢集上建立與您在來源叢集上套用的 AppVault CR 相同的來源應用程式 AppVault CR ，並命名該應用程式（例如 `trident-protect-appvault-primary-destination.yaml`）。

5. 套用 CR ：

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

6. 為目的地叢集上的目的地應用程式建立 AppVault 。視您的儲存供應商而定，請修改中的範例"AppVault 自訂資源"以符合您的環境：

   a. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-appvault-secondary-destination.yaml`）。

   b. 設定下列屬性：

      - * metadata.name*: （ _required ） AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 CR 檔案會參照此值。
      - * spec.providerConfig*: （ _required ）儲存使用指定供應商存取 AppVault 所需的組態。請為您的供應商選擇 `bucketName`和任何其他必要詳細資料。請記下您選擇的值，因為複寫關係所需的其他 CR 檔案會參照這些值。如需 AppVault CRS 與其他供應商的範例，請參閱"AppVault 自訂資源"。
      - * spec.providerCredentials*: （ _required _ ）會儲存使用指定提供者存取 AppVault 所需之任何認證的參考資料。
        - * spec.providerCredentials.valueFromSecret*: （ _required _ ）表示認證值應來自機密。
          - **key** ：（ _required _ ）要從中選擇的密碼的有效金鑰。
          - * 名稱 * ：（ _ 必要 _ ）包含此欄位值的機密名稱。必須位於相同的命名空間中。
        - * spec.providerCredentials.secretAccessKey*: （ _required _ ）存取提供者所用的存取金鑰。* 名稱 * 應與 * spec.providerCredentials.valueFromSecret.name* 相符。
      - * spec.providerType*: （ _required _ ）決定提供備份的內容，例如 NetApp ONTAP S3 ，一般 S3 ， Google Cloud 或 Microsoft Azure 。可能值：
        - AWS
        - Azure
        - GCP
        - generic-S3
        - ONTAP S3

- StorageGRID S3

c. 在您以正確的值填入檔案之後 `trident-protect-appvault-secondary-destination.yaml` 、請套用 CR ：

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

7. 建立 AppMirrorRelationship CR 檔案：

使用 **CR** 建立 **AppMirrorRelationship**

a. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-relationship.yaml`）。

b. 設定下列屬性：

- * metadata.name:* （必要） AppMirrorRelationship 自訂資源的名稱。
- * spec.destinationAppVaultRef*: （ _required _ ）此值必須符合目的地叢集上目的地應用程式的 AppVault 名稱。
- * spec.namespaceMapping*: （ _required _ ）目的地和來源命名空間必須符合各自應用程式 CR 中定義的應用程式命名空間。
- **spec.sourceAppVaultRef** ：（ _required _ ）此值必須符合來源應用程式的 AppVault 名稱。
- **spec.sourceApplicationName** ：（ _required _ ）此值必須符合您在來源應用程式 CR 中定義的來源應用程式名稱。
- **spec.storageClassName** ：（ _required _ ）選擇叢集上有效儲存類別的名稱。儲存類別必須連結至與來源環境對等的 ONTAP 儲存 VM 。
- **spec.recurrenceRule** ：以 UTC 時間和循環時間間隔定義開始日期。

YAML 範例：

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-relationship.yaml` 、請套用 CR ：

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

使用 **CLI** 建立 **AppMirrorRelationship**

a. 建立並套用 AppMirrorRelationship 物件，以環境資訊取代方括號中的值。例如：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --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. （ *Optional* ）檢查複寫關係的狀態和狀態：

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

容錯移轉至目的地叢集

使用 Trident Protect ，您可以將複寫的應用程式容錯移轉至目的地叢集。此程序會停止複寫關係、並在目的地叢集上使應用程式上線。如果來源叢集上的應用程式正常運作， Trident Protect 不會停止該應用程式。

步驟

1. 開啟 AppMirrorRelationship CR 檔案（例如 `trident-protect-relationship.yaml`），並將 * spec.desiredState* 的值變更為 `Promoted`。
2. 儲存 CR 檔案。
3. 套用 CR ：

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

4. （ *Optional* ）在容錯移轉應用程式上建立所需的任何保護排程。
5. （ *Optional* ）檢查複寫關係的狀態和狀態：

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

重新同步容錯移轉複寫關係

重新同步作業會重新建立複寫關係。執行重新同步作業後，原始來源應用程式即成為執行中的應用程式，而對目的地叢集上執行中的應用程式所做的任何變更都會被捨棄。

此程序會在重新建立複寫之前，停止目的地叢集上的應用程式。

> 在容錯移轉期間寫入目的地應用程式的任何資料都會遺失。

步驟

1. 建立來源應用程式的快照。
2. 打開 AppMirrorRelationship CR 文件（例如 `trident-protect-relationship.yaml`），然後將 spec.desiredState 的值更改爲 `Established`。
3. 儲存 CR 檔案。
4. 套用 CR ：

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

5. 如果您在目的地叢集上建立任何保護排程來保護容錯移轉應用程式，請將其移除。任何仍會導致磁碟區快照失敗的排程。

反轉重新同步容錯移轉複寫關係

當您反向重新同步容錯移轉複寫關係時，目的地應用程式會變成來源應用程式，來源會變成目的地。在容錯移轉期間對目的地應用程式所做的變更會保留下來。

步驟

1. 刪除原始目的地叢集上的 AppMirrorRelationship CR 。這會導致目的地成為來源。如果新的目的地叢集上還有任何保護排程，請將其移除。
2. 套用原先用來設定與相對叢集關係的 CR 檔案，以設定複寫關係。
3. 確保每個叢集上的 AppVault CRS 均已就緒。
4. 在相對的叢集上設定複寫關係，設定反轉方向的值。

反轉應用程式複寫方向

當您反轉複寫方向時， Trident Protect 會將應用程式移至目的地儲存後端，同時繼續複寫回原始來源儲存後端。Trident Protect 會停止來源應用程式，並在容錯移轉至目的地應用程式之前，將資料複寫到目的地。

在這種情況下、您要交換來源和目的地。

步驟

1. 建立關機快照：

使用 **CR** 建立關機快照

a. 停用來源應用程式的保護原則排程。

b. 建立 ShutdownSnapshot CR 檔案：

i. 建立自訂資源（ CR ）檔案並命名（例如 `trident-protect-shutdownsnapshot.yaml` ）。

ii. 設定下列屬性：

- * metadata.name*: （ _required ）自訂資源的名稱。
- **spec.AppVaultRef** ：（ _required _ ）此值必須符合來源應用程式的 AppVault metadata.name 欄位。
- **spec.ApplicationRef** ：（ _required _ ）此值必須符合來源應用程式 CR 檔案的 metadata.name 欄位。

YAML 範例：

```
---
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. 在您以正確的值填入檔案之後 `trident-protect-shutdownsnapshot.yaml` 、請套用 CR ：

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

使用 **CLI** 建立關機快照

a. 建立關機快照，以環境資訊取代方括號中的值。例如：

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

2. 快照完成後，取得快照的狀態：

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

3. 使用下列命令尋找 * shutdownsnapshot .status.appArchivePath* 的值，並記錄檔案路徑的最後一部分（也稱為 basename ；這將是最後一個斜線之後的所有項目）：

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

4. 執行容錯移轉，從目的地叢集移轉至來源叢集，並進行下列變更：

> 在容錯移轉程序的步驟 2 中，將欄位包含在 `spec.promotedSnapshot` AppMirrorRelationship CR 檔案中，並將其值設為您在上述步驟 3 中記錄的基礎名稱。

5. 執行中的反向重新同步步驟[反轉重新同步容錯移轉複寫關係]。
6. 在新的來源叢集上啟用保護排程。

結果

由於反向複寫，因此會發生下列動作：

- 原始來源應用程式的 Kubernetes 資源會擷取快照。
- 刪除應用程式的Kubernetes資源（保留PVCS和PVs）、即可順利停止原始來源應用程式的Pod。
- 當 Pod 關機之後、應用程式的磁碟區快照就會被擷取和複寫。
- SnapMirror關係中斷、使目的地磁碟區準備好進行讀寫。
- 應用程式的 Kubernetes 資源會從關機前快照還原、並使用原始來源應用程式關機後複寫的 Volume 資料。
- 複寫會以相反方向重新建立。

將應用程式容錯移轉至原始來源叢集

使用 Trident Protect ，您可以使用下列作業順序，在容錯移轉作業之後達成「容錯回復」。在此工作流程中，為了還原原始複寫方向， Trident Protect 會在還原複寫方向之前，將任何應用程式變更複寫回原始來源應用程式。

此程序從已完成容錯移轉至目的地的關係開始、並涉及下列步驟：

- 從容錯移轉狀態開始。
- 反向重新同步複寫關係。

> 請勿執行正常的重新同步作業，因為這會捨棄在容錯移轉程序期間寫入目的地叢集的資料。

- 反轉複寫方向。

步驟

1. 執行[反轉重新同步容錯移轉複寫關係]步驟。

2. 執行[反轉應用程式複寫方向]步驟。

刪除複寫關係

您可以隨時刪除複寫關係。當您刪除應用程式複寫關係時，會產生兩個獨立的應用程式，兩者之間沒有任何關係。

步驟

1. 刪除 AppMirrorRelationship CR ：

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

使用 **Trident Protect** 移轉應用程式

您可以將備份或快照資料還原至不同的叢集或儲存類別，在叢集或儲存類別之間移轉應用程式。

當您移轉應用程式時，為應用程式設定的所有執行掛鉤都會隨應用程式一起移轉。如果存在還原後執行掛鉤，則會在還原作業中自動執行。

備份與還原作業

若要針對下列案例執行備份與還原作業，您可以自動化特定的備份與還原工作。

複製到同一個叢集

若要將應用程式複製到同一個叢集，請建立快照或備份，然後將資料還原到同一個叢集。

步驟

1. 執行下列其中一項：
   a. "建立快照"。
   b. "建立備份"。
2. 在同一個叢集上，視您建立的是快照或備份而定，請執行下列其中一項：
   a. "從快照還原資料"。
   b. "從備份還原資料"。

複製到不同叢集

若要將應用程式複製到不同的叢集（執行跨叢集複製），請在來源叢集上建立備份，然後將備份還原到不同的叢集。請確定目的地叢集上已安裝 Trident Protect 。

您可以使用在不同叢集之間複寫應用程式"SnapMirror 複寫"。

步驟

1. "建立備份"。

2. 請確定已在目的地叢集上設定包含備份之物件儲存貯體的 AppVault CR 。
3. 在目的地叢集上"從備份還原資料"，。

將應用程式從一個儲存類別移轉至另一個儲存類別

您可以將快照還原至不同的目的地儲存類別，將應用程式從一個儲存類別移轉至不同的儲存類別。

例如（從還原 CR 中排除機密）：

```
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
```

使用 **CR** 還原快照

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的檔案中，設定下列屬性：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.appArchivePath** ：在 AppVault 中儲存快照內容的路徑。您可以使用下列命令來尋找此路徑：

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

   - **spec.appVaultRef** ：（ _required _ ）儲存快照內容的 AppVault 名稱。
   - **spec.namespaceMapping**: 將還原作業的來源命名空間對應至目的地命名空間。以環境中的資訊取代 `my-source-namespace`和 `my-destination-namespace`。

     ```
     ---
     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. 或者，如果您只需要選取要還原的應用程式特定資源，請新增篩選功能，以包含或排除標記有特定標籤的資源：
   - *resourceFilter.resourceSelectionCriteria ：（篩選所需）用於 `include or exclude`包含或排除在 resourceMatchers 中定義的資源。新增下列資源配置工具參數、以定義要納入或排除的資源：
     - **resourceFilter.resourceMatchers** ：一組 resourceMatcher 物件。如果您在此陣列中定義多個元素，它們會比對為 OR 作業，而每個元素（群組，種類，版本）內的欄位會比對為 AND 作業。
       - **resourceMatchers[].group** ：（ *Optional* ）要篩選的資源群組。
       - **resourceMatchers[].cher** ：（ *Optional* ）要篩選的資源種類。
       - **resourceMatchers[].version** ：（ *Optional* ）要篩選的資源版本。
       - 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（ *Optional* ）名稱。
       - 要篩選之資源的 Kubernetes metadata.name 欄位中的 * resourceMatchers[].names* ：（

Optional ）命名空間。

- 資源的 Kubernetes metadata.name 欄位中的 *resourceMatchers[].labelSelectors * ：（ *Optional* ） Label 選取器字串，如中所定義 "Kubernetes文件"。例如 `"trident.netapp.io/os=linux"`：。

  例如：

```
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. 在您以正確的值填入檔案之後 `trident-protect-snapshot-restore-cr.yaml` 、請套用 CR ：

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

使用 **CLI** 還原快照

步驟

1. 將快照還原至不同的命名空間，以環境中的資訊取代方括號中的值。

   ◦ `snapshot`引數使用格式的命名空間和快照名稱 `<namespace>/<name>`。

   ◦ 此 `namespace-mapping`引數使用以冒號分隔的命名空間，以格式將來源命名空間對應至正確的目的地命名空間 `source1:dest1,source2:dest2`。

   例如：

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

管理 Trident Protect 執行攔截器

執行攔截是一種自訂動作、可設定搭配託管應用程式的資料保護作業一起執行。例如、如果您有資料庫應用程式、您可以使用執行掛勾來暫停快照之前的所有資料庫交易、並在快照完成後繼續交易。如此可確保應用程式一致的快照。

執行掛勾的類型

Trident Protect 支援下列類型的執行掛鉤，視執行時機而定：

- 快照前
- 快照後
- 預先備份
- 備份後
- 還原後
- 容錯移轉後

執行順序

執行資料保護作業時、執行掛機事件會依照下列順序發生：

1. 任何適用的自訂操作前執行掛勾都會在適當的容器上執行。您可以視需要建立及執行任意數量的自訂操作前掛勾、但在作業之前執行這些掛勾的順序既不保證也無法設定。
2. 如果適用，檔案系統會凍結。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。
3. 執行資料保護作業。
4. 凍結的檔案系統會在適用的情況下解除凍結。
5. 任何適用的自訂操作後執行掛勾都會在適當的容器上執行。您可以視需要建立及執行任意數量的自訂後置作業掛勾、但在作業後執行這些掛勾的順序並不保證也無法設定。

如果您建立同一類型的多個執行掛勾（例如預先快照）、則無法保證這些掛勾的執行順序。不過、不同類型的掛勾的執行順序也有保證。例如，以下是具有所有不同類型勾點的組態執行順序：

1. 執行快照前掛勾
2. 快照後掛勾已執行
3. 執行備份前掛勾
4. 執行備份後掛勾

上述順序範例僅適用於執行不使用現有快照的備份時。

在正式作業環境中啟用執行攔截指令碼之前、請務必先進行測試。您可以使用'kubecl exec'命令來方便地測試指令碼。在正式作業環境中啟用執行掛勾之後、請測試所產生的快照和備份、以確保它們一致。您可以將應用程式複製到暫用命名空間、還原快照或備份、然後測試應用程式、藉此完成此作業。

如果快照前執行攔截器新增，變更或移除 Kubernetes 資源，則這些變更會包含在快照或備份中，以及任何後續還原作業中。

關於自訂執行掛勾的重要注意事項

規劃應用程式的執行掛勾時、請考量下列事項。

- 執行攔截必須使用指令碼來執行動作。許多執行掛勾可以參照相同的指令碼。
- Trident Protect 需要以可執行的 Shell 指令碼格式寫入執行攔截程式所使用的指令碼。
- 指令碼大小上限為96KB。
- Trident Protect 使用執行掛鉤設定和任何符合條件，來判斷哪些掛勾適用於快照，備份或還原作業。

由於執行掛勾通常會減少或完全停用執行中應用程式的功能、因此您應該一律盡量縮短自訂執行掛勾執行所需的時間。如果您以相關的執行掛勾開始備份或快照作業、但隨後取消它、則如果備份或快照作業已經開始、仍允許掛勾執行。這表示備份後執行掛勾中使用的邏輯無法假設備份已完成。

執行攔截篩選器

當您新增或編輯應用程式的執行掛鉤時，您可以將篩選器新增至執行掛鉤，以管理掛鉤將符合的容器。篩選器對於在所有容器上使用相同容器映像的應用程式來說非常實用、但可能會將每個映像用於不同的用途（例如Elasticsearch）。篩選器可讓您建立執行攔截器在某些容器上執行的案例、但不一定所有容器都相同。如果您為單一執行掛勾建立多個篩選器、這些篩選器會與邏輯和運算子結合使用。每個執行掛機最多可有10個作用中篩選器。

您新增至執行掛勾的每個篩選器都會使用規則運算式來比對叢集中的容器。當掛機符合容器時、掛機會在該容器上執行其相關的指令碼。篩選器的規則運算式使用規則運算式2（RE2）語法、不支援建立篩選器、將容器從相符項目清單中排除。如需 Trident Protect 支援執行攔截篩選器中規則運算式的語法資訊，請參閱 "規則運算式2（RE2）語法支援"。

如果您將命名空間篩選器新增至執行掛勾、而執行還原或複製作業之後執行、且還原或複製來源與目的地位於不同的命名空間、則命名空間篩選器只會套用至目的地命名空間。

執行攔截範例

請造訪 "NetApp Verda GitHub專案" 下載熱門應用程式（例如 Apache Cassandra 和 Elasticsearch ）的實際執行連結。您也可以查看範例、瞭解如何建構您自己的自訂執行掛勾。

建立執行掛鉤

您可以使用 Trident Protect 為應用程式建立自訂執行掛鉤。您需要擁有擁有擁有者、管理員或成員權限、才能建立執行掛勾。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-hook.yaml`。
2. 設定下列屬性以符合 Trident Protect 環境和叢集組態：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **SPEC.applicationRef** ：（ _required _ ）要執行執行攔截的應用程式 Kubernetes 名稱。
   - *spec.Stage * ：（ _required _ ）一個字串，指出執行掛鉤應在動作期間執行的階段。可能值：
     - 準備
     - 貼文
   - **spec.ACTION** ：（ _required _ ）字串，表示執行攔截將採取的行動，前提是指定的任何執行攔截篩選條件都已相符。可能值：
     - Snapshot
     - 備份
     - 還原
     - 容錯移轉
   - **spec.enabled** ：（ *Optional* ）表示此執行掛鉤是否已啟用或停用。如果未指定，則預設值為 true 。
   - **spec.hookSource** ：（ _required _ ）包含 base64 編碼 hook 指令碼的字串。
   - **spec.timeout** ：（ *Optional* ）一個數字，定義允許執行掛鉤執行的時間（以分鐘為單位）。最小值為 1 分鐘，如果未指定，預設值為 25 分鐘。
   - **spec.arguments** ：（ *Optional* ） YAML 引數清單，您可以為執行攔截器指定。
   - *spec.mmatchingCriteria ：（ *Optional* ）選擇性的條件金鑰值配對清單，每個配對組成執行掛機篩選器。每個執行掛鉤最多可新增 10 個篩選器。
   - **spec.matchingCriteria.type** ：（ *Optional* ）識別執行掛鉤篩選器類型的字串。可能值：
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value** ：（ *Optional* ）識別執行掛鉤篩選值的字串或規則運算式。

     YAML 範例：

```
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. 在您以正確的值填入 CR 檔案之後，請套用 CR ：

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

使用CLI

步驟

1. 建立執行掛鉤，以環境資訊取代方括號中的值。例如：

```
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>
```

手動執行掛鉤

您可以手動執行掛鉤以進行測試，或是在故障後需要手動重新執行掛鉤。您需要擁有擁有者，管理員或成員權限，才能手動執行掛鉤。

手動執行掛鉤包含兩個基本步驟：

1. 建立資源備份，收集資源並建立資源備份，以判斷攔截的執行位置
2. 在備份上執行執行掛鉤

步驟 **1** ：建立資源備份

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-resource-backup.yaml`。
2. 設定下列屬性以符合 Trident Protect 環境和叢集組態：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **spec.applicationRef** ：（ _required _ ）要建立資源備份的應用程式 Kubernetes 名稱。
   - **spec.appVaultRef** ：（ _required _ ）儲存備份內容的 AppVault 名稱。
   - **spec.appArchivePath** ：儲存備份內容的 AppVault 內部路徑。您可以使用下列命令來尋找此路徑：

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

     YAML 範例：

```
---
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. 在您以正確的值填入 CR 檔案之後，請套用 CR ：

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

使用**CLI**

步驟

1. 建立備份，以您環境的資訊取代括號中的值。例如：

```
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. 檢視備份狀態。您可以重複使用此範例命令，直到作業完成為止：

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

3. 確認備份成功：

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

步驟 **2** ：執行掛鉤

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-hook-run.yaml` 。
2. 設定下列屬性以符合 Trident Protect 環境和叢集組態：
   - * metadata.name*: （ _required ）此自訂資源的名稱；為您的環境選擇唯一且合理的名稱。
   - **SPEC.applicationRef** ：（ _required _ ）請確保此值符合您在步驟 1 中建立的 ResourceBackup CR 應用程式名稱。
   - **spec.appVaultRef** ：（ _required _ ）請確保此值符合您在步驟 1 中建立的 ResourceBackup CR 的 apVaultRef 。
   - **spec.appArchivePath** ：確保此值與您在步驟 1 中建立的 ResourceBackup CR 中的 appArchivePath 相符。

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

   - **spec.ACTION** ：（ _required _ ）字串，表示執行攔截將採取的行動，前提是指定的任何執行攔截篩選條件都已相符。可能值：
     - Snapshot
     - 備份
     - 還原
     - 容錯移轉
   - *spec.Stage * ：（ _required _ ）一個字串，指出執行掛鉤應在動作期間執行的階段。此掛鉤掃描不會在任何其他階段執行掛鉤。可能值：
     - 準備
     - 貼文

       YAML 範例：

```
---
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. 在您以正確的值填入 CR 檔案之後，請套用 CR ：

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

使用**CLI**

步驟

1. 建立手動執行攔截執行要求：

```
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. 檢查執行攔截執行的狀態。您可以重複執行此命令，直到作業完成為止：

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

3. 說明 exechooksrun 物件以查看最終詳細資料和狀態：

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

解除安裝 Trident Protect

如果您要從試用版升級至完整版產品，則可能需要移除 Trident Protect 元件。

若要移除 Trident Protect ，請執行下列步驟。

步驟

1. 移除 Trident Protect CR 檔案：

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

2. 移除 Trident Protect ：

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

3. 移除 Trident Protect 命名空間：

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