使用Trident Protect 保護應用程式

Trident

NetApp
January 15, 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。

下一步是什麼？

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：

- 亞馬遜彈性 Kubernetes 服務 (EKS)
- Google Kubernetes 引擎 (GKE)
- Microsoft Azure Kubernetes 服務 (AKS)
- 紅帽 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 "這些說明"。

Trident Protect 儲存後端相容性

Trident Protect 支援以下儲存後端：

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP儲存陣列
- Google Cloud NetApp Volumes
- Azure NetApp Files

請確保您的儲存後端符合以下要求：

- 確保連接到叢集的NetApp儲存裝置使用的是Trident 24.02 或更高版本（建議使用Trident 24.10）。
- 請確保您擁有NetApp ONTAP儲存後端。
- 請確保您已配置用於儲存備份的物件儲存桶。
- 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-economy 容量要求

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快照目錄的磁碟區不相容。對於這些應用，您需要透過在ONTAP儲存系統上執行以下命令來隱藏快照目錄：

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

您可以透過對每個 nas-economy 磁碟區執行以下命令來啟用快照目錄，並將命令替換為 `<volume-UUID>`使用要變更的磁碟區的 UUID：

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

您可以透過設定Trident後端設定選項，為新磁碟區預設啟用快照目錄。 `snapshotDir`到 `true`。現有捲數不受影響。

使用 **KubeVirt** 虛擬機器保護數據

Trident Protect 24.10 和 24.10.1 及更高版本在保護運行在 KubeVirt VM 上的應用程式時，行為有所不同。對於這兩個版本，您都可以在資料保護操作期間啟用或停用檔案系統凍結和解凍。

在恢復操作期間，任何 `VirtualMachineSnapshots`為虛擬機器 (VM) 建立的設定檔無法復原。

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。"配置虛擬化"然後使用以下命令：

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

Trident Protect 24.10.1 及更高版本

從Trident Protect 24.10.1 開始， Trident Protect 會在資料保護作業期間自動凍結和解凍 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
```

SnapMirror複製的要求

NetApp SnapMirror複製功能可與Trident Protect 搭配使用，適用於下列ONTAP解決方案：

- 本地部署的NetApp FAS、 AFF和ASA集群
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

ONTAP叢集對SnapMirror複製的要求

如果您打算使用SnapMirror複製功能，請確保您的ONTAP叢集符合以下要求：

- * 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`：極品飛車
  - `ontap-san`：iSCSI
  - `ontap-san`：FC
  - `ontap-san`：NVMe/TCP（最低要求ONTAP版本 9.15.1）
- 許可證：使用資料保護包的ONTAP SnapMirror非同步許可證必須在來源 ONTAP 叢集和目標ONTAP叢集上啟用。請參閱 "ONTAP中的SnapMirror許可概述"了解更多。

  從ONTAP 9.10.1 開始，所有許可證均以NetApp許可證文件 (NLF) 的形式交付，這是一個可以啟用多種功能的單一文件。請參閱"ONTAP One 隨附的許可證"了解更多。

  僅支援SnapMirror非同步保護。

SnapMirror複製的對等連線注意事項

如果您打算使用儲存後端對等互連，請確保您的環境符合以下要求：

- 叢集和 **SVM**： ONTAP儲存後端必須相互連接。請參閱 "叢集和SVM對等連接概述"了解更多。

  確保兩個ONTAP叢集之間複製關係中使用的 SVM 名稱是唯一的。

- * NetApp Trident和 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複製的Trident / ONTAP配置

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

請確保您的 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 讀取資料和寫入數據，以實現應用程式物件複製。
- 網路連線：設定防火牆規則、儲存桶權限和 IP 允許列表，以啟用兩個叢集和 AppVault 之間透過 WAN 進行通訊。

許多企業環境在廣域網路連線中實施嚴格的防火牆策略。在配置複製之前，請先與您的基礎架構團隊確認這些網路需求。

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。

安裝Trident Protect

從NetApp安裝Trident Protect

步驟

1. 新增Trident Helm 倉庫：

```
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 100.2506.0 --create
-namespace --namespace trident-protect
```

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. 將以下鏡像拉取到本地計算機，更新標籤，然後將其推送到您的私有鏡像倉庫：

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

例如：

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

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

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

2. 建立Trident Protect 系統命名空間：

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

3. 登入註冊表：

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

4. 建立用於私有註冊表驗證的拉取金鑰：

```
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 倉庫：

```
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：

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

7. 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 100.2506.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

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

在使用命令列實用程式之前，需要將其安裝在用於存取叢集的電腦上。根據您的機器使用的是 x64 還是ARM CPU，請依照以下步驟操作。

下載適用於 **Linux AMD64 CPU** 的插件

步驟

1. 下載Trident Protect CLI 外掛：

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

下載適用於 **Linux ARM64 CPU** 的插件

步驟

1. 下載Trident Protect CLI 外掛：

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

下載適用於 **Mac AMD64 CPU** 的插件

步驟

1. 下載Trident Protect CLI 外掛：

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

下載適用於 **Mac ARM64 CPU** 的插件

步驟

1. 下載Trident Protect CLI 外掛：

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

1. 啟用插件二進位檔案的執行權限：

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

2. 將插件二進位檔案複製到 PATH 環境變數中定義的位置。例如， `/usr/bin`或者 `/usr/local/bin`（您可能需要更高的權限）：

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

3. 您也可以選擇將插件二進位檔案複製到您主目錄中的某個位置。在這種情況下，建議確保該位置已新增至您的 PATH 環境變數：

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

將插件複製到 PATH 環境變數中的某個位置，即可透過鍵入以下命令來使用該插件。 `tridentctl-protect`或者 `tridentctl protect`從任何地點。

查看**Trident CLI** 外掛程式幫助

您可以使用插件內建的幫助功能來獲得有關插件功能的詳細幫助：

步驟

1. 使用幫助功能查看使用指南：

```
tridentctl-protect help
```

啟用指令自動補全

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。

為 **Bash shell** 啟用自動補全功能

步驟

1. 下載完成腳本：

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

2. 在您的使用者目錄下建立一個新目錄，用於存放腳本：

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

3. 將下載的腳本移到 `~/.bash/completions`目錄：

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

4. 將以下行新增至 `~/.bashrc`您主目錄中的檔案：

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

為 **Z shell** 啟用自動補全

步驟

1. 下載完成腳本：

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

2. 在您的使用者目錄下建立一個新目錄，用於存放腳本：

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

3. 將下載的腳本移到 `~/.zsh/completions`目錄：

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

4. 將以下行新增至 `~/.zprofile`您主目錄中的檔案：

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

結果

下次登入 shell 時，您可以使用 tridentctl-protect 外掛程式進行指令自動補全。

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. 建立一個名為的文件 `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。

   以下範例設定檔顯示了可用設置，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. 應用以下值 `resourceLimits.yaml`文件：

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

自訂安全上下文約束

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. 建立一個名為的文件 `sccconfig.yaml`。
2. 在文件中新增 SCC 選項，並根據您的環境需求修改參數。

   以下範例顯示了 SCC 選項的參數預設值：

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

下表描述了SCC選項的參數：

範圍	描述	預設
創造	確定是否可以建立 SCC 資源。僅當 `scc.create`設定為 `true`Helm 安裝過程會辨識 OpenShift 環境。如果不是在 OpenShift 上運行，或者如果 `scc.create`設定為 `false` 不會創建 SCC 資源。	真的
姓名	指定SCC的名稱。	三叉戟保護工作
優先事項	確定 SCC 的優先順序。優先順序較高的SCC會優先於優先順序較低的SCC進行評估。	1

3. 應用以下值 `sccconfig.yaml`文件：

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

這將用指定的值替換預設值。 `sccconfig.yaml`文件。

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：

範圍	類型	描述
自動支援代理	細繩	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“跳過AutoSupport代理連接的 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` ）

範圍	類型	描述
恢復跳過命名空間註釋	細繩	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	細繩	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. 建立一個設定檔並將其命名為 `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. 填寫完後 `values.yaml`使用正確的值建立配置文件，並套用該設定檔：

```
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`用於指定個別參數的標誌：

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

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. 建立一個名為的文件 `nodeSelectorConfig.yaml`。
2. 在檔案中新增 nodeSelector 選項，並修改檔案以新增或變更節點標籤，從而根據您的環境需求進行限制。例如，以下檔案包含預設的作業系統限制，但同時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. 應用以下值 `nodeSelectorConfig.yaml`文件：

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

這將預設限制替換為您在設定中指定的限制。 `nodeSelectorConfig.yaml`文件。

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。

叢集管理員始終擁有對預設資源的存取權限。 `trident-protect`命名空間，並且可以存取所有其他命名空間中的資源。要控制對資源和應用程式的訪問，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，預設情況下，任何使用者都無法建立應用程式資料管理變更請求 (CR)。 `trident-protect`命名空間。您需要在應用程式命名空間中建立應用程式資料管理 CR（最佳實踐是在與其關聯的應用程式相同的命名空間中建立應用程式資料管理 CR）。

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**：需要儲存桶憑證數據
- **AutoSupportBundle**：收集指標、日誌和其他敏感的Trident Protect數據
- **AutoSupportBundleSchedule**：管理日誌收集計劃

最佳實踐是使用基於角色的存取控制 (RBAC) 將對特權物件的存取限制在管理員範圍內。

有關基於角色的存取控制 (RBAC) 如何管理對資源和命名空間的存取的更多信息，請參閱... "Kubernetes RBAC 文檔" 。

有關服務帳戶的信息，請參閱 "Kubernetes 服務帳戶文檔"。

範例：管理兩組使用者的存取權限

例如，一個組織有集群管理員、一組工程用戶和一組行銷用戶。叢集管理員將完成以下任務，以建立一個環境，其中工程組和行銷組各自只能存取分配給其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

創建命名空間可以讓你從邏輯上分離資源，並更好地控制誰可以存取這些資源。

步驟

1. 為工程組建立一個命名空間：

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

2. 為行銷組創造命名空間：

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

步驟 **2**：建立新的服務帳戶，以便與每個命名空間中的資源互動

您建立的每個新命名空間都附帶一個預設服務帳戶，但您應該為每個使用者群組建立一個服務帳戶，以便將來必要時可以進一步在群組之間劃分權限。

步驟

1. 為工程團隊建立一個服務帳戶：

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

2. 為行銷團隊建立一個服務帳戶：

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

步驟 **3**：為每個新服務帳戶建立一個金鑰

服務帳戶金鑰用於對服務帳戶進行身份驗證，如果遭到洩露，可以輕鬆刪除並重新建立。

步驟

1. 為工程服務帳戶建立一個金鑰：

```
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一個金鑰：

```
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** 對象，將 **ClusterRole** 物件綁定到每個新的服務帳戶。

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. 將叢集角色綁定到工程服務帳戶：

```
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. 將群集角色綁定到行銷服務帳戶：

```
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. 確認工程用戶無法存取行銷資源：

```
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 物件的存取權限

若要執行備份和快照等資料管理任務，叢集管理員需要授予個別使用者對 AppVault 物件的存取權限。

步驟

1. 建立並套用 AppVault 和金鑰組合的 YAML 文件，以授予使用者對 AppVault 的存取權限。例如，以下 CR 授予使用者對 AppVault 的存取權限 `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，使叢集管理員能夠授予對命名空間中特定資源的存取權限。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. 建立並套用角色綁定 CR，將權限綁定到使用者 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. 請確認權限是否正確。

   a. 嘗試檢索所有命名空間的 AppVault 物件資訊：

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

   您應該會看到類似以下內容的輸出：

```
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. 測試用戶是否可以獲得他們現在有權訪問的 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
```

您應該會看到類似以下內容的輸出：

```
yes
```

結果

您授予 AppVault 權限的使用者應該能夠使用授權的 AppVault 物件進行應用程式資料管理操作，並且不應該能夠存取指派的命名空間之外的任何資源，或建立他們無權存取的新資源。

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，它可以接收 Prometheus 等工具發送的警報，並將它們路由到您配置的目標位置。

> 這些步驟中包含的配置和指導僅供參考；您需要根據自己的環境進行自訂。請參閱以下官方文件以取得具體說明和支援：
>
> - "kube-state-metrics 文檔"
> - "普羅米修斯文檔"
> - "Alertmanager 文檔"

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-state-metrics**

您可以使用 Helm 安裝 kube-state-metrics。

步驟

1. 新增 kube-state-metrics Helm chart。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 chart 建立一個設定檔（例如， `metrics-config.yaml` ）。您可以根據自身環境自訂以下範例配置：

metrics-config.yaml：kube-state-metrics Helm chart 配置

```
---
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. 透過部署 Helm chart 來安裝 kube-state-metrics。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-metrics 自訂資源文檔" 。

安裝 Prometheus

您可以按照以下說明安裝 Prometheus： "普羅米修斯文檔" 。

安裝 Alertmanager

您可以按照以下說明安裝 Alertmanager： "Alertmanager 文檔" 。

步驟 2：配置監控工具以協同工作

安裝監控工具後，需要設定它們以使其協同工作。

步驟

1. 將 kube-state-metrics 與 Prometheus 整合。編輯 Prometheus 配置文件(`prometheus.yaml`）並新增 kube-state-metrics 服務資訊。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. 配置 Prometheus 將警報路由到 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

Prometheus 現在可以從 kube-state-metrics 收集指標，並且可以向 Alertmanager 發送警報。現在您可以設定觸發警報的條件以及警報的發送位置。

步驟 **3**：設定警報和警報目標

配置好工具協同工作後，還需要配置哪些類型的信息會觸發警報，以及警報應該發送到哪裡。

警報範例：備份失敗

以下範例定義了一個關鍵警報，當備份自訂資源的狀態設定為「是」時，警報將會被觸發。 `Error`持續5秒或更長時間。您可以自訂此範例以符合您的環境，並將此 YAML 程式碼片段包含在您的專案中。 `prometheus.yaml`設定檔：

rules.yaml：定義備份失敗的 **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."
```

配置 **Alertmanager** 向其他管道發送警報

您可以設定 Alertmanager，使其將通知傳送到其他管道，例如電子郵件、PagerDuty、Microsoft Teams 或其他通知服務，只需在設定檔中指定相應的配置即可。 `alertmanager.yaml`文件。

以下範例配置 Alertmanager 向 Slack 頻道發送通知。若要根據您的環境自訂此範例，請取代以下值： `api_url`金鑰包含您環境中使用的 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** 建立支援包

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-support-bundle.yaml` ）。
2. 配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.triggerType**: (*Required*) 確定支援包是立即產生還是按計劃產生。計劃的資料包產生時間為世界協調時凌晨 12 點。可能的值：
     - 已安排
     - 手動的
   - **spec.uploadEnabled**: (可選) 控制產生支援包後是否應將其上傳至NetApp支援網站。如果未指定，則預設為 `false`。可能的值：
     - 真的
     - false（預設值）
   - **spec.dataWindowStart**: (可選) RFC 3339 格式的日期字串，指定支援包中包含的資料視窗應開始的日期和時間。如果未指定，則預設為 24 小時前。您最早可以指定的日期範圍是 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. 填寫完後 `trident-protect-support-bundle.yaml`將檔案的值正確後，套用 CR：

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

使用 **CLI** 建立支援包

步驟

1. 建立支援包，將括號中的值替換為您環境中的資訊。這 `trigger-type`決定捆綁包是立即建立還是由計劃安排決定創建時間，並且可以是 `Manual`或者 `Scheduled`。預設是 `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`從輸出中讀取命令並運行它，將目標參數替換為您首選的本機目錄。

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 24.10 版本升級時，升級期間執行的快照可能會失敗。此故障不會阻止將來建立快照，無論是手動建立還是計劃建立。如果在升級過程中快照失敗，您可以手動建立新的快照，以確保您的應用程式受到保護。
>
> 為避免潛在的故障，您可以在升級前停用所有快照計劃，並在升級後重新啟用它們。但是，這會導致在升級期間錯過任何計劃的快照。

若要升級Trident Protect，請執行下列步驟。

步驟

1. 更新Trident Helm 倉庫：

```
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. 執行此命令以將 CRD 的管理權從 `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. 執行此命令以刪除 Helm 金鑰 `trident-protect-crds`圖表：

不要卸載 `trident-protect-crds`使用 Helm 建立圖表可能會刪除您的 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 100.2506.0 --namespace trident-protect
```

管理和保護應用程式

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。

配置 **AppVault** 身份驗證和密碼

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。

資料遷移儲存庫密碼

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，請使用 **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。

建立資料遷移儲存庫密碼金鑰

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。

- 根據您使用的資料傳輸工具，您只需輸入該資料傳輸工具對應的密碼即可。例如，如果您正在使用 Restic 並且將來不打算使用 Kopia，則在建立金鑰時，您可以只包含 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`命名空間已被刪除，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，請參閱以下範例： "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。欲了解更多信息，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 身分並將 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，請設定您的 AppVault CR。 `useIAM: true`使用標誌而不是顯式憑證：

```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
```

適用於雲端提供者的 **AppVault** 金鑰產生範例

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雲

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

亞馬遜 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
```

微軟 Azure

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

通用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 範例為每個雲端提供者建立 AppVault 物件。

- 您可以選擇指定一個 Kubernetes secret，其中包含 Restic 和 Kopia 儲存庫加密的自訂密碼。請參閱[資料遷移儲存庫密碼]了解更多。
- 對於 Amazon S3 (AWS) AppVault 對象，您可以選擇性地指定 sessionToken，如果您使用單一登入 (SSO) 進行驗證，這將非常有用。當您為提供者產生金鑰時，將建立此令牌。適用於雲端提供者的 AppVault 金鑰產生範例 。
- 對於 S3 AppVault 對象，您可以選擇性地使用下列方式指定出站 S3 流量的出口代理 URL：`spec.providerConfig.S3.proxyURL`鑰匙。

Google雲

```
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
```

亞馬遜 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，您可以移除 `providerCredentials` 部分並添加 `useIAM: true`在 `s3`改為配置。

微軟 **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
```

通用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

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

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

您可以使用以下 CLI 命令範例為每個提供者建立 AppVault CR。

- 您可以選擇指定一個 Kubernetes secret，其中包含 Restic 和 Kopia 儲存庫加密的自訂密碼。請參閱[資料遷移儲存庫密碼]了解更多。
- 對於 S3 AppVault 對象，您可以選擇性地使用下列方式指定出站 S3 流量的出口代理 URL：`--proxy-url <ip_address:port>`爭論。

Google雲

```
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
```

亞馬遜 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
```

微軟 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
```

通用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
```

查看 **AppVault** 訊息

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. 查看 AppVault 物件的內容：

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

   範例輸出：

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   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）若要查看每個資源的 AppVaultPath，請使用該標誌 `--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**

您可以隨時刪除 AppVault 物件。

> 不要移除 `finalizers`在刪除 AppVault 物件之前，請在 AppVault CR 中輸入金鑰。這樣做可能會導致 AppVault 儲存桶中殘留數據，以及叢集中出現孤立資源。

開始之前

請確保您已刪除要刪除的 AppVault 所使用的所有快照和備份 CR。

使用 **Kubernetes CLI** 刪除 **AppVault**

1. 移除 AppVault 對象，並替換 `appvault-name`要刪除的 AppVault 物件的名稱：

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

使用**Trident Protect CLI** 刪除 **AppVault**

1. 移除 AppVault 對象，並替換 `appvault-name`要刪除的 AppVault 物件的名稱：

```
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. 建立目標應用程式 CR 檔案：
   a. 建立自訂資源 (CR) 檔案並將其命名為（例如， `maria-app.yaml` ）。
   b. 配置以下屬性：
      - **metadata.name**: (必填) 應用程式自訂資源的名稱。請注意您選擇的名稱，因為保護操作所需的其他 CR 檔案會引用此值。
      - **spec.includedNamespaces**: (必要) 使用命名空間和標籤選擇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包含在此清單中。標籤選擇器是可選的，可用來篩選每個指定命名空間內的資源。
      - **spec.includedClusterScopedResources**: (可選) 使用此屬性指定要包含在應用程式定義中的叢集範圍資源。此屬性可讓您根據資源的群組、版本、種類和標籤來選擇這些資源。
        - **groupVersionKind**: (必要) 指定叢集範圍資源的 API 群組、版本和類型。
        - **labelSelector**: (可選) 依照標籤篩選叢集範圍的資源。
      - **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" 。

如果需要在應用程式建立後套用此註解，可以使用以下命令：

```
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）新增篩選條件，包含或排除帶有特定標籤的資源：

   - **resourceFilter.resourceSelectionCriteria**：（篩選時必備）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：

     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。

       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。
       - **resourceMatchers[].version**: (可選) 要篩選的資源版本。
       - **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名稱。

- **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。例如： `"trident.netapp.io/os=linux"` 。

> ⓘ 當兩者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`先運行，然後 `labelSelector`應用於生成的資源。

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. 建立與您的環境相符的應用程式變更要求後，請套用該變更要求。例如：

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

步驟

1. 使用以下範例之一建立並應用應用程式定義，將括號中的值替換為您環境中的資訊。您可以使用逗號分隔的列表，並在範例中顯示參數，將命名空間和資源包含在應用程式定義中。

   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" 。

   在使用 CLI 建立應用程式時，若要指定註解，可以使用以下方法： `--annotation`旗幟。

   - 建立應用程式並使用檔案系統凍結行為的全域設定：

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

- 建立應用程式並配置本機應用程式設定以控製檔案系統凍結行為：

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

您可以使用 `--resource-filter-include`和 `--resource-filter-exclude`用於根據以下條件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"。

建立按需快照

您可以隨時建立按需快照。

如果叢集範圍的資源在應用程式定義中被明確引用，或被引用到任何應用程式命名空間，則這些資源將包含在備份、快照或複製中。

使用 **CR** 建立快照

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-snapshot-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.applicationRef**: 要建立快照的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**: (必要) 應儲存快照內容（元資料）的 AppVault 的名稱。
   - **spec.reclaimPolicy**: (可選) 定義當快照 CR 刪除時，快照的 AppArchive 會發生什麼情況。這意味著即使設定為 `Retain`快照將會被刪除。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. 填寫完後 `trident-protect-snapshot-cr.yaml`將檔案的值正確後，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，將括號中的值替換為您環境中的資訊。例如：

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

建立按需備份

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被明確引用，或被引用到任何應用程式命名空間，則這些資源將包含在備份、快照或複製中。

開始之前

確保 AWS 會話令牌過期時間足以滿足任何長時間運行的 S3 備份作業。如果在備份作業期間令牌過期，則操作可能會失敗。

- 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
- 請參閱 "AWS IAM 文件"有關 AWS 資源憑證的詳細資訊。

使用 **CR** 建立備份

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-backup-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.applicationRef**: (必要) 要備份的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**: (必要) 應儲存備份內容的 AppVault 的名稱。
   - **spec.dataMover**: (可選) 指示要用於備份作業的備份工具的字串。可能的值（區分大小寫）：
     - `Restic`
     - `Kopia`(預設)
   - **spec.reclaimPolicy**: (可選) 定義從其聲明中釋放備份時會發生什麼。可能的值：
     - `Delete`
     - `Retain`(預設)
   - **spec.snapshotRef**: (可選): 要用作備份來源的快照名稱。如果未提供，則會建立並備份臨時快照。
   - **metadata.annotations.protect.trident.netapp.io/full-backup** : (可選) 此註解用於指定備份是否應為非增量備份。預設情況下，所有備份都是增量的。但是，如果此註釋設定為 `true`這樣，備份就變成了非增量備份。如果未指定，備份將遵循預設的增量備份設定。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。

     YAML 範例：

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

3. 填寫完後 `trident-protect-backup-cr.yaml`將檔案的值正確後，套用 CR：

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

使用命令列建立備份

步驟

1. 建立備份，將括號中的值替換為您環境中的資訊。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>
```

您也可以選擇使用 `--full-backup`用於指定備份是否應為非增量備份的標誌。預設情況下，所有備份都是增量的。使用此標誌後，備份將變為非增量備份。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。

制定資料保護計劃

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。

- 您只能透過設定來建立快照計劃。 `backupRetention`歸零和 `snapshotRetention`取大於零的值。環境 `snapshotRetention`將快照值設為零表示任何排程備份仍會建立快照，但這些快照是暫時的，會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被明確引用，或被引用到任何應用程式命名空間，則這些資源將包含在備份、快照或複製中。

使用變更請求 **(CR)** 建立計劃。

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-schedule-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.dataMover**: (可選) 指示要用於備份作業的備份工具的字串。可能的值（區分大小寫）：
     - `Restic`
     - `Kopia`(預設)
   - **spec.applicationRef**: 要備份的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**: (必要) 應儲存備份內容的 AppVault 的名稱。
   - **spec.backupRetention**：要保留的備份數量。零表示不應建立備份（僅快照）。
   - **spec.snapshotRetention**：要保留的快照數量。零表示不建立任何快照。
   - **spec.granularity**: 計畫運行的頻率。可能的值以及所需的關聯欄位：
     - `Hourly`（需要您指定） `spec.minute`）
     - `Daily`（需要您指定） `spec.minute`和 `spec.hour`）
     - `Weekly`（需要您指定） `spec.minute, spec.hour` ， 和 `spec.dayOfWeek`）
     - `Monthly`（需要您指定） `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，則此欄位為必填項。 `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，則此欄位為必填項。 `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，則此欄位為必填項。 `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，則此欄位為必填項。 `Hourly` ， `Daily` ， `Weekly` ， 或者 `Monthly`。該值必須以字串形式提供。
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (可選) 此註解用於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。

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

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. 填寫完後 `trident-protect-schedule-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 **CLI** 建立計劃任務

步驟

1. 建立保護計劃，將括號中的值替換為您環境中的資訊。例如：

您可以使用 `tridentctl-protect create schedule --help`查看此命令的詳細協助資訊。

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

您可以設定 `--full-backup-rule`標記 `always`您可以根據需要進行持續完整備份或自訂備份。例如，如果您選擇按天粒度進行備份，則可以指定應在哪些工作天進行完整備份。例如，使用 `--full-backup-rule "Monday,Thursday"`安排每週一和週四進行全面備份。

對於僅快照計劃，請設定 `--backup-retention 0`並指定一個大於 `0` 的值 `--snapshot-retention`。

刪除快照

刪除不再需要的已排程或按需快照。

步驟

1. 刪除與快照關聯的快照 CR：

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

刪除備份

刪除不再需要的計劃備份或按需備份。

確保回收策略設定為 `Delete`從物件儲存中刪除所有備份資料。此策略的預設為： `Retain`避免意外資料遺失。如果政策不改變 `Delete`備份資料將保留在物件儲存中，需要手動刪除。

步驟

1. 刪除與備份關聯的備份 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的狀態。

步驟

1. 使用以下命令檢索備份作業的狀態，將方括號中的值替換為您環境中的資訊：

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

啟用 **Azure NetApp** 檔案 **(ANF)** 作業的備份和還原

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

確保以下事項：

- 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。
- 你有 `azure-netapp-files`已選為儲存後端的預設儲存類別。

展開檢視設定步驟

1. 如果 ANF 磁碟區是在升級到Trident 24.10 之前建立的，請在Trident中執行以下操作：

   a. 為每個基於 azure-netapp-files 且與應用程式關聯的 PV 啟用快照目錄：

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

   b. 確認已為每個關聯的 PV 啟用快照目錄：

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

   回覆:

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未啟用快照目錄，則選擇常規備份功能，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。

恢復應用程式

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。

- 還原應用程式時，為該應用程式配置的所有執行鉤子都會隨應用程式一起復原。如果存在恢復後執行鉤子，它將作為恢復操作的一部分自動運行。
- 支援從備份還原到不同的命名空間或還原到原始命名空間（適用於 qtree 磁碟區）。但是，對於 qtree 卷，不支援從快照還原到不同的命名空間或還原到原始命名空間。
- 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"。

從備份還原到不同的命名空間

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。

將備份還原到具有現有資源的不同命名空間不會變更與備份中資源同名的任何資源。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，或將備份還原到新的命名空間。

開始之前

確保 AWS 會話令牌的過期時間足以滿足任何長時間運行的 S3 復原操作。如果在恢復操作期間令牌過期，則操作可能會失敗。

- 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
- 請參閱 "AWS IAM 文件"有關 AWS 資源憑證的詳細資訊。

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 "科皮亞文檔"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) 檔案並將其命名為 `trident-protect-backup-restore-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appArchivePath**: AppVault 內儲存備份內容的路徑。您可以使用以下命令尋找此路徑：

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

   - **spec.appVaultRef**: (必要) 儲存備份內容的 AppVault 的名稱。
   - **spec.namespaceMapping**：恢復作業的來源命名空間到目標命名空間的對應。代替 `my-source-namespace`和 `my-destination-namespace`利用來自周圍環境的資訊。

     ```
     ---
     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. （可選）如果您只需要選擇應用程式中的某些資源進行恢復，請新增篩選條件，以包含或排除帶有特定標籤的資源：

   > 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**：（篩選時必備）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。
       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。

- **resourceMatchers[].version**: (可選) 要篩選的資源版本。
- **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名稱。
- **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。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. 填寫完後 `trident-protect-backup-restore-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 CLI

步驟

1. 將備份還原到不同的命名空間，並將括號中的值替換為您環境中的資訊。這 `namespace-mapping`此參數使用冒號分隔的命名空間，將來源命名空間對應到正確的目標命名空間，格式如下： `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到原始命名空間。

開始之前

確保 AWS 會話令牌的過期時間足以滿足任何長時間運行的 S3 復原操作。如果在恢復操作期間令牌過期，則操作可能會失敗。

- 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
- 請參閱 "AWS IAM 文件"有關 AWS 資源憑證的詳細資訊。

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 "科皮亞文檔"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) 檔案並將其命名為 `trident-protect-backup-ipr-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appArchivePath**: AppVault 內儲存備份內容的路徑。您可以使用以下命令尋找此路徑：

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

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

     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. （可選）如果您只需要選擇應用程式中的某些資源進行恢復，請新增篩選條件，以包含或排除帶有特定標籤的資源：

   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**：（篩選時必備）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。
       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。
       - **resourceMatchers[].version**: (可選) 要篩選的資源版本。
       - **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名

稱。

- **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。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. 填寫完後 `trident-protect-backup-ipr-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 **CLI**

步驟

1. 將備份還原到原始命名空間，並將括號中的值替換為您環境中的資訊。這 `backup`參數使用命名空間和備份名稱，格式如下 `<namespace>/<name>`。例如：

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

如果原始叢集出現問題，您可以將備份還原到其他叢集。

> 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 "科皮亞文檔"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。

開始之前

確保滿足以下先決條件：

- 目標叢集已安裝Trident Protect。
- 目標叢集可以存取與來源叢集相同的 AppVault 的儲存桶路徑，備份就儲存在該儲存桶中。
- 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 會話令牌的過期時間足以滿足任何長時間運行的復原操作。如果在恢復操作期間令牌過期，則操作可能會失敗。
  - 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
  - 請參閱 "AWS 文件"有關 AWS 資源憑證的詳細資訊。

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>
```

> 確保用於應用程式還原的命名空間存在於目標叢集上。

2. 查看目標叢集中可用 AppVault 的備份內容：

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

執行此命令將顯示 AppVault 中可用的備份，包括其來源叢集、對應的應用程式名稱、時間戳記和歸檔路徑。

範例輸出：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   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到目標叢集：

使用 **CR**

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-backup-restore-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appVaultRef**: (必要) 儲存備份內容的 AppVault 的名稱。
   - **spec.appArchivePath**: AppVault 內儲存備份內容的路徑。您可以使用以下命令尋找此路徑：

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

     如果 BackupRestore CR 不可用，您可以使用步驟 2 中提到的指令來查看備份內容。

   - **spec.namespaceMapping**：恢復作業的來源命名空間到目標命名空間的對應。代替 `my-source-namespace`和 `my-destination-namespace`利用來自周圍環境的資訊。

     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. 填寫完後 `trident-protect-backup-restore-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 **CLI**

1. 使用以下命令恢復應用程序，將括號中的值替換為您環境中的資訊。命名空間映射參數使用冒號分隔的命名空間，將來源命名空間對應到正確的目標命名空間，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>
```

從快照還原到不同的命名空間

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，但僅當來源儲存類別和目標儲存類別使用相同的儲存後端時才有效。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。

開始之前

確保 AWS 會話令牌的過期時間足以滿足任何長時間運行的 S3 復原操作。如果在恢復操作期間令牌過期，則操作可能會失敗。

- 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
- 請參閱 "AWS IAM 文件"有關 AWS 資源憑證的詳細資訊。

使用 **CR**

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appVaultRef**: (必要) 儲存快照內容的 AppVault 的名稱。
   - **spec.appArchivePath**: AppVault 內儲存快照內容的路徑。您可以使用以下命令尋找此路徑：

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

   - **spec.namespaceMapping**：恢復作業的來源命名空間到目標命名空間的對應。代替 `my-source-namespace`和 `my-destination-namespace`利用來自周圍環境的資訊。

     ```
     ---
     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. （可選）如果您只需要選擇應用程式中的某些資源進行恢復，請新增篩選條件，以包含或排除帶有特定標籤的資源：

   > 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**：（篩選時必備）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。
       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。

- **resourceMatchers[].version**: (可選) 要篩選的資源版本。
- **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名稱。
- **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。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. 填寫完後 `trident-protect-snapshot-restore-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 CLI

步驟

1. 將快照還原到不同的命名空間，並將括號中的值替換為您環境中的資訊。

   - 這 `snapshot`參數使用命名空間和快照名稱，格式如下 `<namespace>/<name>`。
   - 這 `namespace-mapping`此參數使用冒號分隔的命名空間，將來源命名空間對應到正確的目標命名空間，格式如下： `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>
```

從快照還原到原始命名空間

您可以隨時將快照還原到原始命名空間。

開始之前

確保 AWS 會話令牌的過期時間足以滿足任何長時間運行的 S3 復原操作。如果在恢復操作期間令牌過期，則操作可能會失敗。

- 請參閱 "AWS API 文件"有關檢查當前會話令牌過期時間的詳細資訊。
- 請參閱 "AWS IAM 文件"有關 AWS 資源憑證的詳細資訊。

使用 **CR**

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-snapshot-ipr-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appVaultRef**: (必要) 儲存快照內容的 AppVault 的名稱。
   - **spec.appArchivePath**: AppVault 內儲存快照內容的路徑。您可以使用以下命令尋找此路徑：

```
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. （可選）如果您只需要選擇應用程式中的某些資源進行恢復，請新增篩選條件，以包含或排除帶有特定標籤的資源：

> 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**：（篩選時必備）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。
       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。
       - **resourceMatchers[].version**: (可選) 要篩選的資源版本。
       - **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名稱。
       - **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。

- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。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. 填寫完後 `trident-protect-snapshot-ipr-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 CLI

步驟

1. 將快照還原到原始命名空間，並將括號中的值替換為您環境中的資訊。例如：

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

檢查還原操作的狀態

您可以使用命令列來檢查正在進行、已完成或已失敗的還原作業的狀態。

步驟

1. 使用以下命令檢索恢復操作的狀態，將方括號中的值替換為您環境中的資訊：

```
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。

復原和故障轉移操作期間的命名空間註釋和標籤

在復原和故障轉移操作期間，目標命名空間中的標籤和註釋將與來源命名空間中的標籤和註釋相符。來源命名空間中不存在的目標命名空間中的標籤或註釋將被添加，並且任何已存在的標籤或註釋都將被覆蓋以匹配來源命名空間中的值。僅存在於目標命名空間中的標籤或註解保持不變。

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 安全上下文約束文檔"。

您可以透過設定 Kubernetes 環境變數來防止目標命名空間中的特定註解被覆蓋。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在執行復原或故障轉移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_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 "配置AutoSupport和命名空間過濾選項"。

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。

例子

以下範例展示了來源命名空間和目標命名空間，每個命名空間都有不同的註解和標籤。您可以查看操作前後目標命名空間的狀態，以及目標命名空間中的註解和標籤是如何組合或覆蓋的。

在恢復或故障轉移操作之前

下表說明了復原或故障轉移操作之前範例來源命名空間和目標命名空間的狀態：

命名空間	註解	標籤
命名空間 ns-1（來源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 環境=生產 • 合規性=HIPAA • 名稱=ns-1
命名空間 ns-2（目標）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=資料庫

恢復操作後

下表說明了復原或故障轉移作業後範例目標命名空間的狀態。有些按鍵已被添加，有些按鍵已被覆蓋，並且 `name`標籤已更新，以符合目標命名空間：

命名空間	註解	標籤
命名空間 ns-2（目標）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名稱=ns-2 • 合規性=HIPAA • 環境=生產 • 角色=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`屬性定義了從來源應用程式中存在的儲存類別到目標叢集上新儲存類別的對應。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。

例：

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

支持的註釋

本節列出了系統中用於配置各種行為的支援註解。如果使用者沒有明確設定註釋，系統將使用預設值。

註解	類型	描述	預設值
protect.trident.netapp.io/data-mover-timeout-sec	細繩	資料移動器操作允許停止的最長時間（以秒為單位）。	“300”
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	細繩	Kopia 內容快取的最大大小限制（以兆位元組為單位）。	“1000”

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。

復原和故障轉移操作期間的命名空間註釋和標籤

在復原和故障轉移操作期間，目標命名空間中的標籤和註釋將與來源命名空間中的標籤和註釋相符。來源命名空間中不存在的目標命名空間中的標籤或註釋將被添加，並且任何已存在的標籤或註釋都將被覆蓋以匹配來源命名空間中的值。僅存在於目標命名空間中的標籤或註解保持不變。

> ⓘ 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 安全上下文約束文檔"。

您可以透過設定 Kubernetes 環境變數來防止目標命名空間中的特定註解被覆蓋。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在執行復原或故障轉移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_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 "配置AutoSupport和命名空間過濾選項"。

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。

例子

以下範例展示了來源命名空間和目標命名空間，每個命名空間都有不同的註解和標籤。您可以查看操作前後目標

命名空間的狀態，以及目標命名空間中的註解和標籤是如何組合或覆蓋的。

在恢復或故障轉移操作之前

下表說明了復原或故障轉移操作之前範例來源命名空間和目標命名空間的狀態：

命名空間	註解	標籤
命名空間 ns-1（來源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 環境=生產 • 合規性=HIPAA • 名稱=ns-1
命名空間 ns-2（目標）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=資料庫

恢復操作後

下表說明了復原或故障轉移作業後範例目標命名空間的狀態。有些按鍵已被添加，有些按鍵已被覆蓋，並且 `name`標籤已更新，以符合目標命名空間：

命名空間	註解	標籤
命名空間 ns-2（目標）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名稱=ns-2 • 合規性=HIPAA • 環境=生產 • 角色=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鉤子

使用 AppMirror 關係保護應用程式時，在故障轉移和反向操作期間，您應該注意與執行鉤子相關的特定行為。

- 故障轉移期間，執行鉤子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，應用程式上會存在執行鉤子，這些鉤子將在任何相關操作期間執行。
- 在反向同步或反向重同步期間，應用程式上任何現有的執行鉤子都會被移除。當來源應用程式變為目標應用程式時，這些執行鉤子將不再有效，並將被刪除以防止其執行。

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"。

建立複製關係

建立複製關係涉及以下步驟：

- 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）。

- 選擇複製計劃（包括 Kubernetes 資源以及持久卷資料）
- 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一個 AppVault。根據您的儲存供應商，修改範例中的內容。"AppVault 自訂資源"為了適應您的環境：

使用 **CR** 建立 **AppVault**

a. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-appvault-primary-source.yaml` ）。

b. 配置以下屬性：

- **metadata.name**: (必填) AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複製關係所需的其他 CR 檔案會引用此值。
- **spec.providerConfig**: (必要) 儲存使用指定提供者存取 AppVault 所需的設定。選擇儲存桶名稱以及提供者所需的其他任何詳細資訊。請記下您選擇的值，因為複製關係所需的其他 CR 檔案會引用這些值。請參閱"AppVault 自訂資源"例如，AppVault CR 與其他提供者的合作案例。
- **spec.providerCredentials**: (*Required*) 儲存使用指定提供者存取 AppVault 所需的任何憑證的參考。
  - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示憑證值應來自金鑰。
    - **key**: (必填) 要從中選取的有效金鑰。
    - **name**: (必填) 包含此欄位值的金鑰的名稱。必須位於同一命名空間中。
  - **spec.providerCredentials.secretAccessKey**: (必要) 用於存取提供者的存取金鑰。 名稱應與 **spec.providerCredentials.valueFromSecret.name** 相符。
- **spec.providerType**: (必要) 決定備份的提供者；例如， NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能的值：
  - AWS
  - 蔚藍
  - 通用控制協議
  - 通用-s3
  - ontap-s3
  - 儲存網格-s3

c. 填寫完後 `trident-protect-appvault-primary-source.yaml`將檔案的值正確後，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，並將括號中的值替換為您環境中的資訊：

```
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) 檔案並將其命名為（例如， `trident-protect-app-source.yaml` ）。

b. 配置以下屬性：

- **metadata.name**: (必填) 應用程式自訂資源的名稱。請記下您選擇的名稱，因為複製關係所需的其他 CR 檔案會引用此值。
- **spec.includedNamespaces**: (必要) 命名空間及其關聯標籤的陣列。使用命名空間名稱，並可選擇使用標籤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包含在此數組中。

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. 填寫完後 `trident-protect-app-source.yaml`將檔案的值正確後，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 "建立按需快照"。

4. 在來源叢集上，建立複製計劃 CR：

除了下面提供的計劃之外，建議建立一個單獨的每日快照計劃，保留期為 7 天，以在對等ONTAP叢集之間保持共同的快照。這樣可以確保快照最多保留 7 天，但保留期限可以根據使用者需求進行自訂。

如果發生故障轉移，系統可以使用這些快照最多 7 天進行逆向操作。這種方法使得逆向過程更快、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，則無需制定其他計劃。

使用 **CR** 建立複製計劃

a. 為來源應用程式建立複製計劃：

   i. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-schedule.yaml` ）。

   ii. 配置以下屬性：

   - **metadata.name**: (必填) 計畫自訂資源的名稱。
   - **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*) 此欄位為必填項，其值必須設為 0。
   - **spec.enabled**：必須設定為 true。
   - **spec.granularity**：必須設定為 `Custom`。
   - **spec.recurrenceRule**: 定義 UTC 時間的開始日期和重複間隔。
   - **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. 填寫完後 `trident-protect-schedule.yaml`將檔案的值正確後，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>
```

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. 在目標叢集上，建立一個與在來源叢集上應用的 AppVault CR 完全相同的來源應用程式 AppVault CR，並將其命名為（例如， `trident-protect-appvault-primary-destination.yaml` ）。

6. 應用 CR：

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

7. 在目標叢集上為目標應用程式建立目標 AppVault CR。根據您的儲存供應商，修改範例中的內容。"AppVault 自訂資源"為了適應您的環境：

   a. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-appvault-secondary-destination.yaml` ）。

   b. 配置以下屬性：

      - **metadata.name**: (必填) AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複製關係所需的其他 CR 檔案會引用此值。
      - **spec.providerConfig**: (必要) 儲存使用指定提供者存取 AppVault 所需的設定。選擇一個 `bucketName`以及其他任何您需要提供給服務提供者的詳細資訊。請記下您選擇的值，因為複製關係所需的其他 CR 檔案會引用這些值。請參閱"AppVault 自訂資源"例如，AppVault CR 與其他提供者的合作案例。
      - **spec.providerCredentials**: (*Required*) 儲存使用指定提供者存取 AppVault 所需的任何憑證的參考。
        - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示憑證值應來自金鑰。
          - **key**: (必填) 要從中選取的有效金鑰。
          - **name**: (必填) 包含此欄位值的金鑰的名稱。必須位於同一命名空間中。
        - **spec.providerCredentials.secretAccessKey**: (必要) 用於存取提供者的存取金鑰。 名稱 應與 **spec.providerCredentials.valueFromSecret.name** 相符。

- **spec.providerType**: (必要) 決定備份的提供者；例如， NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能的值：
  - AWS
  - 蔚藍
  - 通用控制協議
  - 通用-s3
  - ontap-s3
  - 儲存網格-s3

c. 填寫完後 `trident-protect-appvault-secondary-destination.yaml`將檔案的值正確後，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** 建立 **AppMirrorRelationship**

a. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-relationship.yaml` ）。

b. 配置以下屬性：

- **metadata.name:**（必填）AppMirrorRelationship 自訂資源的名稱。
- **spec.destinationAppVaultRef**: (必要) 此值必須與目標叢集上目標應用程式的 AppVault 名稱相符。
- **spec.namespaceMapping**: (必要) 目標命名空間和來源命名空間必須與對應應用程式 CR 中定義的應用程式命名空間相符。
- **spec.sourceAppVaultRef**: (必要) 此值必須與來源應用程式的 AppVault 名稱相符。
- **spec.sourceApplicationName**: (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名稱相符。
- **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**: 定義 UTC 時間的開始日期和重複間隔。

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. 填寫完後 `trident-protect-relationship.yaml`將檔案的值正確後，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>
```

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. （可選）在目標叢集上，檢查複製關係的狀態：

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

故障轉移到目標集群

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文件。

3. 應用 CR：

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

4. （可選）在故障轉移應用程式上建立所需的任何保護計劃。
5. （可選）檢查複製關係的狀態：

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

重新同步失敗的複製關係

重新同步操作會重新建立複製關係。執行重新同步操作後，原始來源應用程式將成為正在運行的應用程序，對目標叢集上正在運行的應用程式所做的任何更改都將被丟棄。

該過程會在重新建立複製之前停止目標叢集上的應用程式。

故障轉移期間寫入目標應用程式的任何資料都會遺失。

步驟

1. （可選）在來源叢集上，建立來源應用程式的快照。這樣可以確保捕獲源集群的最新變更。
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文件。
4. 應用 CR：

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

5. 如果您在目標叢集上建立了任何保護計劃來保護故障轉移應用程序，請將其刪除。任何殘留的計劃都會導致磁碟區快照失敗。

反向重新同步失敗的複製關係

當您反向同步故障轉移複製關係時，目標應用程式將變為來源應用程序，而來源應用程式將變為目標應用程式。故障轉移期間對目標應用程式所做的變更將被保留。

步驟

1. 在原始目標叢集上，刪除 AppMirrorRelationship CR。這導致目的地變成了出發地。如果新目標叢集上還有任何剩餘的保護計劃，請將其刪除。
2. 透過將最初用於建立關係的 CR 檔案套用到相反的叢集來建立複製關係。
3. 確保新目標（原始來源叢集）配置了兩個 AppVault CR。
4. 在相反的集群上建立複製關係，並配置反向的值。

反向應用程式複製方向

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，你交換了來源位址和目標位址。

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策略計劃。

b. 建立 ShutdownSnapshot CR 檔案：

i. 建立自訂資源 (CR) 檔案並將其命名為（例如， `trident-protect-shutdownsnapshot.yaml` ）。

ii. 配置以下屬性：

- **metadata.name**: (必填) 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**: (必要) 此值必須與來源應用程式 CR 檔案的 metadata.name 欄位相符。

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. 填寫完後 `trident-protect-shutdownsnapshot.yaml`將檔案的值正確後，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，將括號中的值替換為您環境中的資訊。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，使用以下命令尋找 **shutdownsnapshot.status.appArchivePath** 的值，並記錄檔案路徑的最後一部分（也稱為基本名稱；這將是最後一個斜線之後的所有內容）：

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

4. 從新的目標叢集到新的來源叢集執行故障轉移，並進行以下變更：

> 在故障轉移流程的第 2 步驟中，包括：`spec.promotedSnapshot`在 AppMirrorRelationship CR 檔案中，將該欄位的值設定為您在上面的步驟 3 中記錄的基本名稱。

5. 執行反向重新同步步驟[反向重新同步失敗的複製關係]。
6. 在新來源叢集上啟用保護計劃。

結果

由於反向複製，會發生以下操作：

- 對原始來源應用程式的 Kubernetes 資源進行快照。
- 透過刪除應用程式的 Kubernetes 資源（保留 PVC 和 PV），優雅地停止原始來源應用程式的 pod。
- 在 pod 關閉後，會對應用程式的磁碟區進行快照並進行複製。
- SnapMirror關係已斷開，目標磁碟區已準備好進行讀取/寫入操作。
- 該應用程式的 Kubernetes 資源是從關閉前的快照中恢復的，使用的是在原始來源應用程式關閉後複製的捲資料。
- 複製過程以相反的方向重新建立。

將應用程式故障恢復到原始來源集群

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。

流程從已完成故障轉移至目標位置的關係開始，並涉及以下步驟：

- 從故障轉移狀態開始。
- 反向重新同步複製關係。

> 不要執行正常的重新同步操作，因為這將丟棄在故障轉移過程中寫入目標叢集的資料。

- 反轉複製方向。

步驟

1. 執行[反向重新同步失敗的複製關係]步驟。
2. 執行[反向應用程式複製方向]步驟。

刪除複製關係

您可以隨時刪除複製關係。刪除應用程式複製關係後，將產生兩個彼此獨立的應用程序，它們之間沒有任何關係。

步驟

1. 在目前目標叢集上，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。

遷移應用程式時，為該應用程式配置的所有執行鉤子都會隨應用程式一起遷移。如果存在恢復後執行鉤子，它將作為恢復操作的一部分自動運行。

備份和復原作業

針對以下場景，您可以自動執行特定的備份和復原任務，以執行備份和復原作業。

克隆到同一群集

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

步驟

1. 執行下列操作之一：
   a. "建立快照"。
   b. "建立備份"。
2. 在同一群集上，根據您建立的是快照還是備份，執行下列其中一項：
   a. "從快照恢復數據"。
   b. "從備份中恢復數據"。

克隆到不同的集群

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。

您可以使用以下方法在不同的叢集之間複製應用程式"SnapMirror複製"。

步驟

1. "建立備份"。
2. 確保目標叢集上已配置包含備份的物件儲存桶的 AppVault CR。
3. 在目標集群上，"從備份中恢復數據" 。

將應用程式從一個儲存類別遷移到另一個儲存類

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（不包括恢復 CR 中的金鑰）：

```
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** 恢復快照

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的文件中，配置以下屬性：
   - **metadata.name**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.appArchivePath**: AppVault 內儲存快照內容的路徑。您可以使用以下命令尋找此路徑：

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

   - **spec.appVaultRef**: (必要) 儲存快照內容的 AppVault 的名稱。
   - **spec.namespaceMapping**：恢復作業的來源命名空間到目標命名空間的對應。代替 `my-source-namespace`和 `my-destination-namespace`利用來自周圍環境的資訊。

     ```
     ---
     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. 如果您需要僅選擇要復原的應用程式的某些資源，則可以新增篩選條件，以包含或排除具有特定標籤的資源：
   - **resourceFilter.resourceSelectionCriteria**：（篩選時必備）使用 `include or exclude`包含或排除 resourceMatchers 中定義的資源。新增以下 resourceMatchers 參數以定義要包含或排除的資源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 物件陣列。如果在該數組中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的字段（組、種類、版本）之間按 AND 運算匹配。
       - **resourceMatchers[].group**: (可選) 要篩選的資源群組。
       - **resourceMatchers[].kind**: (可選) 要篩選的資源類型。
       - **resourceMatchers[].version**: (可選) 要篩選的資源版本。
       - **resourceMatchers[].names**: (可選) 要過濾的資源的 Kubernetes 元資料.name 欄位中的名稱。
       - **resourceMatchers[].namespaces**: (可選) 要篩選的資源的 Kubernetes 元資料.name 欄位中的命名空間。

- **resourceMatchers[].labelSelectors**: (可選) 資源的 Kubernetes 元資料.name 欄位中的標籤選擇器字串，如在下列位置定義： "Kubernetes 文檔" 。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. 填寫完後 `trident-protect-snapshot-restore-cr.yaml`將檔案的值正確後，套用 CR：

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

使用 **CLI** 恢復快照

步驟

1. 將快照還原到不同的命名空間，並將括號中的值替換為您環境中的資訊。

   - 這 `snapshot`參數使用命名空間和快照名稱，格式如下 `<namespace>/<name>`。
   - 這 `namespace-mapping`此參數使用冒號分隔的命名空間，將來源命名空間對應到正確的目標命名空間，格式如下： `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

執行鉤子是一種自訂操作，您可以將其配置為與受管應用程式的資料保護操作一起運行。例如，如果您有一個資料庫應用程序，則可以使用執行掛鉤在快照之前暫停所有資料庫事務，並在快照完成後恢復事務。這確保了應用程式一致的快照。

執行鉤子的類型

Trident Protect 支援以下幾種執行鉤子類型，取決於它們的運行時機：

- 預快照
- 快照後
- 預備份
- 備份後
- 恢復後
- 故障轉移後

執行順序

當執行資料保護操作時，執行掛鉤事件會依照下列順序發生：

1. 任何適用的自訂預操作執行掛鉤都在適當的容器上運行。您可以根據需要建立和運行任意數量的自訂預操作掛鉤，但這些掛鉤在操作之前的執行順序既無法保證也無法配置。
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. 執行資料保護操作。
4. 如果適用，凍結的檔案系統將被解凍。
5. 任何適用的自訂後操作執行掛鉤都在適當的容器上運行。您可以根據需要建立和運行任意數量的自訂後操作掛鉤，但操作後這些掛鉤的執行順序既無法保證也無法配置。

如果您建立多個相同類型的執行掛鉤（例如，預快照），則無法保證這些掛鉤的執行順序。但是，不同類型的鉤子的執行順序是有保證的。例如，以下是具有所有不同類型鉤子的配置的執行順序：

1. 快照前鉤子執行
2. 快照後鉤子執行
3. 執行備份前掛鉤
4. 執行備份後鉤子

前面的順序範例僅適用於執行不使用現有快照的備份時。

在生產環境中啟用執行掛鉤腳本之前，您應該始終對其進行測試。您可以使用“kubectl exec”命令方便地測試腳本。在生產環境中啟用執行掛鉤後，測試產生的快照和備份以確保它們一致。您可以透過將應用程式複製到臨時命名空間、還原快照或備份，然後測試應用程式來執行此操作。

如果快照前執行鉤子新增、變更或刪除 Kubernetes 資源，則這些變更將包含在快照或備份以及任何後續復原作業中。

關於自訂執行鉤子的重要說明

在為您的應用程式規劃執行掛鉤時，請考慮以下事項。

- 執行鉤子必須使用腳本來執行操作。許多執行鉤子可以引用同一個腳本。
- 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。
- 腳本大小限制為 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。

由於執行鉤子通常會減少或完全停用其所針對的應用程式的功能，因此您應該始終嘗試盡量減少自訂執行鉤子的運行時間。如果您啟動具有相關執行掛鉤的備份或快照操作，但隨後取消它，則如果備份或快照操作已經開始，則仍允許掛鉤運行。這意味著備份後執行掛鉤中使用的邏輯不能假定備份已完成。

執行鉤子過濾器

當您為應用程式新增或編輯執行掛鉤時，您可以向執行掛鉤添加過濾器來管理該掛鉤將匹配哪些容器。過濾器對於在所有容器上使用相同容器鏡像但可能將每個鏡像用於不同目的的應用程式（例如 Elasticsearch）很有用。過濾器可讓您建立執行掛鉤在某些（但不一定是所有）相同的容器上運行的場景。如果為單一執行掛鉤建立多個篩選器，它們將透過邏輯 AND 運算子組合在一起。每個執行掛鉤最多可以有 10 個活動過濾器。

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 "正規表示式 2 (RE2) 語法支持"。

如果將命名空間過濾器新增至在復原或複製作業後執行的執行掛鉤，且復原或複製來源和目標位於不同的命名空間中，則命名空間篩選器僅適用於目標命名空間。

執行鉤子範例

訪問 "NetApp Verda GitHub 項目"下載流行應用程式（如 Apache Cassandra 和 Elasticsearch）的真實執行掛鉤。您還可以查看範例並獲得建立自己的自訂執行掛鉤的想法。

建立執行鉤子

您可以使用以下方法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) 檔案並將其命名為 `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**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.applicationRef**: (必要) 要執行執行鉤子的應用程式的 Kubernetes 名稱。
   - **spec.stage**: (*Required*) 一個字串，指示執行鉤子應該在操作的哪個階段運行。可能的值：
     - 預
     - 郵政
   - **spec.action**: (*Required*) 一個字串，指示執行鉤子將採取什麼操作，假設指定的任何執行鉤子過濾器都匹配。可能的值：
     - 快照
     - 備份
     - 恢復
     - 故障轉移
   - **spec.enabled**: (可選) 指示此執行鉤子是否已啟用或停用。如果未指定，則預設值為 true。
   - **spec.hookSource**: (必要) 包含 base64 編碼的 hook 腳本的字串。
   - **spec.timeout**: (可選) 定義執行鉤子允許運行的分鐘數的數字。最小值為 1 分鐘，如果未指定，則預設值為 25 分鐘。
   - **spec.arguments**: (可選) 一個 YAML 列表，用於指定執行鉤子的參數。
   - **spec.matchingCriteria**: (可選) 一個可選的條件鍵值對列表，每個鍵值對構成一個執行鉤子過濾器。每個執行鉤子最多可以添加 10 個過濾器。
   - **spec.matchingCriteria.type**: (可選) 用於識別執行鉤過濾器類型的字串。可能的值：
     - 容器影像
     - 容器名稱
     - Pod名稱
     - PodLabel
     - 命名空間名稱
   - **spec.matchingCriteria.value**: (可選) 用於識別執行鉤過濾器值的字串或正規表示式。

     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. 在用正確的值填入 CR 檔案後，套用 CR：

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

使用 CLI

步驟

1. 建立執行鉤子，將括號中的值替換為您環境中的資訊。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>
```

手動運行執行鉤子

您可以手動執行執行鉤子進行測試，或者在失敗後需要手動重新運行鉤子時也可以這樣做。您需要擁有所有者、管理員或成員權限才能手動執行執行鉤子。

手動運行執行鉤子包含兩個基本步驟：

1. 建立資源備份，該備份會收集資源並建立它們的備份，從而確定鉤子函數的運作位置。
2. 針對備份運行執行鉤子

步驟 **1**：建立資源備份

使用 **CR**

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `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**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.applicationRef**: (必要) 要為其建立資源備份的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**: (必要) 儲存備份內容的 AppVault 的名稱。
   - **spec.appArchivePath**: AppVault 內儲存備份內容的路徑。您可以使用以下命令尋找此路徑：

     ```
     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. 在用正確的值填入 CR 檔案後，套用 CR：

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

使用 **CLI**

步驟

1. 建立備份，將括號中的值替換為您環境中的資訊。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. 查看備份狀態。您可以重複使用此範例命令，直到操作完成：

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

3. 確認備份是否成功：

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

步驟 **2**：運行執行鉤子

使用 **CR**

步驟

1. 建立自訂資源 (CR) 檔案並將其命名為 `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**: (必填) 此自訂資源的名稱；請為您的環境選擇一個唯一且有意義的名稱。
   - **spec.applicationRef**: (必需) 確保此值與您在步驟 1 中建立的 ResourceBackup CR 中的應用程式名稱相符。
   - **spec.appVaultRef**: (必需) 確保此值與您在步驟 1 中建立的 ResourceBackup CR 中的 appVaultRef 相符。
   - **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*) 一個字串，指示執行鉤子將採取什麼操作，假設指定的任何執行鉤子過濾器都匹配。可能的值：
     - 快照
     - 備份
     - 恢復
     - 故障轉移
   - **spec.stage**: (*Required*) 一個字串，指示執行鉤子應該在操作的哪個階段運行。這次鉤子運行不會在任何其他階段運行鉤子。可能的值：
     - 預
     - 郵政

     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. 在用正確的值填入 CR 檔案後，套用 CR：

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

使用 **CLI**

步驟

1. 建立手動執行鉤子運行請求：

```
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. 檢查執行鉤子運行狀態。您可以重複執行此命令，直到操作完成：

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

3. 描述 exechooksrun 物件以查看最終詳細資訊和狀態：

```
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
```

版權資訊

商標資訊