管理和保護應用程式
Trident

NetApp
April 08, 2026

目錄

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式 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）中所需的設定。只有管理員才能建立 AppVaults。

在對應用程式執行資料保護操作時，您需要手動或透過命令列建立 AppVault CR。AppVault CR 是針對您的環境而設定的，您可以參考此頁面上的範例來建立 AppVault CR。

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

設定 **AppVault** 驗證和密碼

在建立 AppVault CR 之前，請確保 AppVault 和您選擇的資料移動程式能夠透過提供者和任何相關資源進行身份驗證。

Data mover 儲存庫密碼

當您使用 CR 或 Trident Protect CLI 外掛程式建立 AppVault 物件時，您可以指定 Kubernetes Secret，其中包含用於 Restic 和 Kopia 加密的自訂密碼。如果您未指定 Secret，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`參數指定金鑰。

建立資料移動器儲存庫密碼密鑰

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

- 根據您使用的資料遷移工具，您只需包含該資料遷移工具對應的密碼即可。例如，如果您使用的是 Restic，並且將來不打算使用 Kopia，則在建立 secret 時只需包含 Restic 的密碼即可。
- 請妥善保管密碼。您需要使用此密碼在同一叢集或其他叢集上還原資料。如果叢集或 `trident-protect`命名空間被刪除，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 相容儲存 **IAM** 權限

當您使用 Trident Protect 存取 S3 相容儲存（例如 Amazon S3、Generic S3 "StorageGrid S3"或 "ONTAP S3"）時，您需要確保所提供的使用者認證具有存取儲存貯體的必要權限。以下是授予使用 Trident Protect 存取所需最低權限的原則範例。您可以將此原則套用至管理 S3 相容儲存貯體原則的使用者。

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

有關 Amazon S3 策略的更多資訊，請參閱 "Amazon S3 文件" 中的範例。

用於 **Amazon S3**（**AWS**）驗證的 **EKS Pod Identity**

Trident Protect 支援 EKS Pod Identity 進行 Kopia 資料遷移操作。此功能可安全存取 S3 儲存貯體，而無需在 Kubernetes Secret 中儲存 AWS 認證。

使用 **Trident Protect** 的 **EKS Pod Identity** 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 S3 儲存桶權限的 IAM 角色。如需深入瞭解，請參閱 "S3 相容儲存 IAM 權限"。
- IAM 角色與下列 Trident Protect 服務帳戶相關聯：
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

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

AppVault 設定 使用 EKS Pod Identity 時，請使用 `useIAM: true`標誌設定 AppVault CR，而不是明確憑證：

```
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
```

AppVault 雲端服務提供者的金鑰產生範例

定義 AppVault CR 時，除非使用 IAM 驗證，否則您需要包含存取提供者託管資源的憑證。憑證金鑰的產生方式因提供者而異。以下是幾個提供者的命令列金鑰產生範例。您可以使用以下範例為每個雲端提供者的憑證建立金鑰。

Google Cloud

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

Amazon S3（AWS）

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

Microsoft Azure

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

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

以下是各提供者的 AppVault 定義範例。

AppVault CR 範例

您可以使用以下 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。如需更多資訊，請參閱 Data mover 儲存庫密碼。
- 對於 Amazon S3 (AWS) AppVault 物件，您可以選擇性地指定 sessionToken，如果您使用單一登入（SSO）進行身份驗證，這將非常有用。此令牌在您為提供者產生金鑰時建立 AppVault 雲端服務提供者的金鑰產生範例。
- 對於 S3 AppVault 對象，您可以使用 `spec.providerConfig.S3.proxyURL` 鍵選擇性地指定出站 S3 流量的出口代理 URL。

Google Cloud

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

Amazon S3（AWS）

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

對於使用 Pod Identity 與 Kopia data mover 的 EKS 環境，您可以移除 `providerCredentials`區段，並將 `useIAM: true`新增到 `s3`設定下方。

Microsoft Azure

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

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。如需更多資訊，請參閱 Data mover 儲存庫密碼。
- 對於 S3 AppVault 對象，您可以使用 `--proxy-url <ip_address:port>`參數選擇性地指定出站 S3 流量的出口代理 URL。

Google Cloud

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

Amazon S3（AWS）

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

Microsoft Azure

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

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
```

支援的 `providerConfig.s3` 配置選項

請參閱下表以了解 S3 提供者組態選項：

參數	說明	預設	範例
`providerConfig.s3.skipCertValidation`	停用 SSL/TLS 憑證驗證。	錯誤	"true"、"false"
`providerConfig.s3.secure`	啟用與 S3 端點的安全 HTTPS 通訊。	true	"true"、"false"
`providerConfig.s3.proxyURL`	指定用於連接到 S3 的 Proxy 伺服器 URL。	無	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	提供用於 SSL/TLS 驗證的自訂根 CA 憑證。	無	"CN=MyCustomCA"
`providerConfig.s3.useIAM`	啟用 IAM 驗證以存取 S3 儲存桶。適用於 EKS Pod Identity。	錯誤	true、false

查看 AppVault 資訊

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

步驟

1. 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，請使用標誌 `--show-paths`。

只有在 Trident Protect Helm 安裝過程中指定了叢集名稱時，表格第一列中的叢集名稱才可用。例如：`--set clusterName=production1`。

移除 AppVault

您可以隨時移除 AppVault 物件。

在刪除 AppVault 物件之前，請勿移除 AppVault CR 中的 `finalizers`金鑰。否則，可能會導致 AppVault 儲存桶中殘留資料，並在叢集中產生孤立資源。

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. 刪除 AppVault 物件，將 `appvault-name`替換為要刪除的 AppVault 物件名稱：

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

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

1. 刪除 AppVault 物件，將 `appvault-name`替換為要刪除的 AppVault 物件名稱：

```
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作業時使用，且該 AppVault CR 必須位於安裝了 Trident Protect 的叢集上。AppVault CR 特定於您的環境；有關 AppVault CR 的範例，請參閱"AppVault 自訂資源。"

定義應用程式

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

使用 **CR** 新增應用程式

步驟

1. 建立目的地應用程式 CR 檔案：
   a. 建立自訂資源（CR）檔案並將其命名為（例如、 `maria-app.yaml`）。
   b. 設定下列屬性：
      - **metadata.name**：（必填）應用程式自訂資源的名稱。請記住您選擇的名稱，因為保護作業所需的其他 CR 檔案會參照此值。
      - **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中的資源。
      - **spec.includedClusterScopedResources**：（選用）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、類型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器定義的應用程式，例如在 KubeVirt 環境中，快照之前會發生檔案系統凍結。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，則應用程式會忽略全域設定，並可在快照期間寫入檔案系統。如果設定為 false，則應用程式會忽略全域設定，且檔案系統會在快照期間凍結。如果已指定但應用程式定義中沒有虛擬機器，則會忽略該註解。如果未指定，則應用程式會遵循"全球 Trident Protect 凍結設定"。

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. （選用）新增篩選條件，包含或排除標記有特定標籤的資源：

   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `Include`或 `Exclude`來包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位中的名稱。

- **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

> ⓘ 當兩者 `resourceFilter` 和 `labelSelector` 同時使用時，`resourceFilter` 首先運行，然後 `labelSelector` 將應用於生成的資源。

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 CR 後、套用該 CR。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，並將括號中的值替換為您環境中的資訊。您可以使用範例中所示的參數，以逗號分隔的清單在應用程式定義中包含命名空間和資源。

   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器定義的應用程式、例如在 KubeVirt 環境中、快照之前會發生檔案系統凍結。如果將註解設為 `true`、則應用程式會忽略全域設定、並可在快照期間寫入檔案系統。如果將其設為 `false`、則應用程式會忽略全域設定、且檔案系統會在快照期間凍結。如果您使用註解、但應用程式定義中沒有虛擬機器、則會忽略該註解。如果您不使用註解、則應用程式會遵循"全球 Trident Protect 凍結設定"。

   使用 CLI 建立應用程式時，若要指定註解，可以使用 `--annotation` 標誌。

- 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>
```

- 建立應用程式並設定本機應用程式設定以控制檔案系統凍結行為：

```
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` 標誌來根據 `resourceSelectionCriteria` 群組、種類、版本、標籤、名稱和命名空間等條件包含或排除資源，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原則或臨機方式拍攝快照和備份，以保護 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及解除凍結檔案系統。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。

建立隨需快照

您可以隨時建立隨需快照。

如果叢集範圍的資源在應用程式定義中被明確引用，或引用了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. 在您建立的檔案中、設定以下屬性：
   - **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**：（必填）應儲存快照內容（元資料）的 AppVault 名稱。
   - **spec.reclaimPolicy**：（選用）定義當快照 CR 刪除時，快照的 AppArchive 會發生什麼情況。這意味著即使設為 `Retain`，快照也會被刪除。有效選項：
     - `Retain` (預設)
     - `Delete`

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

3. 在 `trident-protect-snapshot-cr.yaml` 檔案中填入正確的值後，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，並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>
```

建立隨需備份

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，或引用了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

請確保 AWS 會話令牌的有效期限足以應付任何長時間運行的 s3 備份作業。如果令牌在備份作業期間過期，則操作可能會失敗。

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

使用 **CR** 建立備份

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-backup-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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**：（必需）要備份的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**：（必填）應儲存備份內容的 AppVault 名稱。
   - **spec.dataMover**：（選用）一個字串，指示要用於備份作業的備份工具。可能的值（區分大小寫）：
     - `Restic`
     - `Kopia` (預設)
   - **spec.reclaimPolicy**：（選用）定義備份從其聲明中釋放後會發生什麼。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，則會建立並備份一個臨時快照。

     YAML 範例：

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

3. 在 `trident-protect-backup-cr.yaml` 檔案中填入正確的值後，套用 CR：

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

使用 **CLI** 建立備份

步驟

1. 建立備份，並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>
```

您可以選擇使用 `--full-backup` 標誌來指定備份是否為非增量備份。預設情況下，所有備份均為增量備份。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，並在兩次完整備份之間執行增量備份，以最大程度地降低還原風險。

支援的備份註釋

下表說明建立備份 CR 時可使用的註解：

註解	類型	說明	預設值
protect.trident.netapp.io/full-backup	字串	指定備份是否為非增量備份。設定為 `true`可建立非增量備份。最佳實踐是定期執行完整備份，並在兩次完整備份之間執行增量備份，以最大程度地降低還原相關風險。	"false"
protect.trident.netapp.io/snapshot-completion-timeout	字串	允許完成整個快照作業的最長時間。	"60 分鐘"
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	字串	磁碟區快照達到可用狀態所允許的最長時間。	"30 分鐘"
protect.trident.netapp.io/volume-snapshots-created-timeout	字串	建立磁碟區快照所允許的最長時間。	"5 分鐘"
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待所有新建立的 PersistentVolumeClaims（PVC）達到 `Bound`階段的最長時間（以秒為單位），超過此時間操作將會失敗。	"1200"（20 分鐘）

建立資料保護排程

保護原則透過按定義的排程建立快照、備份或兩者來保護應用程式。您可以選擇每小時、每天、每週和每月建立快照和備份，並可指定要保留的副本數量。您可以使用 full-backup-rule 註解來排程非增量完整備份。預設情況下，所有備份都是增量備份。定期執行完整備份以及其間的增量備份，有助於降低與還原相關的風險。

- 您可以透過將 `backupRetention`設為零並將 `snapshotRetention`設為大於零的值來建立僅用於快照的排程。將 `snapshotRetention`設為零表示任何排程的備份仍會建立快照，但這些快照是暫時的，並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，或引用了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** 建立排程

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-schedule-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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**：（選用）一個字串，指示要用於備份作業的備份工具。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**：（必填）應儲存備份內容的 AppVault 名稱。
   - **spec.backupRetention**：（必填）要保留的備份數量。零表示不應建立任何備份（僅快照）。
   - **spec.backupReclaimPolicy**：（選用）決定如果在保留期間內刪除備份 CR 時，備份會發生什麼情況。保留期間過後，備份一律會被刪除。可能的值（區分大小寫）：
     - `Retain` (預設)
     - `Delete`
   - **spec.snapshotRetention**：（必填）要保留的快照數量。零表示不建立任何快照。
   - **spec.snapshotReclaimPolicy**：（選用）決定在保留期間內刪除快照 CR 時，快照會發生什麼情況。保留期間過後，快照一律會被刪除。可能的值（區分大小寫）：
     - `Retain`
     - `Delete` (預設)
   - **spec.granularity**：排程應執行的頻率。可能的值以及相關的必填欄位：
     - `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 、 spec.dayOfWeek`）
     - `Monthly` （需要您指定 `spec.minute, spec.hour 、 spec.dayOfMonth`）
     - `Custom`
   - **spec.dayOfMonth**：（選用）排程應執行的月份日期（1 - 31）。如果精細度設定為 `Monthly`，則此欄位為必填。此值必須以字串形式提供。
   - **spec.dayOfWeek**：（選用）排程應執行的星期幾（0 - 7）。值為 0 或 7 表示星期日。如果精細度設為 `Weekly`，則此欄位為必填項。該值必須以字串形式提供。
   - **spec.hour**：（選用）排程應執行的一天中的小時（0 - 23）。如果粒度設定為 `Daily、 Weekly`或 `Monthly`，則此欄位為必填項。值必須以字串形式提供。
   - **spec.minute**：（選用）排程應執行的小時分鐘數（0 - 59）。如果粒度設定為 `Hourly、 Daily、 Weekly`或 `Monthly`，則此欄位為必填項。該值必須以字串形式提供。

備份和快照排程的範例 YAML：

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

僅快照排程的範例 YAML ：

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

3. 在 `trident-protect-schedule-cr.yaml` 檔案中填入正確的值後，套用 CR：

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

使用 **CLI** 建立排程

步驟

1. 建立保護排程，並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`來查看此命令的詳細說明資訊。

```
tridentctl-protect create schedule <my_schedule_name> \
  --appvault <my_appvault_name> \
  --app <name_of_app_to_snapshot> \
  --backup-retention <how_many_backups_to_retain> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_to_run_schedule> \
  --granularity <frequency_to_run> \
  --hour <hour_of_day_to_run> \
  --minute <minute_of_hour_to_run> \
  --recurrence-rule <recurrence> \
  --snapshot-retention <how_many_snapshots_to_retain> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

以下旗標可讓您對排程進行更多控制：

- 完整備份排程：使用 `--full-backup-rule` 標誌來排程非增量完整備份。此標誌僅適用於 `--granularity Daily`。可能的值：
  - `Always`：每天都要建立完整備份。
  - 特定工作日：指定一個或多個以逗號分隔的日期（例如， `"Monday,Thursday"`）。有效值：Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。

    `--full-backup-rule` 標誌不適用於每小時、每週或每月粒度。
- 僅快照排程：設定 `--backup-retention 0` 並為 `--snapshot-retention` 指定大於零的值。

支援的排程註釋

下表描述了建立排程 CR 時可以使用的註釋：

註解	類型	說明	預設值
protect.trident.netapp.io/full-backup-rule	字串	指定完整備份的調度規則。您可以將其設定為 `Always`持續完整備份，或根據您的需求進行自訂。例如，如果您選擇每日粒度，則可以指定執行完整備份的星期幾（例如，`"Monday,Thursday"）。有效的星期幾值為：星期一、星期二、星期三、星期四、星期五、星期六、星期日。請注意，此註解只能與已將 `granularity`設定為 `Daily`的計劃一起使用。	未設定 （所有備份均為增量備份）
protect.trident.netapp.io/snapshot-completion-timeout	字串	允許完成整個快照作業的最長時間。	"60 分鐘"
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	字串	磁碟區快照達到可用狀態所允許的最長時間。	"30 分鐘"
protect.trident.netapp.io/volume-snapshots-created-timeout	字串	建立磁碟區快照所允許的最長時間。	"5 分鐘"
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待所有新建立的 PersistentVolumeClaims （PVC）達到 `Bound`階段的最長時間（以秒為單位），超過此時間操作將會失敗。	"1200"（20 分鐘）

刪除快照

刪除不再需要的已排程或隨需快照。

步驟

1. 刪除與快照相關聯的快照 CR：

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

刪除備份

刪除不再需要的排程備份或隨需備份。

請確保將回收策略設定為 `Delete`，以從物件儲存中刪除所有備份資料。此策略的預設設定為 `Retain`，以避免意外資料遺失。如果未將策略變更為 `Delete`，則備份資料將保留在物件儲存中，需要手動刪除。

步驟

1. 刪除與備份相關聯的備份 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狀態。

步驟

1. 使用以下命令檢索備份作業的狀態，將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-files**（**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 磁碟區，並且不會佔用容量池中的額外空間。

開始之前

請確保以下事項：

- 您已安裝 Trident Protect 。
- 您已在 Trident Protect 中定義了一個應用程式。在您完成此程序之前，該應用程式的保護功能將受到限制。
- 您已選擇 `azure-netapp-files`作為儲存後端的預設儲存類別。

展開以檢視設定步驟

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 且與應用程式相關聯的 PV 啟用 Snapshot 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 Snapshot 目錄：

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

   回應：

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未啟用快照目錄，Trident Protect 將選擇常規備份功能，該功能會在備份過程中暫時佔用容量池中的空間。在這種情況下，請確保容量池中有足夠的空間來建立一個與待備份磁碟區大小相同的臨時磁碟區。

結果

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

還原應用程式

使用 **Trident Protect** 恢復應用程式

您可以使用 Trident Protect 從快照或備份還原應用程式。如果要將應用程式還原到同一個叢集，從現有快照還原速度會更快。

- 還原應用程式時，所有為該應用程式配置的執行鉤子都會隨應用程式一起還原。如果存在還原後執行鉤子，它會在還原操作過程中自動執行。
- qtree 磁碟區支援從備份還原到不同的命名空間或原始命名空間。但是，qtree 磁碟區不支援從快照還原到不同的命名空間或原始命名空間。
- 您可以使用進階設定來自訂還原操作。若要深入瞭解、請參閱 "使用進階 Trident Protect 還原設定"。

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式，並為還原後的應用程式建立一個應用程式 CR。若要保護還原後的應用程式，您可以建立按需備份或快照，或設定保護計畫。

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

開始之前

請確保 AWS 工作階段權杖的有效期限足以應付任何長時間執行的 s3 還原作業。如果權杖在還原作業期間過期、作業可能會失敗。

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

當您使用 Kopia 作為資料移動工具還原備份時，您可以選擇在 CR 中或使用 CLI 指定註釋，以控制 Kopia 使用的暫存的行為。有關可配置選項的更多資訊，請參閱 "Kopia 說明文件"。使用 `tridentctl-protect create --help`命令以取得有關使用 Trident Protect CLI 指定註釋的更多資訊。

使用 **CR**

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-backup-restore-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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來尋找此路徑：

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

   - **spec.appVaultRef**：（必填）儲存備份內容的 AppVault 名稱。
   - **spec.namespaceMapping**：復原作業的來源命名空間到目標命名空間的對應。請將 `my-source-namespace`和 `my-destination-namespace`替換為您環境中的資訊。

     ```
     ---
     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還原，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，如果您選擇了持久卷聲明資源，並且它關聯了一個 pod，Trident Protect 也會恢復該關聯 pod。

   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `Include`或 `Exclude`來包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位中的名稱。
- **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

  例如：

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

4. 在 `trident-protect-backup-restore-cr.yaml` 檔案中填入正確的值後，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，並將方括號中的值替換為您環境中的資訊。 `namespace-mapping`引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 <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

從備份還原到原始命名空間

您可以隨時將備份還原至原始命名空間。

開始之前

請確保 AWS 工作階段權杖的有效期限足以應付任何長時間執行的 s3 還原作業。如果權杖在還原作業期間過期、作業可能會失敗。

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

> 當您使用 Kopia 作為資料移動工具還原備份時，您可以選擇在 CR 中或使用 CLI 指定註釋，以控制 Kopia 使用的暫存的行為。有關可配置選項的更多資訊，請參閱 "Kopia 說明文件"。使用 `tridentctl-protect create --help`命令以取得有關使用 Trident Protect CLI 指定註釋的更多資訊。

使用 **CR**

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-backup-ipr-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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來尋找此路徑：

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

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

     例如：

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

3. （可選）如果您只需要選擇應用程式中的某些資源進行還原，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，如果您選擇了持久卷聲明資源，並且它關聯了一個 pod，Trident Protect 也會恢復該關聯 pod。

   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `Include`或 `Exclude`來包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位

中的名稱。

- **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

  例如：

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

4. 在 `trident-protect-backup-ipr-cr.yaml` 檔案中填入正確的值後，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，並將方括號中的值替換為您環境中的資訊。 `backup` 參數使用命名空間和備份名稱，格式為 `<namespace>/<name>`。例如：

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

從備份還原到不同的叢集

如果原始叢集出現問題、您可以將備份還原至不同的叢集。

- 當您使用 Kopia 作為資料移動工具還原備份時，您可以選擇在 CR 中或使用 CLI 指定註釋，以控制 Kopia 使用的暫存的行為。有關可配置選項的更多資訊，請參閱 "Kopia 說明文件"。使用 `tridentctl-protect create --help`命令以取得有關使用 Trident Protect CLI 指定註釋的更多資訊。
- 使用 CR 還原到新命名空間時，必須先手動建立目標命名空間，然後再套用 CR。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

開始之前

請確保符合下列先決條件：

- 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 儲存桶路徑，備份檔案就儲存在該路徑中。
- 執行 `tridentctl-protect get appvaultcontent`命令時，請確保本機環境可以連接到 AppVault CR 中定義的物件儲存桶。如果網路限制導致無法存取，請改為在目標叢集的 Pod 內執行 Trident Protect CLI。
- 請確保 AWS 工作階段權杖的有效期限足以應付任何長時間執行的還原作業。如果權杖在還原作業期間過期、作業可能會失敗。
  - 有關檢查當前會話令牌過期時間的更多資訊，請參閱 "AWS API 文件"。
  - 如需 AWS 資源憑證的詳細資訊，請參閱 "AWS 文件"。

步驟

1. 使用 Trident Protect CLI 外掛程式驗證 AppVault CR 是否存在於目標叢集上：

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

如果目標叢集上不存在 AppVault CR，請按照 "使用 Trident Protect AppVault 物件來管理儲存桶" 中的步驟建立它。

2. 查看目標叢集上可用 AppVault 的備份內容，並記下 `appArchivePath`要還原的備份：

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

執行此命令將顯示 AppVault 中的可用備份，包括其來源叢集、相應的應用程式名稱、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 時，請確保用於應用程式還原的命名空間存在於目的地叢集上。

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. 在您建立的檔案中、設定以下屬性：
   - **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**：（必填）儲存備份內容的 AppVault 名稱。
   - **spec.appArchivePath**：（必填）AppVault 內儲存備份內容的路徑。使用步驟 2 中的指令查看備份內容，並找到 `appArchivePath`要還原的備份。
   - **spec.namespaceMapping**：復原作業的來源命名空間到目標命名空間的對應。請將 `my-source-namespace`和 `my-destination-namespace`替換為您環境中的資訊。

     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. 在 `trident-protect-backup-restore-cr.yaml` 檔案中填入正確的值後，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式，並將方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）檔案從快照還原資料，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式，並為還原的應用程式建立一個應用程式 CR。若要保護還原的應用程式，您可以建立隨需備份或快照，或建立保護排程。

- SnapshotRestore 支援 `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支援。如果嘗試還原到使用不同儲存後端的 `StorageClass`，則還原操作將會失敗。
- 使用 CR 還原到新命名空間時，必須先手動建立目標命名空間，然後再套用 CR。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

開始之前

請確保 AWS 工作階段權杖的有效期限足以應付任何長時間執行的 s3 還原作業。如果權杖在還原作業期間過期、作業可能會失敗。

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

使用 **CR**

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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**：（必填）儲存快照內容的 AppVault 名稱。
   - **spec.appArchivePath**：AppVault 內儲存快照內容的路徑。您可以使用以下命令來尋找此路徑：

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

   - **spec.namespaceMapping**：復原作業的來源命名空間到目標命名空間的對應。請將 `my-source-namespace`和 `my-destination-namespace`替換為您環境中的資訊。

     ```
     ---
     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還原，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，如果您選擇了持久卷聲明資源，並且它關聯了一個 pod，Trident Protect 也會恢復該關聯 pod。

   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `Include`或 `Exclude`來包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位中的名稱。
- **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。
- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

  例如：

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

4. 在 `trident-protect-snapshot-restore-cr.yaml` 檔案中填入正確的值後，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。

   - `snapshot`引數使用命名空間和快照名稱，格式為 `<namespace>/<name>`。
   - `namespace-mapping` 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 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至原始命名空間。

如果您的應用程式使用多個命名空間，且這些命名空間中存在同名的 PVC，則快照還原作業（無論是就地還原或還原到新命名空間）都將無法正常運作。所有還原的磁碟區將包含相同的資料，而不是每個命名空間對應的正確資料。請使用備份還原代替快照還原，或升級到 26.02 或更高版本以修復此問題。

開始之前

請確保 AWS 工作階段權杖的有效期限足以應付任何長時間執行的 s3 還原作業。如果權杖在還原作業期間過期、作業可能會失敗。

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

使用 **CR**

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-snapshot-ipr-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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**：（必填）儲存快照內容的 AppVault 名稱。
   - **spec.appArchivePath**：AppVault 內儲存快照內容的路徑。您可以使用以下命令來尋找此路徑：

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

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

3. （可選）如果您只需要選擇應用程式中的某些資源進行還原，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，如果您選擇了持久卷聲明資源，並且它關聯了一個 pod，Trident Protect 也會恢復該關聯 pod。

   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `Include`或 `Exclude`來包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位中的名稱。
       - **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。

- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

  例如：

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

4. 在 `trident-protect-snapshot-ipr-cr.yaml` 檔案中填入正確的值後，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，並將方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 <namespace/snapshot_to_restore> \
-n <application_namespace>
```

檢查還原作業的狀態

您可以使用命令列來檢查正在進行、已完成或已失敗的還原作業狀態。

步驟

1. 使用以下命令檢索還原作業的狀態，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 還原設定

您可以利用註釋、命名空間設定和儲存選項等進階設定來自訂還原作業，以滿足您的特定需求。

還原和容錯移轉作業期間的命名空間註釋和標籤

在還原和容錯移轉作業期間，目的地命名空間中的標籤和註釋會與來源命名空間中的標籤和註釋相符。來源命名空間中存在但目的地命名空間中不存在的標籤或註釋會被新增，而任何已存在的標籤或註釋則會被覆寫以符合來源命名空間中的值。僅存在於目的地命名空間中的標籤或註釋則保持不變。

如果您使用 Red Hat OpenShift，請務必注意命名空間註解在 OpenShift 環境中的關鍵作用。命名空間註解可確保復原的 Pod 遵循 OpenShift 安全上下文約束（SCC）定義的相應權限和安全性配置，並能無權限問題地存取磁碟區。如需更多資訊，請參閱"OpenShift 安全上下文約束文檔"。

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

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

執行還原或容錯移轉作業時， `restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 中指定的任何命名空間註釋和標籤都會從還原或容錯移轉作業中排除。請確保在初始 Helm 安裝期間配置這些設定。若要深入瞭解、請參閱 "設定其他 Trident Protect Helm Chart 設定"。

如果您使用 Helm 並帶有 `--create-namespace` 標誌安裝了來源應用程式，則會對 `name` 標籤鍵進行特殊處理。在還原或容錯移轉過程中，Trident Protect 會將此標籤複製到目的地命名空間，但如果來源的值與來源命名空間相符，則會將值更新為目的地命名空間值。如果此值與來源命名空間不相符，則會將其複製到目的地命名空間而不做任何變更。

範例

以下範例展示了來源命名空間和目標命名空間，它們各自具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，以及註解和標籤在目標命名空間中是如何組合或覆蓋的。

在還原或容錯移轉作業之前

下表說明了復原或容錯移轉作業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」	• environment=production • 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

還原作業後

下表展示了復原或故障轉移作業後範例目標命名空間的狀態。一些鍵已被添加，一些鍵已被覆蓋，並且 `name` 標籤已更新以匹配目標命名空間：

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 • environment=production • 角色=資料庫

支援的欄位

本節說明可用於還原作業的其他欄位。

儲存類別對應

此 `spec.storageClassMapping`屬性定義了從來源應用程式中的儲存類別到目標叢集上新儲存類別的對應。在將應用程式遷移到具有不同儲存類別的叢集之間，或變更 BackupRestore 操作的儲存後端時，可以使用此屬性。

- 範例： *

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

支援的註釋

本節列出系統中用於配置各種行為的支援註解。如果使用者未明確設定註解，系統將使用預設值。

註解	類型	說明	預設值
protect.trident.netapp.io/data-mover-timeout-sec	字串	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	字串	Kopia 內容快取的大小上限（以 MB 為單位）。	"1000"
protect.trident.netapp.io/pvc-bind-timeout-sec	字串	等待所有新建立的 PersistentVolumeClaims（PVC）達到 `Bound`階段的最長時間（以秒為單位），超過此時間操作將會失敗。適用於所有還原 CR 類型（BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore）。如果您的儲存後端或叢集通常需要更多時間，請使用更高的值。	"1200"（20 分鐘）

使用 NetApp SnapMirror 和 Trident Protect 複製應用程式

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

還原和容錯移轉作業期間的命名空間註釋和標籤

在還原和容錯移轉作業期間，目的地命名空間中的標籤和註釋會與來源命名空間中的標籤和註釋相符。來源命名空間中存在但目的地命名空間中不存在的標籤或註釋會被新增，而任何已存在的標籤或註釋則會被覆寫以符合來源命名空間中的值。僅存在於目的地命名空間中的標籤或註釋則保持不變。

如果您使用 Red Hat OpenShift，請務必注意命名空間註解在 OpenShift 環境中的關鍵作用。命名空間註解可確保復原的 Pod 遵循 OpenShift 安全上下文約束（SCC）定義的相應權限和安全性配置，並能無權限問題地存取磁碟區。如需更多資訊，請參閱"OpenShift 安全上下文約束文檔"。

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

```
helm upgrade trident-protect -n trident-protect netapp-trident-protect/trident-protect \
  --set-string restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_key_to_skip_2>}" \
  --reuse-values
```

執行還原或容錯移轉作業時， `restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 中指定的任何命名空間註釋和標籤都會從還原或容錯移轉作業中排除。請確保在初始 Helm 安裝期間配置這些設定。若要深入瞭解、請參閱 "設定其他 Trident Protect Helm Chart 設定"。

如果您使用 Helm 並帶有 `--create-namespace` 標誌安裝了來源應用程式，則會對 `name` 標籤鍵進行特殊處理。在還原或容錯移轉過程中，Trident Protect 會將此標籤複製到目的地命名空間，但如果來源的值與來源命名空間相符，則會將值更新為目的地命名空間值。如果此值與來源命名空間不相符，則會將其複製到目的地命名空間而不做任何變更。

範例

以下範例展示了來源命名空間和目標命名空間，它們各自具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，以及註解和標籤在目標命名空間中是如何組合或覆蓋的。

在還原或容錯移轉作業之前

下表說明了復原或容錯移轉作業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」	• environment=production • 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

還原作業後

下表展示了復原或故障轉移作業後範例目標命名空間的狀態。一些鍵已被添加，一些鍵已被覆蓋，並且 `name` 標籤已更新以匹配目標命名空間：

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 • environment=production • 角色=資料庫

您可以設定 Trident Protect 在資料保護作業期間凍結及解除凍結檔案系統。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。

故障轉移和反向操作期間的執行掛鉤

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。故障轉移後，執行鉤子將存在於應用程式中，並在執行任何相關操作時運行。
- 在反向同步或反向重同步過程中，應用程式上所有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，這些執行鉤子將失效並被刪除以防止其執行。

若要深入瞭解執行掛鉤，請參閱 "管理 Trident Protect 執行掛鉤"。

建立複寫關係

建立複寫關係涉及下列項目：

- 選擇 Trident Protect 拍攝應用程式快照的頻率（包括應用程式的 Kubernetes 資源以及應用程式每個磁碟區的磁碟區快照）
- 選擇複寫排程（包括 Kubernetes 資源以及持續性磁碟區資料）
- 設定拍攝 Snapshot 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 "AppVault 自訂資源" 中的範例以適應您的環境：

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. 設定下列屬性：

- **metadata.name**：（必填）AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 CR 檔案會引用此值。
- **spec.providerConfig**：（必要）儲存使用指定提供者存取 AppVault 所需的組態。選擇 bucketName 以及提供者所需的任何其他詳細資料。請記下您選擇的值，因為複寫關係所需的其他 CR 檔案會參照這些值。如需其他提供者的 AppVault CR 範例，請參閱 "AppVault 自訂資源"。
- **spec.providerCredentials**：（*Required*）儲存使用指定提供者存取 AppVault 所需的任何憑證參考。
  - **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 secret 名稱。必須位於同一個 namespace 中。
  - **spec.providerCredentials.secretAccessKey**：（必要）用於存取提供者的存取金鑰。**name** 應與 **spec.providerCredentials.valueFromSecret.name** 相符。
- **spec.providerType**：（必填）確定備份服務商；例如，NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可選值：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. 在 `trident-protect-appvault-primary-source.yaml` 檔案中填入正確的值後，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. 設定下列屬性：

- **metadata.name**：（必填）應用程式自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 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，並可選擇性地使用標籤來縮小命名空間的範圍，以指定此處列出的命名空間中存在的資源。應用程式命名空間必須包含在此陣列中。

  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. 在 `trident-protect-app-source.yaml` 檔案中填入正確的值後，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。若要建立按需快照，請參閱 "建立隨需快照"。

4. 在來源叢集上、建立複寫排程 CR：

除了下方提供的排程之外，建議建立一個單獨的每日快照排程，並將保留期設定為 7 天，以便在對等 ONTAP 叢集之間維護一個通用的快照。這樣可以確保快照最多可用 7 天，但保留期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進行最多 7 天的逆向操作。這種方法可以加快逆向過程並提高效率，因為只會傳輸自上次快照以來所做的變更，而不是所有資料。

如果應用程式的現有排程已符合所需的保留要求，則不需要其他排程。

使用 **CR** 建立複寫排程

a. 為來源應用程式建立複寫排程：

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

   ii. 設定下列屬性：

   - **metadata.name**：（必填）排程自訂資源的名稱。
   - **spec.appVaultRef**：（必填）此值必須與來源應用程式的 AppVault metadata.name 欄位相符。
   - **spec.applicationRef**：（必填）此值必須與來源應用程式 CR 的 metadata.name 欄位相符。
   - **spec.backupRetention**：（必填）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. 在 `trident-protect-schedule.yaml` 檔案中填入正確的值後，套用 CR：

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

使用 **CLI** 建立複寫排程

a. 建立複寫排程，並將括號中的值替換為您環境中的資訊：

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

- 範例： *

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

5. 在目標叢集上，建立一個與在來源叢集上應用的 AppVault CR 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. 套用 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 "AppVault 自訂資源" 中的範例以適應您的環境：
   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. 設定下列屬性：
      - **metadata.name**：（必填）AppVault 自訂資源的名稱。請記下您選擇的名稱，因為複寫關係所需的其他 CR 檔案會引用此值。
      - **spec.providerConfig**：（必要）儲存使用指定提供者存取 AppVault 所需的組態。選擇一個 `bucketName`以及提供者所需的任何其他詳細資訊。請記下您選擇的值，因為複寫關係所需的其他 CR 檔案會參照這些值。請參閱"AppVault 自訂資源"以取得其他提供者的 AppVault CR 範例。
      - **spec.providerCredentials**：（*Required*）儲存使用指定提供者存取 AppVault 所需的任何憑證參考。
        - **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 secret 名稱。必須位於同一個 namespace 中。
        - **spec.providerCredentials.secretAccessKey**：（必要）用於存取提供者的存取金鑰。**name** 應與 **spec.providerCredentials.valueFromSecret.name** 相符。

- **spec.providerType**：（必填）確定備份服務商；例如，NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可選值：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. 在 `trident-protect-appvault-secondary-destination.yaml` 檔案中填入正確的值後，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 時，請在套用 CR 之前手動建立目的地命名空間。Trident Protect 僅在使用 CLI 時才會自動建立命名空間。

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

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

b. 設定下列屬性：

- **metadata.name:** （必填）AppMirrorRelationship 自訂資源的名稱。
- **spec.destinationAppVaultRef**：（必填）此值必須與目標叢集上目標應用程式的 AppVault 名稱相符。
- **spec.namespaceMapping**：（必需）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**：（*Required*）此值必須與來源應用程式的 AppVault 名稱相符。
- **spec.sourceApplicationName**：（必需）此值必須與您在來源應用程式 CR 中定義的來源應用程式名稱相符。
- **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**：（選用）選擇叢集上有效儲存類別的名稱。儲存類別必須連結至與來源環境對等的 ONTAP 儲存 VM。如果未提供儲存類別，則預設會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. 在 `trident-protect-relationship.yaml` 檔案中填入正確的值後，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 物件，將括號中的值替換為您環境中的資訊：

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

- 範例： *

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

9. （可選）在目標叢集上、檢查複寫關係的狀態和狀態：

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

故障轉移至目的地叢集

使用 Trident Protect，您可以將複製的應用程式容錯移轉至目的地叢集。此程序會停止複寫關係，並使應用程式在目的地叢集上線上。如果來源叢集上的應用程式正在運作，Trident Protect 不會停止該應用程式。

步驟

1. 在目標叢集上，編輯 AppMirrorRelationship CR 檔案（例如， `trident-protect-relationship.yaml` ），並將 **spec.desiredState** 的值變更為 `Promoted`。
2. 儲存 CR 檔案。

3. 套用 CR：

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

4. （可選）在故障轉移應用程式上建立所需的任何保護排程。
5. （可選）檢查複寫關係的狀態和狀態：

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

重新同步故障轉移的複寫關係

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

該過程會在重新建立複寫之前停止目的地叢集上的應用程式。

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. 選用：在來源叢集上，建立來源應用程式的快照。這可確保擷取來源叢集的最新變更。
2. 在目標叢集上，編輯 AppMirrorRelationship CR 檔案（例如， `trident-protect-relationship.yaml` ），並將 spec.desiredState 的值變更為 `Established`。
3. 儲存 CR 檔案。
4. 套用 CR：

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

5. 如果您在目的地叢集上建立了任何保護排程來保護故障轉移的應用程式，請將其移除。任何保留的排程都會導致磁碟區快照失敗。

反向重新同步已容錯移轉的複寫關係

當您對故障轉移的複製關係進行反向同步時，目標應用程式將變為來源應用程式，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。這將使目標叢集變為來源叢集。如果新目標叢集上仍有任何保護排程，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，以建立複寫關係。
3. 確保新目標（原始來源叢集）配置了兩個 AppVault CR。
4. 在相反的叢集上建立複寫關係，並設定反向的值。

反向應用程式複寫方向

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

在這種情況下，您正在交換來源和目的地。

步驟

1. 在來源叢集上，建立關機快照：

使用 **CR** 建立關機快照

a. 停用來源應用程式的保護原則排程。

b. 建立 ShutdownSnapshot CR 檔案：

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

ii. 設定下列屬性：

- **metadata.name**：（必填）自訂資源的名稱。
- **spec.AppVaultRef**：（*Required*）此值必須與來源應用程式的 AppVault metadata.name 欄位相符。
- **spec.ApplicationRef**：（必填）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. 在 `trident-protect-shutdownsnapshot.yaml` 檔案中填入正確的值後，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，並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 Snapshot 完成後，取得關機 Snapshot 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；這是最後一個斜槓之後的所有內容）：

```
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容錯移轉，並進行以下變更：

> 在故障轉移過程的步驟 2 中，將 `spec.promotedSnapshot` 欄位包含在 AppMirrorRelationship CR 檔案中，並將其值設定為您在上面的步驟 3 中記錄的基本名稱。

5. 在 [反向重新同步已容錯移轉的複寫關係] 中執行反向重新同步步驟。
6. 在新來源叢集上啟用保護排程。

結果

由於反向複寫，會發生以下動作：

- 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 Volume 進行快照並進行複寫。
- SnapMirror 關係已中斷，使目的地磁碟區準備好進行讀取 / 寫入。
- 該應用程式的 Kubernetes 資源是從關閉前的快照中還原，使用的是在原始來源應用程式關閉後複寫的磁碟區資料。
- 複寫會以相反的方向重新建立。

將應用程式容錯回復至原始來源叢集

使用 Trident Protect，您可以透過以下步驟在故障轉移作業後實現「故障復原」。在此工作流程中，為了恢復原始複寫方向，Trident Protect 會在反轉複寫方向之前，將所有應用程式變更複寫（重新同步）回原始來源應用程式。

此程序從已完成容錯移轉至目的地的關係開始，並涉及下列步驟：

- 從故障轉移狀態開始。
- 反向重新同步複寫關係。

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

- 反轉複寫方向。

步驟

1. 執行[反向重新同步已容錯移轉的複寫關係]步驟。
2. 執行[反向應用程式複寫方向]步驟。

刪除複寫關係

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

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，在叢集之間遷移應用程式或將應用程式遷移到不同的儲存類別。

遷移應用程式時，所有為該應用程式配置的執行鉤子都會隨應用程式一起遷移。如果存在復原後執行鉤子，它會在復原操作過程中自動執行。

備份與還原作業

若要針對下列案例執行備份與還原作業，您可以自動執行特定的備份與還原工作。

複製到同一叢集

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

步驟

1. 請執行下列其中一項操作：
   a. "建立快照".
   b. "建立備份".
2. 在同一叢集上，根據您建立的是快照還是備份，執行下列其中一項：
   a. "從快照還原資料".
   b. "從備份中恢復資料".

複製到不同的叢集

若要將應用程式複製到另一個叢集（執行跨叢集複製），請在來源叢集上建立備份，然後將備份還原到另一個叢集。確保目標叢集上已安裝 Trident Protect。

您可以使用 "SnapMirror 複製" 在不同的叢集之間複製應用程式。

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. 在目標叢集上，"從備份中恢復資料"。

將應用程式從一個儲存類別遷移到另一個儲存類別

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

例如（不包括還原 CR 中的機密）：

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

使用 **CR** 還原快照

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您建立的檔案中、設定以下屬性：
   - **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來尋找此路徑：

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

   - **spec.appVaultRef**：（必填）儲存快照內容的 AppVault 名稱。
   - **spec.namespaceMapping**：復原作業的來源命名空間到目標命名空間的對應。請將 `my-source-namespace`和 `my-destination-namespace`替換為您環境中的資訊。

     ```
     ---
     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. （選用）如果您需要僅選擇要還原的應用程式的某些資源，請新增篩選條件，以包含或排除標記有特定標籤的資源：
   - **resourceFilter.resourceSelectionCriteria**：（篩選必需）使用 `include or exclude`包含或排除在 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 物件的陣列。如果在此陣列中定義多個元素，則它們之間按 OR 運算匹配，每個元素內的欄位（group、kind、version）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 metadata.name 欄位中的名稱。
       - **resourceMatchers[].namespaces**：（可選）要篩選的資源的 Kubernetes metadata.name 欄位中的命名空間。

- **resourceMatchers[].labelSelectors**：（可選）資源在 Kubernetes metadata.name 欄位中定義的標籤選擇器字串 "Kubernetes 說明文件"。例如：`"trident.netapp.io/os=linux"`。

  例如：

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

4. 在 `trident-protect-snapshot-restore-cr.yaml` 檔案中填入正確的值後，套用 CR：

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

使用 **CLI** 還原快照

步驟

1. 將快照還原到不同的命名空間，並將括號中的值替換為您環境中的資訊。

   - `snapshot`引數使用命名空間和快照名稱，格式為 `<namespace>/<name>`。
   - `namespace-mapping` 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 snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

管理 Trident Protect 執行掛鉤

執行鉤子是一種自訂操作，您可以將其配置為與託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式，則可以使用執行鉤子在建立快照之前暫停所有資料庫事務，並在快照完成後恢復事務。這可以確保應用程式快照的一致性。

執行掛鉤的類型

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

- 快照前
- 快照後
- 備份前
- 備份後
- 還原後
- 故障轉移後

執行順序

執行資料保護作業時，執行掛鉤事件會依照以下順序發生：

1. 任何適用的自訂預操作執行鉤子都會在相應的容器上運行。您可以建立並運行任意數量的自訂預操作鉤子，但這些鉤子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，檔案系統會凍結。"深入瞭解如何使用 Trident Protect 設定檔案系統凍結"。
3. 執行資料保護作業。
4. 如果適用，則解凍已凍結的檔案系統。
5. 任何適用的自訂後操作執行鉤子都會在相應的容器上運行。您可以建立並運行任意數量的自訂後操作鉤子，但這些鉤子在操作後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鉤子（例如，快照前鉤子），則這些鉤子的執行順序無法保證。但是，不同類型鉤子的執行順序是有保證的。例如，以下是包含所有不同類型鉤子的組態的執行順序：

1. 已執行快照前鉤子
2. 執行快照後掛鉤
3. 執行備份前掛鉤
4. 執行備份後掛鉤

前述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鉤子腳本之前，請務必先對其進行測試。您可以使用 'kubectl exec' 指令方便地測試這些腳本。在生產環境中啟用執行鉤子後，請測試產生的快照和備份，以確保它們的一致性。您可以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。

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還原作業中。

關於自訂執行掛鉤的重要說明

在規劃應用程式的執行鉤子時,請考慮以下幾點。

- 執行掛鉤必須使用指令碼來執行動作。許多執行掛鉤可以參照同一個指令碼。
- Trident Protect 要求執行鉤子使用的腳本以可執行 shell 腳本的格式編寫。
- 指令碼大小限制為 96KB。
- Trident Protect 使用執行掛鉤設定和任何相符條件來判斷哪些掛鉤適用於快照、備份或還原作業。

由於執行鉤子通常會降低甚至完全停用其所針對應用程式的功能，因此您應該始終盡量縮短自訂執行鉤子的執行時間。如果您啟動了一個帶有關聯執行鉤子的備份或快照操作，但隨後取消了該操作，只要備份或快照操作已經開始，這些鉤子仍然可以執行。這意味著備份後執行鉤子中使用的邏輯不能假定備份已經完成。

執行掛鉤篩選器

當您新增或編輯應用程式的執行鉤子時，您可以為該執行鉤子添加過濾器，以管理鉤子將匹配哪些容器。過濾器適用於所有容器都使用相同容器映像，但每個映像可能用於不同用途的應用程式（例如 Elasticsearch）。過濾器可讓您建立執行鉤子在部分（但不一定全部）相同容器上運行的場景。如果為單一執行鉤子建立多個過濾器，它們將透過邏輯 AND 運算子進行組合。每個執行鉤子最多可以有 10 個活動過濾器。

新增到執行鉤子的每個過濾器都使用正規表示式來匹配叢集中的容器。當鉤子匹配到容器時，它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 支援的執行鉤子過濾器正規表示式語法的更多資訊，請參閱 "Regular Expression 2 (RE2) 語法支援"。

如果在還原或複製作業之後執行的執行鉤子中新增命名空間篩選器，且還原或複製來源和目的地位於不同的命名空間中，則命名空間篩選器僅套用於目的地命名空間。

執行掛鉤範例

造訪 "NetApp Verda GitHub 專案"下載適用於 Apache Cassandra 和 Elasticsearch 等熱門應用程式的真實執行鉤子。您也可以查看範例，並從中取得建置自訂執行鉤子的想法。

建立執行掛鉤

您可以使用 Trident Protect 為應用程式建立自訂執行鉤子。您需要擁有 Owner、Admin 或 Member 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 環境和叢集組態：
   - **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**：（必填）要執行執行鉤子的應用程式的 Kubernetes 名稱。
   - **spec.stage**：（必需）一個字串，指示執行鉤子應在操作的哪個階段運行。可能的值：
     - 預先
     - 發佈
   - **spec.action**：（必要）一個字串，指示執行鉤子將執行的操作，假設符合任何指定的執行鉤子篩選器。可能的值：
     - 快照
     - 備份
     - 還原
     - 容錯移轉
   - **spec.enabled**：（可選）指示此執行鉤子是否啟用或停用。如果未指定，則預設值為 true。
   - **spec.hookSource**：（必填）包含 base64 編碼的 hook 腳本的字串。
   - **spec.timeout**：（可選）一個數字，定義執行鉤子允許運行的最長時間（以分鐘為單位）。最小值為 1 分鐘，如果未指定，則預設值為 25 分鐘。
   - **spec.arguments**：（可選）您可以為執行掛鉤指定的 YAML 引數清單。
   - **spec.matchingCriteria**：（可選）條件鍵值對的可選清單，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新增 10 個過濾器。
   - **spec.matchingCriteria.type**：（可選）用於識別執行鉤子過濾器類型的字串。可能的值：
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value**：（可選）識別執行鉤子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，並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>
```

手動執行執行掛鉤

您可以手動執行 execution hook 以進行測試，或者在失敗後需要手動重新執行 hook。您需要擁有 Owner 、Admin 或 Member 權限才能手動執行 execution hook。

手動執行執行掛鉤包含兩個基本步驟：

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. 針對備份執行執行掛鉤

步驟 **1**：建立資源備份

使用 **CR**

步驟

1. 建立自訂資源（CR）檔案並將其命名為 `trident-protect-resource-backup.yaml`。
2. 設定以下屬性以符合您的 Trident Protect 環境和叢集組態：
   - **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**：（必填）要為其建立資源備份的應用程式的 Kubernetes 名稱。
   - **spec.appVaultRef**：（必填）儲存備份內容的 AppVault 名稱。
   - **spec.appArchivePath**：AppVault 內儲存備份內容的路徑。您可以使用以下命令來尋找此路徑：

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

     YAML 範例：

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

3. 在用正確的值填入 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，並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. 檢視備份狀態。您可以重複使用此範例命令，直到作業完成：

```
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**：執行 **execution hook**

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 環境和叢集組態：
   - **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**：（必填）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**：（必填）請確保此值與您在步驟 1 中建立的 ResourceBackup CR 的 appVaultRef 相符。
   - **spec.appArchivePath**：請確保此值與您在步驟 1 中建立的 ResourceBackup CR 內的 appArchivePath 相符。

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

   - **spec.action**：（必要）一個字串，指示執行鉤子將執行的操作，假設符合任何指定的執行鉤子篩選器。可能的值：
     - 快照
     - 備份
     - 還原
     - 容錯移轉
   - **spec.stage**：（必需）一個字串，指示執行鉤子應在操作的哪個階段運行。此鉤子運行不會在任何其他階段運行鉤子。可能的值：
     - 預先
     - 發佈

     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. 建立手動執行掛鉤執行請求：

```
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. 檢查執行掛鉤執行的狀態。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>
```