使用Trident Protect 保護應用程式

Trident

NetApp
February 02, 2026

目錄

使用Trident Protect 保護應用程式

了解Trident Protect

NetApp Trident Protect 提供進階應用程式資料管理功能，增強了由NetApp ONTAP儲存系統和NetApp Trident CSI 儲存供應器支援的有狀態 Kubernetes 應用程式的功能和可用性。Trident Protect 簡化了跨公有雲和本地環境的容器化工作負載的管理、保護和遷移。它還透過其 API 和 CLI 提供自動化功能。

您可以透過建立自訂資源 (CR) 或使用Trident Protect CLI 來使用Trident Protect 保護應用程式。

接下來呢？

您可以先了解Trident Protect 的相關需求，然後再進行安裝：

- "Trident保護要求"

安裝Trident Protect

Trident保護要求

首先，請驗證您的運行環境、應用程式叢集、應用程式和授權是否已準備就緒。確保您的環境符合部署和執行Trident Protect 的這些要求。

Trident Protect Kubernetes 叢集相容性

Trident Protect 與各種完全託管和自架的 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

- Trident Protect備份僅支援Linux運算節點。Windows 運算節點不支援備份作業。
- 確保安裝Trident Protect 的叢集已配置正在執行中的快照控制器和相關的 CRD。若要安裝快照控制器，請參閱 "這些指示"。
- 確保至少存在一個 VolumeSnapshotClass。更多信息，請參閱"Volume SnapshotClass"。

Trident Protect 儲存後端相容性

Trident Protect 支援以下儲存後端：

- 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：

- 確保連接到叢集的 NetApp 儲存裝置使用 Trident 24.02 或更新版本（建議使用 Trident 24.10）。
- 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。
- 建立您計劃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了不存在的命名空間，則操作將會失敗。

NAS 經濟容量需求

Trident Protect 支援對 nas-economy 磁碟區進行備份和復原作業。目前不支援將快照、克隆和SnapMirror複製到 nas-economy 磁碟區。您需要為計劃與Trident Protect 一起使用的每個 nas-economy 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

Trident Protect 在資料保護作業期間為 KubeVirt 虛擬機器提供檔案系統凍結和解凍功能，以確保資料一致性。虛擬機器凍結操作的配置方法和預設行為在Trident Protect 的不同版本中有所不同，較新的版本透過 Helm chart 參數提供了簡化的配置。

在恢復操作期間，任何 `VirtualMachineSnapshots`為虛擬機器 (VM) 所建立的資料不會被復原。

Trident Protect 25.10 及更新版本

Trident Protect 在資料保護作業期間自動凍結和解凍 KubeVirt 檔案系統，以確保一致性。從Trident Protect 25.10 開始，您可以使用以下方法停用此行為： `vm.freeze` Helm Chart 安裝過程中的參數。此參數預設啟用。

```
helm install ... --set vm.freeze=false ...
```

Trident Protect 24.10.1 至 25.06

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解凍 KubeVirt 檔案系統。您也可以使用以下命令停用此自動行為：

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

Trident Protect 24.10

Trident Protect 24.10 在資料保護作業期間不會自動確保 KubeVirt VM 檔案系統的一致性狀態。如果您想使用Trident Protect 24.10 保護您的 KubeVirt VM 數據，則需要在執行資料保護作業之前手動啟用檔案系統的凍結/解凍功能。這樣可以確保檔案系統處於一致狀態。

您可以設定Trident Protect 24.10 來管理資料保護作業期間 VM 檔案系統的凍結與解凍。"設定虛擬化"然後使用以下命令：

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

SnapMirror 複寫需求

NetApp SnapMirror複製功能可與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：

- * NetApp Trident *：使用ONTAP作為後端服務的來源 Kubernetes 叢集和目標 Kubernetes 叢集上都必須存在NetApp Trident 。Trident Protect 支援使用NetApp SnapMirror技術進行複製，該技術使用以下驅動程式支

援的儲存類別：

- `ontap-nas` ：NFS
- `ontap-san` ：iSCSI
- `ontap-san` ：足球俱樂部
- `ontap-san` ：NVMe/TCP（要求最低 ONTAP 版本 9.15.1）

- * 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" 。

  從 ONTAP 9.10.1 開始、所有授權都會以 NetApp 授權檔案（ NLF ）的形式交付、這是一個可啟用多項功能的單一檔案。如需詳細資訊、請參閱 "ONTAP One 隨附授權" 。

  僅支援 SnapMirror 非同步保護。

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。

- **NetApp Trident** 與 **SVM**：對等遠端 SVM 必須可供目標叢集上的 NetApp Trident 使用。
- 託管後端：您需要在Trident Protect 中新增和管理ONTAP儲存後端，以建立複製關係。

用於 SnapMirror 複寫的 Trident / ONTAP 組態

Trident Protect 要求您至少設定一個支援來源叢集和目標叢集複製的儲存後端。如果來源叢集和目標叢集相同，為了獲得最佳彈性，目標應用程式應該使用與來源應用程式不同的儲存後端。

SnapMirror複製的 Kubernetes 叢集要求

確保您的 Kubernetes 叢集符合以下要求：

- **AppVault** 可存取性：來源叢集和目標叢集都必須具有網路存取權限，才能從 AppVault 讀取和寫入應用程式物件複製。
- 網路連線：設定防火牆規則、儲存桶權限和 IP 允許列表，以實現跨 WAN 的叢集和 AppVault 之間的通訊。

許多企業環境在 WAN 連線中實施嚴格的防火牆策略。在配置複製之前，請與您的基礎設施團隊驗證這些網路需求。

安裝並設定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，或從您自己的私人註冊表中安裝它。如果您的叢集無法存取互聯網，從私有註冊表安裝會很有幫助。

安装Trident Protect

從**NetApp**安裝**Trident Protect**

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. 使用 Helm 安裝Trident Protect。代替 `<name-of-cluster>`集群名稱將分配給集群，並用於標識集群的備份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2510.0 --create
-namespace --namespace trident-protect
```

3. （選用）若要啟用偵錯日誌記錄（建議用於故障排除），請使用：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2510.0 --create-namespace --namespace trident-protect
```

偵錯日誌記錄有助於NetApp支援人員排除故障，而無需變更日誌等級或重現問題。

從私人註冊表安裝**Trident Protect**

如果您的 Kubernetes 叢集無法存取互聯網，您可以從私人鏡像倉庫安裝Trident Protect。在這些範例中，請將括號中的值替換為您環境中的資訊：

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：

```
docker.io/netapp/controller:25.10.0
docker.io/netapp/restic:25.10.0
docker.io/netapp/kopia:25.10.0
docker.io/netapp/kopiablockrestore:25.10.0
docker.io/netapp/trident-autosupport:25.10.0
docker.io/netapp/exechook:25.10.0
docker.io/netapp/resourcebackup:25.10.0
docker.io/netapp/resourcerestore:25.10.0
docker.io/netapp/resourcedelete:25.10.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

例如：

```
docker pull docker.io/netapp/controller:25.10.0
```

```
docker tag docker.io/netapp/controller:25.10.0 <private-registry-url>/controller:25.10.0
```

```
docker push <private-registry-url>/controller:25.10.0
```

> 要取得 Helm Chart，首先需要在可以存取網路的電腦上下載 Helm Chart。`helm pull trident-protect --version 100.2510.0 --repo https://netapp.github.io/trident-protect-helm-chart`然後複製結果 `trident-protect-100.2510.0.tgz` 將檔案複製到您的離線環境並進行安裝 `helm install trident-protect ./trident-protect-100.2510.0.tgz` 而不是在最後一步使用存儲庫引用。

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. 建立一個名為的文件 `protectValues.yaml`。請確保其中包含以下Trident Protect 設定：

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

這 `imageRegistry`和 `imagePullSecrets`這些值適用於所有組件影像，包括 `resourcebackup`和 `resourcerestore`。如果您將鏡像推送到註冊表中的特定儲存庫路徑（例如， `example.com:443/my-repo`），請在登錄欄位中包含完整路徑。這將確保所有圖像都從此處提取。 `<private-registry-url>/<image-name>:<tag>`。

7. 使用 Helm 安裝Trident Protect。代替 `<name_of_cluster>`集群名稱將分配給集群，並用於標識集群的備份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name_of_cluster> --version 100.2510.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

8. （選用）若要啟用偵錯日誌記錄（建議用於故障排除），請使用：

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name-of-cluster> --set logLevel=debug --version 100.2510.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

偵錯日誌記錄有助於NetApp支援人員排除故障，而無需變更日誌等級或重現問題。

有關其他 Helm Chart 設定選項，包括AutoSupport設定和命名空間過濾，請參閱 "自訂Trident Protect 安裝"。

安裝Trident Protect CLI 插件

您可以使用Trident Protect 命令列插件，它是Trident的一個擴充。 `tridentctl`用於建立和與Trident Protect 自訂資源 (CR) 互動的實用程式。

安裝Trident Protect CLI 插件

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：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.10.0/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：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.10.0/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：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.10.0/tridentctl-protect-macos-amd64
```

下載 **Mac ARM64 CPU** 的外掛程式

步驟

1. 下載Trident Protect CLI 外掛：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.10.0/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 外掛程式後，您可以為某些指令啟用自動補全功能。

啟用 **Bash Shell** 的自動完成功能

步驟

1. 建立完成腳本：

```
tridentctl-protect completion bash > 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. 建立完成腳本：

```
tridentctl-protect completion zsh > 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 的預設配置，以符合您環境的特定要求。

指定Trident Protect 容器資源限制

安裝Trident Protect 後，您可以使用設定檔來指定Trident Protect 容器的資源限制。設定資源限制可以控制Trident Protect 作業消耗叢集資源的程度。

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 後，您可以使用設定檔來修改Trident Protect 容器的 OpenShift 安全上下文約束 (SCC)。這些約束定義了 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 -n 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`。

設定其他**Trident Protect** 舵圖設置

您可以自訂AutoSupport設定和命名空間過濾以滿足您的特定要求。下表描述了可用的配置參數：

參數	類型	說明
自動支援代理	字串	為NetApp AutoSupport連線配置代理 URL。使用此功能透過代理伺服器路由支援包上傳。例子：`http://my.proxy.url` 。
自動支援.不安全	布林值	設定為時跳過AutoSupport代理連線的 TLS 驗證 `true`。僅用於不安全的代理連線。 (預設: `false`)
自動支援已啟用	布林值	啟用或停用每日Trident Protect AutoSupport組合包上傳。設定為 `false`每日定時上傳功能已停用，但您仍可以手動產生支援包。（預設： `true` ）
恢復跳過命名空間註釋	字串	若要從備份和復原作業中排除的命名空間註解的逗號分隔清單。允許您根據註釋過濾命名空間。
restoreSkipNamespaceLabels	字串	若要從備份和復原作業中排除的命名空間標籤的逗號分隔清單。允許您根據標籤過濾命名空間。

您可以使用 YAML 設定檔或命令列標誌來設定這些選項：

使用 **YAML** 文件

步驟

1. 建立設定檔並命名 `values.yaml`。
2. 在您建立的文件中，新增您想要自訂的設定選項。

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

3. 填充後 `values.yaml`具有正確值的文件，應用設定檔：

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

使用 **CLI** 標誌

步驟

1. 使用以下命令 `--set`標誌來指定單一參數：

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

將**Trident Protect Pod** 限制在特定節點上

您可以使用 Kubernetes nodeSelector 節點來選擇約束，根據節點標籤來控制哪些節點有資格執行Trident Protect pod。預設情況下， Trident Protect 僅限於執行 Linux 的節點。您可以根據需要進一步自訂這些限制條件。

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`。

管理Trident Protect

管理Trident Protect 授權和存取控制

Trident Protect 使用 Kubernetes 的角色為基礎的存取控制 (RBAC) 模型。預設情況下，Trident Protect 提供一個系統命名空間及其關聯的預設服務帳戶。如果您的組織擁有眾多使用者或特定的安全需求，則可以使用Trident Protect 的 RBAC 功能來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 ）。

只有管理員才能存取具有特權的Trident Protect 自訂資源對象，其中包括：

- * AppVault* ：需要儲存庫認證資料
- **AutoSupportBundle**：收集指標、日誌和其他敏感的Trident Protect數據
- * 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 時會建立一個預設的 ClusterRole 物件。您可以透過建立和套用 RoleBinding 物件將此 ClusterRole 綁定到服務帳戶。

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保護資源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 開源工具來監控Trident Protect 保護的資源的健康狀況。

kube-state-metrics 服務從 Kubernetes API 通訊產生指標。將其與Trident Protect 結合使用，可以顯示有關環境中資源狀態的有用資訊。

Prometheus 是一個工具包，它可以接收 kube-state-metrics 產生的數據，並將其呈現為關於這些物件的易於閱讀的資訊。kube-state-metrics 和 Prometheus 共同提供了一種方法，讓您可以監控使用Trident Protect 管理的資源的健康狀況和狀態。

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

要在Trident Protect 中啟用資源監控，您需要安裝和設定 kube-state-metrics、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. 將 Prometheus ServiceMonitor CRD 應用到叢集：

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

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

# Collect metrics from all namespaces
namespaces: ""

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

4. 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
```

5. 請依照下列說明配置 kube-state-metrics，以產生Trident Protect 使用的自訂資源的指標： "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.yaml`）並添加 kube 狀態指標服務信息。例如：

   prometheus.yaml：kube-state-metrics 服務與 Prometheus 的集成

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

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

prometheus.yaml：向 **Alertmanager** 發送警報

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.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.yaml`：

rules.yaml：定義失敗備份的 **Prometheus** 警報

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

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.yaml`。

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.yaml：向 **Slack** 頻道發送警報

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

產生**Trident Protect** 支援包

Trident Protect 使管理員能夠產生包含對NetApp支援有用的信息的捆綁包，包括有關受管理叢集和應用程式的日誌、指標和拓撲資訊。如果您已連接到互聯網，則可以使用自訂資源 (CR) 檔案將支援包上傳至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. 填充後 `trident-protect-support-bundle.yaml`具有正確值的文件，應用 CR：

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

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> -n trident-protect
```

監視和檢索支援包

使用任一方法建立支援包後，您可以監視其產生進度並將其檢索到本機系統。

步驟

1. 等待 `status.generationState`到達 `Completed`狀態。您可以使用以下命令監控產生進度：

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

2. 將支援包檢索到您的本機系統。從已完成的AutoSupport套件中取得複製指令：

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

找到 `kubectl cp`從輸出執行命令並運行它，用您喜歡的本機目錄取代目標參數。

升級Trident保護

您可以將Trident Protect 升級到最新版本，以享受新功能或修復錯誤。

- 從版本 24.10 升級時，升級期間執行的快照可能會失敗。此失敗不會阻止將來建立快照（無論是手動快照還是計劃快照）。如果升級期間快照失敗，您可以手動建立新快照以確保應用程式受到保護。

  為避免潛在的故障，您可以在升級前停用所有快照計劃，然後在升級後重新啟用。但是，這會導致升級期間遺失所有計劃的快照。

- 對於私人鏡像倉庫安裝，請確保目標版本所需的 Helm Chart 和鏡像在您的私人鏡像倉庫中可用，並驗證您的自訂 Helm 值與新 Chart 版本相容。更多信息，請參閱"從私人註冊表安裝Trident Protect"。

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 CRD：

> 如果您是從 25.06 之前的版本升級，則需要執行此步驟，因為 CRD 現在已包含在Trident Protect Helm 圖表中。

a. 運行此命令將 CRD 的管理從 `trident-protect-crds`到 `trident-protect` ：

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

b. 運行此命令刪除 `trident-protect-crds`圖表：

> 不要卸載 `trident-protect-crds`圖表使用 Helm，因為這可能會刪除您的 CRD 和任何相關資料。

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

3. 升級Trident保護：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2510.0 --namespace trident-protect
```

> 您可以透過新增以下內容來配置升級期間的日誌等級。 `--set logLevel=debug`升級命令。預設日誌等級為 `warn`。建議啟用偵錯日誌記錄進行故障排除，因為它可以幫助NetApp支援人員診斷問題，而無需更改日誌等級或重現問題。

管理及保護應用程式

使用**Trident Protect AppVault** 物件來管理儲存桶。

Trident Protect 的儲存桶自訂資源 (CR) 稱為 AppVault。AppVault 物件是儲存桶的聲明性 Kubernetes 工作流程表示。AppVault CR 包含儲存桶在保護作業（例如備份、快照、復原作業和SnapMirror複製）中所使用的必要配置。只有管理員才能建立應用保險庫。

在應用程式上執行資料保護操作時，您需要手動或從命令列建立 AppVault CR。 AppVaultCR 特定於您的環境，您可以使用本頁上的範例作為建立 AppVault CR 的指南。

> 確保 AppVault CR 位於安裝了Trident Protect 的叢集上。如果 AppVault CR 不存在或您無法訪問，命令列將顯示錯誤。

設定 **AppVault** 驗證和密碼

在建立 AppVault CR 之前，請確保您選擇的 AppVault 和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 CR 或Trident Protect CLI 外掛程式建立 AppVault 物件時，您可以為 Restic 和 Kopia 加密指定帶有自訂密碼的 Kubernetes 金鑰。如果您不指定金鑰， Trident Protect 將使用預設密碼。

- 手動建立 AppVault CR 時，使用 **spec.dataMoverPasswordSecretRef** 欄位指定金鑰。
- 使用Trident Protect CLI 建立 AppVault 物件時，請使用 `--data-mover-password-secret-ref`用於指定密鑰的參數。

建立資料移動者儲存庫密碼機密

請參考以下範例建立密碼密鑰。建立 AppVault 物件時，您可以指示Trident Protect 使用此金鑰向資料移動器儲存庫進行驗證。

- 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。
- 請將密碼保存在安全的地方。您將需要它來還原同一叢集或其他叢集上的資料。如果集群或 `trident-protect`命名空間被刪除後，沒有密碼您將無法還原備份或快照。

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
```

S3 相容於儲存 IAM 權限

當您存取與 S3 相容的儲存空間（例如 Amazon S3、通用 S3）時， "StorageGRID S3" ， 或者 "ONTAP S3"使用Trident Protect 時，您需要確保提供的使用者憑證具有存取儲存桶的必要權限。以下是授予使用Trident Protect 進行存取所需的最低權限的政策範例。您可以將此策略套用至管理 S3 相容儲存桶策略的使用者。

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

有關 Amazon S3 策略的更多信息，請參閱 "Amazon S3 文檔" 。

用於 Amazon S3（AWS）驗證的 EKS Pod Identity

Trident Protect 支援 Kopia 資料移動器操作的 EKS Pod Identity。此功能可實現對 S3 儲存桶的安全訪問，而無需將 AWS 憑證儲存在 Kubernetes 機密中。

EKS Pod Identity 與Trident Protect 的需求

在將 EKS Pod Identity 與Trident Protect 結合使用之前，請確保以下事項：

- 您的 EKS 叢集已啟用 Pod Identity。
- 您已建立具有必要的 S3 儲存桶權限的 IAM 角色。要了解更多信息，請參閱"S3 相容於儲存 IAM 權限"。
- IAM 角色與下列Trident Protect 服務帳號關聯：
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

有關啟用 Pod Identity 以及將 IAM 角色與服務帳戶關聯的詳細說明，請參閱 "AWS EKS Pod Identity 文檔"。

AppVault 設定 使用 EKS Pod Identity 時，請使用下列設定來設定您的 AppVault CR `useIAM: true`標記而不是明確的憑證：

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

適用於雲端供應商的 **AppVault** 主要世代範例

定義 AppVault CR 時，您需要包含憑證以存取提供者託管的資源，除非您使用 IAM 驗證。如何產生憑證金鑰將根據提供者的不同而有所不同。以下是幾個提供者的命令列金鑰產生範例。您可以使用下列範例為每個雲端提供者的憑證建立金鑰。

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 Amazon S3 （ AWS ） 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，這會很有用。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
```

對於使用 Pod Identity 和 Kopia 資料移動器的 EKS 環境，您可以刪除 `providerCredentials` 部分並添加 `useIAM: true`根據 `s3`配置。

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
```

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
```

StorageGRID S3

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

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

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
```

支援 `providerConfig.s3`配置選項

請參閱下表以了解 S3 提供者設定選項：

參數	說明	預設	範例
`providerConfig.s3.skipCertValidation`	停用 SSL/TLS 憑證驗證。	錯	“真”，“假”
`providerConfig.s3.secure`	啟用與 S3 端點的安全 HTTPS 通訊。	是的	“真”，“假”
`providerConfig.s3.proxyURL`	指定用於連接 S3 的代理伺服器的 URL。	沒有任何	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	提供用於 SSL/TLS 驗證的自訂根 CA 憑證。	沒有任何	"CN=MyCustomCA"
`providerConfig.s3.useIAM`	啟用 IAM 驗證以存取 S3 儲存桶。適用於 EKS Pod 識別。	錯	對、錯

檢視 **AppVault** 資訊

您可以使用Trident Protect CLI 外掛程式查看有關您在叢集上建立的 AppVault 物件的資訊。

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 安裝中指定了叢集名稱時，表格第一列中的叢集名稱才可用。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** 刪除 **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

您可以透過建立應用程式 CR 和關聯的 AppVault CR 來定義要使用Trident Protect 管理的應用程式。

建立 **AppVault CR**

您需要建立一個 AppVault CR，該 CR 將在對應用程式執行資料保護操作時使用，並且 AppVault CR 需要位於安裝了Trident Protect 的叢集上。AppVault CR 是針對您的特定環境的；有關 AppVault CR 的範例，請參閱："AppVault 自訂資源。"

定義應用程式

您需要定義要使用Trident Protect 管理的每個應用程式。您可以透過手動建立應用程式 CR 或使用Trident Protect CLI 來定義要管理的應用程式。

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 ）使用命名空間和標籤選取器來指定應用程式使用的命名空間和資源。應用程式命名空間必須是此清單的一部分。標籤選取器為選用項目，可用於篩選每個指定命名空間內的資源。
      - * spec.includedClusterScopedResources*: （ *Optional* ）使用此屬性來指定要包含在應用程式定義中的叢集範圍資源。此屬性可讓您根據這些資源的群組，版本，種類和標籤來選取這些資源。
        - **groupVersionKind** ：（ _required _ ）指定叢集範圍資源的 API 群組，版本及種類。
        - *labelSelector * ：（ *Optional* ）根據叢集範圍的資源標籤來篩選資源。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (可選) 此註解僅適用於從虛擬機定義的應用程序，例如 KubeVirt 環境，其中檔案系統凍結發生在快照之前。指定此應用程式在快照期間是否可以寫入檔案系統。如果設定為 true，應用程式將忽略全域設置，並且可以在快照期間寫入檔案系統。如果設定為 false，應用程式將忽略全域設置，並且在快照期間檔案系統將被凍結。如果指定了註解，但應用程式定義中沒有虛擬機，則忽略該註解。如未特別說明，則申請流程如下："全球Trident Protect 冷凍設置" 。

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:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. （可選）新增包含或排除標有特定標籤的資源的過濾：

   - **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"`：。

> 當兩者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`首先運行，然後 `labelSelector`應用於結果資源。

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"]
```

2. 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。

   建立應用程式時，您可以選擇使用註解來指定應用程式在快照期間是否可以寫入檔案系統。這僅適用於從虛擬機定義的應用程序，例如 KubeVirt 環境，其中檔案系統凍結發生在快照之前。如果您將註釋設定為 `true`該應用程式忽略全域設置，可以在快照期間寫入檔案系統。如果你把它設定為 `false`該應用程式忽略全域設置，導致檔案系統在快照期間凍結。如果使用了註解，但應用程式定義中沒有虛擬機，則該註解將被忽略。如果您不使用註解，應用程式將遵循以下規則："全球Trident Protect 冷凍設置" 。

   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">
```

您可以使用 `--resource-filter-include`和 `--resource-filter-exclude`用於包含或排除資源的標誌 `resourceSelectionCriteria`例如群組、類型、版本、標籤、名稱和命名空間，如下例所示：

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

使用Trident Protect 保護應用程式

您可以使用自動保護策略或臨時保護策略，透過拍攝快照和備份來保護Trident Protect 管理的所有應用程式。

您可以設定Trident Protect 在資料保護作業期間凍結和解凍檔案系統。"了解更多關於使用Trident Protect 設定檔系統凍結的信息"。

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**：（可選）：用作備份來源的快照的名稱。如果未提供，將會建立並備份暫存快照。

     YAML 範例：

```
---
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>
```

您可以選擇性地使用 `--full-backup`旗標來指定備份是否應為非遞增備份。依預設，所有備份都是遞增備份。使用此旗標時，備份會變成非遞增備份。最佳做法是定期執行完整備份，然後在完整備份之間執行遞增備份，以將與還原相關的風險降至最低。

支援的備份註釋

下表描述了建立備份 CR 時可以使用的註解：

註釋	類型	說明	預設值
protect.trident.netapp.io/full-backup	字串	指定備份是否應為非增量備份。設定為 `true` 建立非增量備份。最佳實踐是定期執行完整備份，然後在兩次完整備份之間執行增量備份，以最大限度地降低與復原相關的風險。	"假"
protect.trident.netapp.io/snapshot-completion-timeout	字串	完成整個快照操作允許的最長時間。	60米
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	字串	卷快照達到可用狀態所需的最長時間。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	字串	建立磁碟區快照允許的最長時間。	5米
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待新建立的持久卷聲明 (PVC) 到達的最大時間（以秒為單位） `Bound`操作失敗前的階段。	1200（20分鐘）

建立資料保護排程

保護策略透過按照定義的計劃建立快照、備份或兩者來保護應用程式。您可以選擇每小時、每天、每周和每月建立快照和備份，並可以指定要保留的副本數量。您可以使用 full-backup-rule 註解來排程非增量式完整備份。預設情況下，所有備份都是增量的。定期執行完整備份以及其間的增量備份有助於降低與復原相關的風險。

- 您可以透過設定 `backupRetention`歸零， `snapshotRetention`為大於零的值。環境 `snapshotRetention`為零意味著任何計劃的備份仍將建立快照，但這些快照是臨時的，並在備份完成後立即刪除。
- 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**: (必要) 要保留的備份數量。零表示不應建立備份（僅快照）。
   - **spec.backupReclaimPolicy**: (可選) 決定如果備份 CR 在其保留期內被刪除，則備份會發生什麼情況。保留期過後，備份檔案總是會被刪除。可能的值（區分大小寫）：
     - `Retain` （預設）
     - `Delete`
   - **spec.snapshotRetention**: (必需) 要保留的快照數量。零表示不建立任何快照。
   - **spec.snapshotReclaimPolicy**: (可選) 決定如果快照 CR 在其保留期內被刪除，則快照會發生什麼情況。保留期過後，快照總是會被刪除。可能的值（區分大小寫）：
     - `Retain`
     - `Delete`(預設)
   - * 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`)
     - `Custom`
   - **spec.dayOfMonth**：（可選）計畫應運行的月份日期（1 - 31）。如果粒度設定為 `Monthly`。該值必須以字串形式提供。
   - **spec.dayOfWeek**：（可選）計畫應運行的星期幾（0 - 7）。值 0 或 7 表示星期日。如果粒度設定為 `Weekly`。該值必須以字串形式提供。
   - **spec.hour**：（可選）計畫應運行的小時數（0 - 23）。如果粒度設定為 `Daily` ， `Weekly` ，或者 `Monthly`。該值必須以字串形式提供。
   - **spec.minute**：（可選）計畫應運行的小時中的分鐘數（0 - 59）。如果粒度設定為 `Hourly` ， `Daily` ， `Weekly` ，或者 `Monthly`。該值必須以字串形式提供。

備份和快照計劃的範例 YAML：

```
---
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: Daily
  hour: "0"
  minute: "0"
```

僅快照計劃的範例 YAML：

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

3. 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> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_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> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

以下選項可讓您對日程安排進行更多控制：

- 完整備份計畫：使用 `--full-backup-rule` 標記以安排非增量式完整備份。此標誌僅適用於 `--granularity Daily`。可能的值：
  - `Always`每天都要建立完整備份。
  - 具體工作日：指定一個或多個日期，以逗號分隔（例如， `"Monday,Thursday"`）。有效值：星期一、星期二、星期三、星期四、星期五、星期六、星期日。

    這 `--full-backup-rule` 此標誌位不適用於按小時、按週或按月劃分的粒度。
- 僅快照計劃：設定 `--backup-retention 0` 並指定一個大於零的值 `--snapshot-retention`。

支援的日程註釋

下表描述了建立計劃變更請求 (CR) 時可以使用的註釋：

註釋	類型	說明	預設值
protect.trident.netapp.io/full-backup-rule	字串	指定安排完整備份的規則。你可以將其設定為 `Always` 您可以根據需要進行持續完整備份或自訂備份。例如，如果您選擇按日粒度進行備份，則可以指定應進行完整備份的星期幾（例如， `"Monday,Thursday"`）。有效的工作日值為：星期一、星期二、星期三、星期四、星期五、星期六、星期日。請注意，此註釋只能用於已包含以下內容的日程表： `granularity` 設定為 `Daily`。	未設定（所有備份均為增量備份）
protect.trident.netapp.io/snapshot-completion-timeout	字串	完成整個快照操作允許的最長時間。	60米
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	字串	卷快照達到可用狀態所需的最長時間。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	字串	建立磁碟區快照允許的最長時間。	5米
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待新建立的持久卷聲明 (PVC) 到達的最大時間（以秒為單位） `Bound`操作失敗前的階段。	1200（20分鐘）

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。

確保回收策略設定為 `Delete`**從物件儲存中刪除所有備份資料。該策略的預設值是** `Retain`**以避免意外資料遺失。如果政策沒有改變** `Delete` **，備份資料將保留在物件儲存中，需要手動刪除。**

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，則可以為使用 azure-netapp-files 儲存類別且在Trident 24.06 之前建立的儲存後端啟用節省空間的備份和還原功能。此功能適用於 NFSv4 卷，並且不會佔用容量池中的額外空間。

開始之前

請確認下列事項：

- 您已安裝Trident Protect。
- 您已在Trident Protect中定義了一個應用程式。在您完成此步驟之前，此應用程式的保護功能將受到限制。
- 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"
   ```

   +
   如果未啟用快照目錄， Trident Protect 將選擇常規備份功能，該功能會在備份過程中暫時佔用容量池中的空間。在這種情況下，請確保容量池中有足夠的空間來建立與被備份磁碟區大小相同的臨時磁碟區。

結果

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

還原應用程式

使用**Trident Protect** 恢復應用程式

您可以使用Trident Protect 從快照或備份中還原您的應用程式。將應用程式還原到同一群集時，從現有快照恢復速度會更快。

- 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。
- 對於 qtree 卷，支援從備份還原到其他命名空間或原始命名空間。但是，對於 qtree 卷，不支援從快照還原到其他命名空間或原始命名空間。
- 您可以使用進階設定來自訂恢復操作。欲了解更多信息，請參閱 "使用進階Trident Protect 恢復設定"。

從備份還原至不同的命名空間

當您使用 BackupRestore CR 將備份還原到不同的命名空間時， Trident Protect 會在新的命名空間中還原應用程序，並為還原的應用程式建立一個應用程式 CR。為了保護已還原的應用程序，可以建立按需備份或快照，或

製定保護計劃。

- 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。
- 使用 CR 還原到新命名空間時，必須先手動建立目標命名空間，然後再套用 CR。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

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"。

當您使用 Kopia 作為資料移動器還原備份時，您可以選擇在 CR 中指定註解或使用 CLI 來控制 Kopia 使用的暫存的行為。請參閱 "Kopia 文件"有關您可以配置的選項的詳細資訊。使用 `tridentctl-protect create --help`有關使用Trident Protect CLI 指定註釋的更多信息，請參閱命令。

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`。

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

3. （ *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 會自動選擇一些資源，因為它們與您選擇的資源有關聯。例如，如果您選擇持久性磁碟區宣告資源且它有一個關聯的 pod， Trident Protect 也會還原關聯的 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>
```

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"。

> 當您使用 Kopia 作為資料移動器還原備份時，您可以選擇在 CR 中指定註解或使用 CLI 來控制 Kopia 使用的暫存的行為。請參閱 "Kopia 文件"有關您可以配置的選項的詳細資訊。使用 `tridentctl-protect create --help`有關使用Trident Protect CLI 指定註釋的更多信息，請參閱命令。

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 會自動選擇一些資源，因為它們與您選擇的資源有關聯。例如，如果您選擇持久性磁碟區宣告資源且它有一個關聯的 pod， Trident Protect 也會還原關聯的 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>
```

如果原始叢集發生問題，您可以將備份還原至不同的叢集。

- 當您使用 Kopia 作為資料移動器還原備份時，您可以選擇在 CR 中指定註解或使用 CLI 來控制 Kopia 使用的暫存的行為。請參閱 "Kopia 文件"有關您可以配置的選項的詳細資訊。使用 `tridentctl-protect create --help`有關使用Trident Protect CLI 指定註釋的更多信息，請參閱命令。
- 使用 CR 還原到新命名空間時，必須先手動建立目標命名空間，然後再套用 CR。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

開始之前

確保符合下列先決條件：

- 目標叢集已安裝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。
- 執行 AppVault CR 時，請確保本機環境可以連接到 AppVault CR 中定義的物件儲存桶。 `tridentctl-protect get appvaultcontent`命令。如果網路限制阻止訪問，請改為從目標叢集上的 pod 內執行Trident Protect CLI。
- 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 外掛程式檢查目標叢集上 AppVault CR 的可用性：

```
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> \
--context <destination_cluster_name> \
-n <application_namespace>
```

從快照還原至不同的命名空間

您可以使用自訂資源 (CR) 檔案從快照還原數據，還原到不同的命名空間或原始來源命名空間。當您使用 SnapshotRestore CR 將快照還原到不同的命名空間時， Trident Protect 會在新的命名空間中還原應用程序，並為還原的應用程式建立應用程式 CR。為了保護已還原的應用程序，可以建立按需備份或快照，或製定保護計劃。

- SnapshotRestore 支持 `spec.storageClassMapping`屬性，但僅當來源和目標儲存類別使用相同的儲存後端。如果您嘗試恢復到 `StorageClass`如果使用不同的儲存後端，則復原操作將會失敗。
- 使用 CR 還原到新命名空間時，必須先手動建立目標命名空間，然後再套用 CR。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

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-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`。

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

3. （ *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 會自動選擇一些資源，因為它們與您選擇的資源有關聯。例如，如果您選擇持久性磁碟區宣告資源且它有一個關聯的 pod， Trident Protect 也會還原關聯的 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 會自動選擇一些資源，因為它們與您選擇的資源有關聯。例如，如果您選擇持久性磁碟區宣告資源且它有一個關聯的 pod， Trident Protect 也會還原關聯的 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 <namespace/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}'
```

使用進階**Trident Protect** 恢復設定

您可以使用進階設定（例如註解、命名空間設定和儲存選項）自訂復原操作，以滿足您的特定要求。

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 Red Hat OpenShift，請務必注意命名空間註解在 OpenShift 環境中的重要角色。命名空間註解可確保復原的 pod 遵守 OpenShift 安全性情境約束 (SCC) 定義的適當權限和安全性配置，並且可以存取磁碟區而不會出現權限問題。欲了解更多信息，請參閱"OpenShift 安全性內容限制文件"。

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：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

執行復原或故障轉移操作時，任何命名空間註解和標籤都將生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不參與恢復或故障轉移操作。確保在初始 Helm 安裝期間配置這些設定。欲了解更多信息，請參閱 "設定其他Trident Protect 舵圖設置"。

如果您使用 Helm 安裝了來源應用程序， `--create-namespace`國旗，給予特殊待遇 `name`標籤鍵。在復原或故障轉移過程中， Trident Protect 會將此標籤複製到目標命名空間，但如果來源命名空間的值與來源命名空間的值匹配，則會將值更新為目標命名空間的值。如果此值與來源命名空間不匹配，則會將其複製到目標命名空間，而不做任何變更。

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

支援的字段

本節介紹可用於恢復操作的其他欄位。

儲存類別映射

這 `spec.storageClassMapping`屬性定義從來源應用程式中的現有儲存類別到目標叢集上的新儲存類別的對應。您可以在具有不同儲存類別的叢集之間移轉應用程式時或變更 BackupRestore 作業的儲存後端時使用此功能。

範例：

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

支持的註釋

本節列出了系統中支援配置各種行為的註解。如果使用者未明確設定註解，系統將使用預設值。

註釋	類型	說明	預設值
protected.trident.netapp.io/data-mover-timeout-sec	字串	允許資料移動設備操作停止的最長時間（以秒為單位）。	“300”
protected.trident.netapp.io/kopia-content-cache-size-limit-mb	字串	Kopia 內容快取的最大大小限制（以兆位元組為單位）。	“1000”
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待新建立的持久卷聲明 (PVC) 到達的最大時間（以秒為單位）`Bound`操作失敗前的階段。適用於所有還原 CR 類型（備份還原、備份就地還原、快照還原、快照就地還原）。如果您的儲存後端或叢集經常需要更多時間，請使用更高的值。	1200（20分鐘）

使用NetApp SnapMirror和Trident Protect 複製應用程式

使用Trident Protect，您可以利用NetApp SnapMirror技術的非同步複製功能，將資料和應用程式變更從一個儲存後端複製到另一個儲存後端，無論是在同一叢集內還是在不同叢集之間。

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 Red Hat OpenShift，請務必注意命名空間註解在 OpenShift 環境中的重要角色。命名空間註解可確保復原的 pod 遵守 OpenShift 安全性情境約束 (SCC) 定義的適當權限和安全性配置，並且可以存取磁碟區而不會出現權限問題。欲了解更多信息，請參閱"OpenShift 安全性內容限制文件"。

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：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

> ⓘ 執行復原或故障轉移操作時，任何命名空間註解和標籤都將生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不參與恢復或故障轉移操作。確保在初始 Helm 安裝期間配置這些設定。欲了解更多信息，請參閱 "設定其他Trident Protect 舵圖設置"。

如果您使用 Helm 安裝了來源應用程序， `--create-namespace`國旗，給予特殊待遇 `name`標籤鍵。在復原或故障轉移過程中， Trident Protect 會將此標籤複製到目標命名空間，但如果來源命名空間的值與來源命名空間的值匹配，則會將值更新為目標命名空間的值。如果此值與來源命名空間不匹配，則會將其複製到目標命名空間，而不做任何變更。

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

您可以設定Trident Protect 在資料保護作業期間凍結和解凍檔案系統。"了解更多關於使用Trident Protect 設定檔系統凍結的信息"。

故障轉移和反向操作期間的執行掛鉤

當使用 AppMirror 關係保護您的應用程式時，您應該在故障轉移和反轉操作期間注意與執行掛鉤相關的特定行為。

- 在故障轉移期間，執行掛鉤會自動從來源叢集複製到目標叢集。您無需手動重新建立它們。故障轉移後，執行掛鉤仍存在於應用程式中，並將在任何相關操作期間執行。
- 在反向同步或反向重新同步期間，應用程式上所有現有的執行鉤子都將被移除。當來源應用程式成為目標應用程式時，這些執行鉤子將失效，並將被刪除以阻止其執行。

要了解有關執行鉤子的更多信息，請參閱"管理Trident Protect 執行鉤子"。

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 資源以及應用程式每個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. 在來源叢集上，為來源應用程式建立 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> -n
trident-protect
```

2. 在來源叢集上，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. （可選）在來源叢集上，對來源應用程式進行快照。此快照將用作目標叢集上應用程式的基礎。如果跳過此步驟，則需要等待下一次排程快照運行，以便取得最新的快照。若要建立隨選快照，請參閱 "建立隨需快照"。

4. 在來源叢集上，建立複製計劃 CR：

除了下面提供的計劃外，建議建立一個單獨的每日快照計劃，保留期為 7 天，以便在對等 ONTAP 叢集之間維護通用快照。這可確保快照最多可用 7 天，但保留期可根據使用者需求自訂。

如果發生故障轉移，系統可以使用這些快照最多 7 天進行反向操作。這種方法使反向過程更快、更有效率，因為只會傳輸自上次快照以來所做的更改，而不是所有資料。

如果應用程式的現有計劃已經滿足所需的保留要求，則不需要額外的計劃。

使用 **CR** 建立複製計劃

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**: (必需) 此值必須與來源應用程式的 AppVault 的 metadata.name 欄位相符。
      - **spec.applicationRef**: (必需) 此值必須與來源應用程式 CR 的 metadata.name 欄位相符。
      - *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
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

   i. 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** 建立複製計劃

a. 建立複製計劃，並將括號中的值替換為您環境中的資訊：

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

範例：

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

5. 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`）。
6. 套用 CR ：

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

7. 為目的地叢集上的目的地應用程式建立目的地 AppVault CR 。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 trident-protect
```

8. 在目標叢集上，建立 AppMirrorRelationship CR 檔案。

使用 CR 時，請在套用 CR 之前手動建立目標命名空間。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

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.sourceApplicationUID**: （必要）此值必須與您在來源應用程式 CR 中定義的來源應用程式的 UID 相符。
- **spec.storageClassName**: (可選) 選擇叢集上有效的儲存類別的名稱。儲存類別必須連結到與來源環境建立對等連線的ONTAP儲存 VM。如果未提供儲存類，則預設使用叢集上的預設儲存類別。
- **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 對象，並將括號中的值替換為您環境中的資訊：

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

範例：

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

9. （ *Optional* ）在目的地叢集上，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，您可以將複製的應用程式故障轉移到目標叢集。此過程會停止複製關係，並將應用程式在目標叢集上連線。如果來源叢集上的應用程式正在運行， Trident Protect 不會停止該應用程式。

步驟

1. 在目標叢集上，編輯 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. 選用：在來源叢集上，建立來源應用程式的快照。如此可確保擷取來源叢集的最新變更。
2. 在目標叢集上，編輯 AppMirrorRelationship CR 檔案（例如 `trident-protect-relationship.yaml`），並將 spec.desiredState 的值變更為 `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. 在原始目的地叢集上，刪除 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. 請確定新目的地（原始來源叢集）已同時使用 AppVault CRS 進行設定。
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

當您反轉複製方向時， Trident Protect 會將應用程式移至目標儲存後端，同時繼續複製回原始來源儲存後端。 Trident Protect 會停止來源應用程式並將資料複製到目標位置，然後再故障轉移到目標應用程式。

在這種情況下、您要交換來源和目的地。

步驟

1. 在來源叢集上，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. 在來源叢集上，關機快照完成後，取得關機快照的狀態：

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

3. 在來源叢集上，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 ；這將是最後一條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，從新的目的地叢集移轉至新的來源叢集，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，您可以透過以下步驟序列在故障轉移作業後實現「故障復原」。在此恢復原始複製方向的工作流程中， Trident Protect 會將任何應用程式變更複製（重新同步）回原始來源應用程序，然後再反轉複製方向。

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. 在目前的目標叢集上，刪除 AppMirrorRelationship CR ：

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

使用Trident Protect 遷移應用程式

您可以透過還原備份資料在叢集之間或不同的儲存類別之間遷移您的應用程式。

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（執行跨叢集克隆），請在來源叢集上建立備份，然後將備份還原到不同的叢集。請確保目標叢集上已安裝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

您可以透過將備份還原到目標儲存類，將應用程式從一個儲存類別遷移到另一個儲存類別。

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 執行鉤子

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 支援以下幾種執行鉤子類型，取決於它們的運行時機：

- 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，則會發生檔案系統凍結。"了解更多關於使用Trident Protect 設定檔系統凍結的信息"。
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 要求執行鉤子使用的腳本以可執行 shell 腳本的格式編寫。
- 指令碼大小上限為96KB。
- Trident Protect 使用執行鉤子設定和任何符合條件來決定哪些鉤子適用於快照、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。

新增到執行掛鉤的每個過濾器都使用正規表示式來匹配叢集中的容器。當鉤子與容器匹配時，鉤子將在該容器上運行其關聯的腳本。過濾器的正規表示式使用正規表示式 2 (RE2) 語法，該語法不支援建立從符合清單中排除容器的過濾器。有關Trident Protect 在執行鉤子過濾器中支援的正規表示式語法的詳細信息，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鉤子。您需要擁有所有者、管理員或成員權限才能建立執行鉤子。

使用 **CR**

步驟

1. 建立自訂資源（ CR ）檔案並命名為 `trident-protect-hook.yaml`。
2. 設定以下屬性以符合您的Trident Protect 環境和叢集設定：
   - * 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. 設定以下屬性以符合您的Trident Protect 環境和叢集設定：
   - * 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. 設定以下屬性以符合您的Trident Protect 環境和叢集設定：
   - * 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到完整版產品，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. 刪除Trident Protect CR 檔案：

   25.06 及更高版本不需要此步驟。

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

2. 移除Trident保護：

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

3. 移除Trident Protect 命名空間：

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