



管理和保护应用程序 Trident

NetApp
October 29, 2026

目录

管理和保护应用程序	1
使用 Trident Protect AppVault 对象管理存储桶	1
配置 AppVault 身份验证和密码	1
AppVault 创建示例	4
查看 AppVault 信息	10
移除 AppVault	11
使用 Trident Protect 定义管理应用程序	12
创建 AppVault CR	12
定义应用程序	12
使用 Trident Protect 保护应用程序	16
创建按需快照	16
创建按需备份	17
创建数据保护计划	19
删除快照	23
删除备份	24
检查备份操作的状态	24
为 azure-netapp-files (ANF) 操作启用备份和还原	24
还原应用程序	25
使用 Trident Protect 还原应用程序	25
使用高级 Trident Protect 还原设置	38
使用自定义资源还原资源时修改资源	40
使用 NetApp SnapMirror 和 Trident Protect 复制应用程序	45
还原和故障转移操作期间的命名空间注释和标签	45
在故障转移和反向操作期间执行挂钩	46
设置复制关系	47
反向应用程序复制方向	55
使用 Trident Protect 迁移应用程序	57
备份和还原操作	57
将应用程序从一个存储类迁移到另一个存储类	58
使用 Trident Protect 保护 KubeVirt 虚拟机	61
KubeVirt 保护模式	61
备份和快照期间收集的资源	61
使用基于命名空间的应用程序定义保护虚拟机	63
使用标签选择器实现动态虚拟机保护	64
使用 includedVirtualMachines 保护特定 VM	65
还原特定 KubeVirt VM	66
使用 CLI 创建 VM 范围的应用程序	71
相关信息	71
管理 Trident Protect 执行挂钩	71

执行挂钩的类型	71
关于自定义执行挂钩的重要说明	72
执行挂钩筛选器	73
执行挂钩示例	73
创建执行挂钩	73
手动运行执行挂钩	75

管理和保护应用程序

使用 Trident Protect AppVault 对象管理存储桶

Trident Protect 的存储桶自定义资源 (CR) 称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含在保护操作（如备份、快照、还原操作和 SnapMirror 复制）中使用存储桶所需的配置。只有管理员可以创建 AppVaults。

在对应用程序执行数据保护操作时，您需要手动或从命令行创建 AppVault CR。AppVault CR 特定于您的环境，您可以在创建 AppVault CR 时使用此页面上的示例作为指南。



确保 AppVault CR 位于安装 Trident Protect 的集群上。如果 AppVault CR 不存在或您无法访问它，则命令行将显示错误。

配置 AppVault 身份验证和密码

在创建 AppVault CR 之前，请确保您选择的 AppVault 和数据移动器可以通过提供商和任何相关资源进行身份验证。

Data Mover 存储库密码

当您使用 CR 或 Trident Protect CLI 插件创建 AppVault 对象时，您可以为 Restic 和 Kopia 加密指定带有自定义密码的 Kubernetes secret。如果您未指定 secret，Trident Protect 将使用默认密码。

- 手动创建 AppVault CR 时，请使用 `spec.dataMoverPasswordSecretRef` 字段指定密码。
- 使用 Trident Protect CLI 创建 AppVault 对象时，请使用 `--data-mover-password-secret-ref` 参数指定密码。

创建数据移动器存储库密码机密

使用以下示例创建密码密钥。创建 AppVault 对象时，可以指示 Trident Protect 使用此密钥向数据移动器存储库进行身份验证。



- 根据您使用的数据移动器，您只需要包括该数据移动器的相应密码。例如，如果您正在使用 Restic 并且不打算将来使用 Kopia，则在创建密码时可以仅包含 Restic 密码。
- 请将密码保存在安全的地方。您将需要它来恢复同一个集群或另一个集群上的数据。如果集群或 ``trident-protect`` 命名空间被删除，如果没有密码，您将无法还原备份或快照。

使用 CR

```
---
apiVersion: v1
data:
  KOPA_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 兼容存储 IAM 权限

当您使用 Trident Protect 访问 S3 兼容存储（如 Amazon S3、Generic S3 "[StorageGrid S3](#)"或 "[ONTAP S3](#)"）时，您需要确保提供的用户凭据具有访问存储桶所需的权限。以下是授予使用 Trident Protect 访问所需的最低权限的策略示例。您可以将此策略应用于管理 S3 兼容存储桶策略的用户。

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

有关 Amazon S3 策略的详细信息，请参见 "[Amazon S3 文档](#)" 中的示例。

EKS Pod Identity for Amazon S3 (AWS) 身份验证

Trident Protect 支持用于 Kopia 数据移动器操作的 EKS Pod Identity。此功能可实现对 S3 存储桶的安全访问，而无需将 AWS 凭据存储在 Kubernetes 密钥中。

EKS Pod Identity 与 Trident Protect 的要求

在将 EKS Pod Identity 与 Trident Protect 一起使用之前，请确保以下内容：

- 您的 EKS 集群已启用 Pod Identity。
- 您已创建具有必要 S3 存储桶权限的 IAM 角色。要了解更多信息，请参阅 ["S3 兼容存储 IAM 权限"](#)。
- IAM 角色与以下 Trident Protect 服务帐户关联：
 - <trident-protect>-controller-manager
 - <trident-protect>-resource-backup
 - <trident-protect>-resource-restore
 - <trident-protect>-resource-delete

有关启用 Pod Identity 以及将 IAM 角色与服务帐户关联的详细说明，请参见 ["AWS EKS Pod Identity 文档"](#)。

AppVault 配置 使用 EKS Pod Identity 时，使用 `useIAM: true` 标志而不是显式凭据配置 AppVault CR：

```
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
```

针对云提供商的 **AppVault** 密钥生成示例

定义 AppVault CR 时，需要包括访问提供程序托管的资源的凭据，除非您正在使用 IAM 身份验证。您为凭据生成密钥的方式将根据提供程序而有所不同。以下是多个提供程序的命令行密钥生成示例。您可以使用以下示例为每个云提供商的凭据创建密钥。

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 创建示例

下面是每个提供程序的示例 AppVault 定义。

AppVault CR 示例

您可以使用以下 CR 示例为每个云提供商创建 AppVault 对象。



- 您可以选择指定 Kubernetes 机密，其中包含 Restic 和 Kopia 存储库加密的自定义密码。有关详细信息，请参见[Data Mover 存储库密码](#)。
- 对于 Amazon S3 (AWS) AppVault 对象，您可以选择指定 sessionToken，这在使用单点登录 (SSO) 进行身份验证时非常有用。当您在 [针对云提供商的 AppVault 密钥生成示例](#) 中为提供

商生成密钥时，将创建此令牌。

- 对于 S3 AppVault 对象，您可以选择使用 `spec.providerConfig.S3.proxyURL` 密钥为出站 S3 流量指定出口代理 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
```



对于使用 Pod Identity 和 Kopia data mover 的 EKS 环境，你可以移除 `providerCredentials` 部分，并将 `useIAM: true` 添加到 `s3` 配置下。

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

```
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 创建 AppVault 示例

您可以使用以下 CLI 命令示例为每个提供程序创建 AppVault CR。



- 您可以选择指定 Kubernetes 机密，其中包含 Restic 和 Kopia 存储库加密的自定义密码。有关详细信息，请参见[Data Mover 存储库密码](#)。
- 对于 S3 AppVault 对象，您可以选择使用 `--proxy-url <ip_address:port>` 参数为出站 S3 流量指定出口代理 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
```

支持的 `providerConfig.s3` 配置选项

有关 S3 提供程序配置选项，请参见下表：

参数	说明	默认	示例
providerConfig.s3.skipCertValidation	禁用 SSL/TLS 证书验证。	false	"true"、"false"
providerConfig.s3.secure	启用与 S3 端点的安全 HTTPS 通信。	true	"true"、"false"
providerConfig.s3.proxyURL	指定用于连接到 S3 的代理服务器的 URL。	无	http://proxy.example.com:8080
providerConfig.s3.rootCA	为 SSL/TLS 验证提供自定义根 CA 证书。	无	"CN=MyCustomCA"
providerConfig.s3.useIAM	启用 IAM 身份验证以访问 S3 存储桶。适用于 EKS Pod Identity。	false	true, false

查看 AppVault 信息

您可以使用 Trident Protect CLI 插件查看有关在集群上创建的 AppVault 对象的信息。

步骤

1. 查看 AppVault 对象的内容：

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

示例输出：

```
+-----+-----+-----+-----+
+-----+
| 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. (可选) 要查看每个资源的 AppVaultPath, 请使用标志 `--show-paths`。

只有在 Trident Protect helm 安装中指定了集群名称时, 表格第一列中的集群名称才可用。例如: `--set clusterName=production1`。

移除 AppVault

您可以随时删除 AppVault 对象。



在删除 AppVault 对象之前, 请勿移除 AppVault CR 中的 `'finalizers'` 密钥。如果这样做, 可能会导致 AppVault 存储区中的剩余数据和集群中的孤立资源。

开始之前

确保已删除要删除的 AppVault 使用的所有快照和备份 CR。

使用 Kubernetes CLI 删除 AppVault

1. 删除 AppVault 对象, 将 `appvault-name` 替换为要删除的 AppVault 对象的名称:

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

使用 Trident Protect CLI 删除 AppVault

1. 删除 AppVault 对象, 将 `appvault-name` 替换为要删除的 AppVault 对象的名称:

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

使用 Trident Protect 定义管理应用程序

您可以通过创建应用程序 CR 和相关 AppVault CR 来定义要使用 Trident Protect 管理的应用程序。

创建 AppVault CR

您需要创建将在对应用程序执行数据保护操作时使用的 AppVault CR，并且 AppVault CR 需要驻留在安装 Trident Protect 的集群上。AppVault CR 特定于您的环境；有关 AppVault CR 的示例，请参阅 ["AppVault 自定义资源"](#)。

定义应用程序

您需要定义要使用 Trident Protect 管理的每个应用程序。您可以通过手动创建应用程序 CR 或使用 Trident Protect CLI 来定义用于管理的应用程序。



要定义通过名称而不是整个命名空间来保护特定 KubeVirt 虚拟机 (VM) 的 VM 范围应用程序，请使用应用程序 CR 中的 `spec.includedVirtualMachines` 字段或 CLI 中的 `--virtual-machines` 标志。有关详细信息、字段参考和 YAML 示例，请参阅 ["使用 Trident Protect 保护 KubeVirt 虚拟机"](#)。

使用 CR 添加应用程序

步骤

1. 创建目标应用程序 CR 文件：

- a. 创建自定义资源 (CR) 文件并将其命名（例如，`maria-app.yaml`）。
- b. 配置以下属性：
 - **metadata.name:** (必需) 应用程序自定义资源的名称。请注意您选择的名称，因为保护操作所需的其他 CR 文件会引用此值。
 - **spec.includedNamespaces:** (*Required*) 使用命名空间和标签选择器指定应用程序使用的命名空间和资源。应用程序命名空间必须是此列表的一部分。标签选择器是可选的，可用于筛选每个指定命名空间内的资源。
 - **spec.includedClusterScopedResources:** (*Optional*) 使用此属性指定要包含在应用程序定义中的群集范围的资源。此属性允许您根据其组、版本、种类和标签选择这些资源。
 - **groupVersionKind:** (必需) 指定集群范围内资源的 API 组、版本和种类。
 - **labelSelector:** (可选) 根据集群范围资源的标签对其进行筛选。
 - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze:** (*Optional*) 此注释仅适用于从虚拟机定义的应用程序，例如在 KubeVirt 环境中，在快照之前会发生文件系统冻结。指定此应用程序是否可以在快照期间写入文件系统。如果设置为 `true`，应用程序将忽略全局设置，并且可以在快照期间写入文件系统。如果设置为 `false`，应用程序将忽略全局设置，并在快照期间冻结文件系统。如果指定，但应用程序在应用程序定义中没有虚拟机，则忽略注释。如果未指定，则应用程序遵循 ["全局 Trident Protect 冻结设置"](#)。

如果需要在已创建应用程序后应用此批注，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"
```

+
示例 YAML:

+

```
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. (Optional) 如果需要，可以将资源筛选添加到同一 CR 以包括或排除特定资源：

◦ 通用过滤器示例：

- **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include` 或 `Exclude` 来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
- **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作匹配，并且每个元素（group、kind、version）内的字段将作为 AND 操作匹配。

- **resourceMatchers[].group**: (*Optional*) 要筛选的资源的组。
- **resourceMatchers[].kind**: (*Optional*) 要筛选的资源类型。
- **resourceMatchers[].version**: (*Optional*) 要筛选的资源版本。
- **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes metadata.name 字段中的名称。
- **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "[Kubernetes 文档](#)" 中所定义。例如：
"trident.netapp.io/os=linux"。



当同时使用 `resourceFilter` 和 `labelSelector` 时，`resourceFilter` 首先运行，然后将 `labelSelector` 应用于生成的资源。

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"]
```

◦ 仅 PVC 过滤器示例：

要定义仅限 PVC 的应用程序，还必须在资源筛选器中包含 PersistentVolume 和 VolumeSnapshotClass。快照和备份操作依赖于 PersistentVolume（绑定到每个 PVC 的集群范围卷）和 VolumeSnapshotClass（快照驱动程序），如果没有它们，则会失败。例如：

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

```
spec:
  includedNamespaces:
  - namespace: my-app-namespace
  resourceFilter:
    resourceMatchers:
    - kind: PersistentVolumeClaim
      version: v1
    - kind: PersistentVolume
      version: v1
    - kind: VolumeSnapshotClass
      version: v1
    resourceSelectionCriteria: Include
```

2. 创建与环境匹配的应用程序 CR 后，应用 CR。例如：

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

步骤

1. 使用以下示例之一创建并应用应用程序定义，使用环境中的信息替换括号中的值。可以使用逗号分隔的列表和示例中显示的参数在应用程序定义中包含命名空间和资源。

在创建应用程序时，您可以选择使用注释来指定应用程序是否可以在快照期间写入文件系统。这仅适用于从虚拟机定义的应用程序，例如在 KubeVirt 环境中，在快照之前会发生文件系统冻结。如果将注释设置为 `true`，应用程序将忽略全局设置，并且可以在快照期间写入文件系统。如果将其设置为 `false`，应用程序将忽略全局设置，并在快照期间冻结文件系统。如果使用注释，但应用程序在应用程序定义中没有虚拟机，则该注释将被忽略。如果不使用注释，应用程序将遵循[全局 Trident Protect 冻结设置](#)。

+ 使用 CLI 创建应用程序时，可以使用 `--annotation` 标志指定注释。

+ * 创建应用程序并使用文件系统冻结行为的全局设置：

+

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

+ * 创建应用程序并配置文件系统冻结行为的本地应用程序设置：

+

```
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">
```

+ * 可以使用 `--resource-filter-include` 和 `--resource-filter-exclude` 标志根据 `resourceSelectionCriteria`（例如组、种类、版本、标签、名称和命名空间）来包含或排除资源，如以下示例所示：

+

```
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" ] } ]'
```

+ * 要定义仅限 PVC 的应用程序，您还必须在资源筛选器中包含 `PersistentVolume` 和 `VolumeSnapshotClass`。快照和备份操作依赖于 `PersistentVolume`（绑定到每个 PVC 的集群范围卷）和 `VolumeSnapshotClass`（快照驱动程序），如果没有它们，则会失败。例如：

+

```
tridentctl-protect create app my-pvc-app --namespaces <my-app-
namespace> --resource-filter-include
'[{ "Kind": "PersistentVolumeClaim", "Version": "v1" }, { "Kind": "PersistentVo
lume", "Version": "v1" }, { "Kind": "VolumeSnapshotClass", "Version": "v1" } ]'
-n <my-app-namespace>
```

使用 Trident Protect 保护应用程序

您可以通过使用自动保护策略或临时拍摄快照和备份来保护 Trident Protect 管理的所有应用。



您可以配置 Trident Protect 在数据保护操作期间冻结和解冻文件系统。[详细了解如何使用 Trident Protect 配置文件系统冻结](#)。

创建按需快照

您可以随时创建按需快照。



如果集群范围的资源在应用程序定义中显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、快照或克隆中。

使用 **CR** 创建快照
步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name**: (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.applicationRef**: 要快照的应用程序的 Kubernetes 名称。
 - **spec.appVaultRef**: (必需) 应存储快照内容 (元数据) 的 AppVault 的名称。
 - **spec.reclaimPolicy**: (可选) 定义删除快照 CR 时快照的 AppArchive 会发生什么情况。这意味着即使设置为 Retain，快照也将被删除。有效选项：
 - Retain (默认)
 - 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. 使用正确的值填充 `'trident-protect-snapshot-cr.yaml'` 文件后，应用 CR:

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

使用 CLI 创建快照

步骤

1. 创建快照，用环境中的信息替换括号中的值。例如：

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

创建按需备份

您可以随时备份应用程序。



如果集群范围的资源在应用程序定义中显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、快照或克隆中。

开始之前

确保 AWS 会话令牌过期时间足以支持任何长时间运行的 s3 备份操作。如果令牌在备份操作期间过期，操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
- 有关 AWS 资源凭据的详细信息，请参见 ["AWS IAM 文档"](#)。

使用 CR 创建备份

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.applicationRef:** (必需) 要备份的应用程序的 Kubernetes 名称。
 - **spec.appVaultRef:** (必需) 应存储备份内容的 AppVault 的名称。
 - **spec.dataMover:** (*Optional*) 一个字符串，指示要用于备份操作的备份工具。可能的值（区分大小写）：
 - `Restic`
 - `Kopia` (默认)
 - **spec.reclaimPolicy:** (可选) 定义从声明中释放备份时会发生什么。可能的值：
 - `Delete`
 - `Retain` (默认)
 - **spec.snapshotRef:** (*Optional*): 要用作备份源的快照的名称。如果未提供，将创建并备份临时快照。

示例 YAML:

```
---
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. 使用正确的值填充 `trident-protect-backup-cr.yaml` 文件后，应用 CR:

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

使用 CLI 创建备份

步骤

1. 创建备份，用环境中的信息替换括号中的值。例如：

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

您可以选择使用 `--full-backup` 标志来指定备份是否应为非增量备份。默认情况下，所有备份都是增量备份。使用此标志时，备份将变为非增量备份。最佳做法是定期执行完整备份，然后在两次完整备份之间执行增量备份，以最大限度地降低与恢复相关的风险。

支持的备份注释

下表介绍了创建备份 CR 时可以使用的批注：

标注	类型	说明	默认值
protect.trident.netapp.io/full-backup	string	指定备份是否应为非增量备份。设置为 `true` 以创建非增量备份。最佳做法是定期执行完整备份，然后在两次完整备份之间执行增量备份，以最大限度地降低与恢复相关的风险。	"false"
protect.trident.netapp.io/snapshots-hot-completion-timeout	string	整个快照操作完成所允许的最长时间。	"60 分钟"
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	允许卷快照达到准备就绪状态的最长时间。	"30 分钟"
protect.trident.netapp.io/volume-snapshots-created-timeout	string	允许创建卷快照的最长时间。	"5 分钟"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	在操作失败之前，等待任何新创建的 PersistentVolumeClaims (PVC) 到达 `Bound` 阶段的最长时间（以秒为单位）。	"1200"（20 分钟）

创建数据保护计划

保护策略通过按定义的时间表创建快照、备份或同时创建快照和备份来保护应用。您可以选择每小时、每天、每周和每月创建快照和备份，还可以指定要保留的副本数。您可以使用 `full-backup-rule` 注释安排非增量完整备份。默认情况下，所有备份都是增量备份。定期执行完整备份以及之间的增量备份有助于降低与恢复相关的风险。



- 您只能通过将 `backupRetention` 设置为零并将 `snapshotRetention` 设置为大于零的值来创建快照计划。将 `snapshotRetention` 设置为零意味着任何计划备份仍将创建快照，但这些快照是临时的，并在备份完成后立即删除。
- 如果集群范围的资源在应用程序定义中显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、快照或克隆中。

使用 CR 创建计划

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-schedule-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.dataMover:** (*Optional*) 一个字符串，指示要用于备份操作的备份工具。可能的值（区分大小写）：
 - Restic
 - Kopia (默认)
 - **spec.applicationRef:** 要备份的应用程序的 Kubernetes 名称。
 - **spec.appVaultRef:** (必需) 应存储备份内容的 AppVault 的名称。
 - **spec.backupRetention:** (*Required*) 要保留的备份数量。零表示不应创建备份（仅限快照）。
 - **spec.backupReclaimPolicy:** (可选) 确定如果备份 CR 在其保留期间被删除，备份会发生什么情况。保留期限过后，始终会删除备份。可能的值（区分大小写）：
 - Retain (默认)
 - Delete
 - **spec.snapshotRetention:** (*Required*) 要保留的快照数。零表示不应创建快照。
 - **spec.snapshotReclaimPolicy:** (可选) 确定如果快照 CR 在其保留期间被删除，快照会发生什么情况。保留期过后，快照始终被删除。可能的值（区分大小写）：
 - Retain
 - Delete (默认)
 - **spec.granularity:** 计划应运行的频率。可能的值以及必需的相关字段：
 - Hourly (要求您指定 `spec.minute`)
 - Daily (要求您指定 `spec.minute` 和 `spec.hour`)
 - Weekly (要求您指定 `spec.minute`, `spec.hour` 和 `spec.dayOfWeek`)
 - Monthly (要求您指定 `spec.minute`, `spec.hour` 和 `spec.dayOfMonth`)
 - Custom
 - **spec.dayOfMonth:** (*Optional*) 应运行计划的月份日期 (1 - 31)。如果粒度设置为 Monthly，则此字段为必填字段。该值必须以字符串形式提供。
 - **spec.dayOfWeek:** (*Optional*) 应运行计划的星期几 (0 - 7)。值 0 或 7 表示星期天。如果粒度设置为 Weekly，则此字段为必填字段。该值必须以字符串形式提供。
 - **spec.hour:** (*Optional*) 应运行计划的一天中的小时 (0 - 23)。如果粒度设置为 Daily、Weekly 或 Monthly，则此字段为必填字段。该值必须以字符串形式提供。
 - **spec.minute:** (*Optional*) 计划应运行的小时中的分钟数 (0 - 59)。如果粒度设置为 Hourly、Daily、Weekly 或 Monthly，则此字段为必填字段。该值必须以字符串形式提供。
 - **spec.runImmediately:** (可选) 设置为 `true` 以在创建计划时触发一次性的即时基线运行（根据

保留设置进行备份和/或快照)。默认为 `false`。这不会修改后续重复周期。

备份和快照计划的 YAML 示例:

```
---
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"
```

仅快照计划的 YAML 示例:

```
---
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"
```

立即运行的计划的示例 YAML:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
```

```

name: my-daily-schedule-run-immediately
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "7"
  snapshotRetention: "7"
  granularity: Daily
  hour: "3"
  minute: "0"
  runImmediately: true

```

3. 使用正确的值填充 `trident-protect-schedule-cr.yaml` 文件后，应用 CR：

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

使用 CLI 创建计划

步骤

1. 创建保护计划，用环境中的信息替换括号中的值。例如：



您可以使用 `tridentctl-protect create schedule --help` 查看此命令的详细帮助信息。

```

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>

```

以下标记为您的日程安排提供了额外的控制：

- 完全备份计划：使用 `--full-backup-rule` 标志来计划非增量完全备份。此标志仅适用于 `--granularity Daily`。可能的值：

- **Always**: 每天创建完整备份。
- **特定工作日**: 指定一个或多个以逗号分隔的日期（例如，`"Monday,Thursday"`）。有效值：`Monday`、`Tuesday`、`Wednesday`、`Thursday`、`Friday`、`Saturday`、`Sunday`。



此 `--full-backup-rule` 标记不适用于每小时、每周或每月粒度。

- **立即基线保护**: 使用 `--run-immediately true` 在创建计划时立即创建初始备份或快照，而不是等待第一个计划的运行时间。默认值为 `false`。
- **仅快照计划**: 设置 `--backup-retention 0` 并为 `--snapshot-retention` 指定一个大于零的值。

支持的计划注释

下表介绍了创建计划 CR 时可以使用的批注：

标注	类型	说明	默认值
<code>protect.trident.netapp.io/full-backup-rule</code>	string	指定计划完整备份的规则。您可以将其设置为 <code>Always</code> 以进行持续完整备份，也可以根据您的要求进行自定义。例如，如果选择每日粒度，则可以指定应进行完整备份的工作日（例如， <code>"Monday,Thursday"</code> ）。有效的工作日值为： <code>Monday</code> 、 <code>Tuesday</code> 、 <code>Wednesday</code> 、 <code>Thursday</code> 、 <code>Friday</code> 、 <code>Saturday</code> 、 <code>Sunday</code> 。请注意，此批注只能用于将 <code>'granularity'</code> 设置为 <code>'Daily'</code> 的计划。	未设置（所有备份都是增量备份）
<code>protect.trident.netapp.io/snapshots-hot-completion-timeout</code>	string	整个快照操作完成所允许的最长时间。	"60 分钟"
<code>protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout</code>	string	允许卷快照达到准备就绪状态的最长时间。	"30 分钟"
<code>protect.trident.netapp.io/volume-snapshots-created-timeout</code>	string	允许创建卷快照的最长时间。	"5 分钟"
<code>protect.trident.netapp.io/pvc-bind-timeout-sec</code>	string	在操作失败之前，等待任何新创建的 <code>PersistentVolumeClaims (PVC)</code> 到达 <code>'Bound'</code> 阶段的最长时间（以秒为单位）。	"1200"（20 分钟）

删除快照

删除不再需要的计划快照或按需快照。

步骤

1. 删除与快照关联的快照 CR：

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

删除备份

删除不再需要的计划备份或按需备份。



确保将回收策略设置为 `Delete` 以从对象存储中删除所有备份数据。策略的默认设置是 `Retain` 以避免意外数据丢失。如果策略未更改为 `Delete`，则备份数据将保留在对象存储中，并需要手动删除。

步骤

1. 删除与备份关联的备份 CR：

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

检查备份操作的状态

您可以使用命令行检查正在进行、已完成或已失败的备份操作的状态。

步骤

1. 使用以下命令可检索备份操作的状态，用环境中的信息替换括号中的值：

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

为 **azure-netapp-files (ANF)** 操作启用备份和还原

如果您已安装 Trident Protect，则可以为使用 `azure-netapp-files` 存储类并在 Trident 24.06 之前创建的存储后端启用节省空间的备份和恢复功能。此功能适用于 NFSv4 卷，不会消耗容量池中的额外空间。

开始之前

确保以下内容：

- 您已安装 Trident Protect。
- 您已在 Trident Protect 中定义了一个应用程序。在完成此过程之前，此应用程序将具有有限的保护功能。
- 您已选择 `azure-netapp-files` 作为存储后端的默认存储类。

▼ 展开以查看配置步骤

1. 如果 ANF 卷是在升级到 Trident 24.10 之前创建的，请在 Trident 中执行以下操作：
 - a. 为每个基于 `azure-netapp-files` 并与应用程序关联的 PV 启用快照目录：

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

- b. 确认已为每个关联的 PV 启用快照目录：

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

响应：

```
snapshotDirectory: "true"
```

+

未启用快照目录时，Trident Protect 会选择常规备份功能，该功能在备份过程中会暂时占用容量池中的空间。在这种情况下，请确保容量池中有足够的空间可用于创建要备份的卷大小的临时卷。

结果

该应用程序可以使用 Trident Protect 进行备份和还原。每个 PVC 也可供其他应用程序用于备份和还原。

还原应用程序

使用 Trident Protect 还原应用程序

您可以使用 Trident Protect 从快照或备份恢复您的应用程序。将应用程序还原到同一集群时，从现有快照还原将更快。



- 还原应用程序时，为应用程序配置的所有执行挂钩都会随应用程序一起还原。如果存在还原后执行挂钩，它将作为还原操作的一部分自动运行。
- qtree 卷支持从备份还原到其他命名空间或原始命名空间。但是，qtree 卷不支持从快照还原到其他命名空间或原始命名空间。
- 您可以使用高级设置自定义还原操作。要了解更多信息，请参阅 ["使用高级 Trident Protect 还原设置"](#)。
- 要从存档还原特定 KubeVirt 虚拟机 (VM) 而不还原所有命名空间资源，请使用还原 CR 上的 `includedVirtualMachines` 字段。有关支持的 CR、字段引用和 YAML 示例，请参阅 ["使用 Trident Protect 保护 KubeVirt 虚拟机"](#)。

从备份还原到其他命名空间

使用 BackupRestore CR 将备份还原到其他命名空间时，Trident Protect 会在新命名空间中还原应用程序，并为还原的应用程序创建应用程序 CR。要保护还原的应用程序，请创建按需备份或快照，或建立保护计划。



- 使用现有资源将备份还原到其他命名空间不会更改与备份中的名称共享的任何资源。要还原备份中的所有资源，请删除并重新创建目标命名空间，或将备份还原到新命名空间。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
- 有关 AWS 资源凭据的详细信息，请参见 ["AWS IAM 文档"](#)。



使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 ["Kopia 文档"](#)。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 ``tridentctl-protect create --help`` 命令。

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-restore-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appArchivePath:** AppVault 中存储备份内容的路径。您可以使用以下命令查找此路径：

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

- **spec.appVaultRef:** (必需) 存储备份内容的 AppVault 的名称。
- **spec.destinationApplicationName:** (可选) 已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
- **spec.namespaceMapping:** 还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`` 和 ``my-destination-namespace`。

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

3. (Optional) 如果仅需要选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：



Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了持久卷声明资源，且该资源具有关联的 pod，则 Trident Protect 也将还原该关联的 pod。

- **resourceFilter.resourceSelectionCriteria:** (筛选时需要) 使用 `Include` 或 `Exclude` 来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源:
- **resourceFilter.resourceMatchers:** resourceMatcher 对象数组。如果在此数组中定义多个元素, 则它们将作为 OR 操作匹配, 并且每个元素 (group、kind、version) 内的字段将作为 AND 操作匹配。
 - **resourceMatchers[].group:** (*Optional*) 要筛选的资源的组。
 - **resourceMatchers[].kind:** (*Optional*) 要筛选的资源的类型。
 - **resourceMatchers[].version:** (*Optional*) 要筛选的资源的版本。
 - **resourceMatchers[].names:** (可选) 要过滤的资源的 Kubernetes metadata.name 字段中的名称。
 - **resourceMatchers[].namespaces:** (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
 - **resourceMatchers[].labelSelectors:** (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串, 如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-backup-restore-cr.yaml` 文件后, 应用 CR:

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

使用 CLI

步骤

1. 将备份还原到其他命名空间, 用环境中的信息替换括号中的值。 namespace-mapping 参数使用冒号

分隔的命名空间将源命名空间映射到格式为 `source1:dest1,source2:dest2` 的正确目标命名空间。例如：

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

从备份还原到原始命名空间

您可以随时将备份还原到原始命名空间。当您执行就地恢复时，Trident Protect 会自动管理保护计划和正在进行的操作，以防止无效的恢复点：

- 在开始还原之前，将禁用应用程序的所有已启用的保护计划。这会阻止在还原应用程序资源时运行定时备份或快照。
- 成功完成还原后，仅重新启用还原之前启用的计划。已禁用的计划将保持禁用状态。
- 在恢复开始之前，任何正在进行的备份或快照操作都将被取消。如果操作未在 5 分钟内取消，还原将继续，并在还原 CR 状态下记录警告。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
- 有关 AWS 资源凭据的详细信息，请参见 ["AWS IAM 文档"](#)。



使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 ["Kopia 文档"](#)。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 `tridentctl-protect create --help` 命令。

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-ipr-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name**： (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appArchivePath**： AppVault 中存储备份内容的路径。您可以使用以下命令查找此路径：

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

- **spec.appVaultRef:** (必需) 存储备份内容的 AppVault 的名称。

例如:

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

3. (Optional) 如果仅需要选择要还原的应用程序的某些资源, 请添加包含或排除标有特定标签的资源的筛选:



Trident Protect 会自动选择一些资源, 因为它们与您选择的资源之间存在关系。例如, 如果您选择了持久卷声明资源, 且该资源具有关联的 pod, 则 Trident Protect 也将还原该关联的 pod。

- **resourceFilter.resourceSelectionCriteria:** (筛选时需要) 使用 `Include` 或 `Exclude` 来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源:
 - **resourceFilter.resourceMatchers:** resourceMatcher 对象数组。如果在此数组中定义多个元素, 则它们将作为 OR 操作匹配, 并且每个元素 (group、kind、version) 内的字段将作为 AND 操作匹配。
 - **resourceMatchers[].group:** (Optional) 要筛选的资源的组。
 - **resourceMatchers[].kind:** (Optional) 要筛选的资源的类型。
 - **resourceMatchers[].version:** (Optional) 要筛选的资源的版本。
 - **resourceMatchers[].names:** (可选) 要过滤的资源的 Kubernetes metadata.name 字段中的名称。
 - **resourceMatchers[].namespaces:** (Optional) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
 - **resourceMatchers[].labelSelectors:** (Optional) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串, 如 "[Kubernetes 文档](#)" 中所定义。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. 使用正确的值填充 `trident-protect-backup-ipr-cr.yaml` 文件后，应用 CR：

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

使用 CLI

步骤

1. 将备份还原到原始命名空间，用环境中的信息替换括号中的值。backup 参数使用格式为 `<namespace>/<name>` 的命名空间和备份名称。例如：

```

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

```

从备份还原到其他集群

如果原始群集出现问题，可以将备份还原到其他群集。



- 使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 ["Kopia 文档"](#)。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 `tridentctl-protect create --help` 命令。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

开始之前

确保满足以下先决条件：

- 目标集群已安装 Trident Protect。
- 目标集群可以访问与源集群相同的 AppVault 存储桶路径，备份存储在该路径中。
- 运行 `tridentctl-protect get appvaultcontent` 命令时，确保您的本地环境可以连接到 AppVault

CR 中定义的对象存储桶。如果网络限制阻止访问，请在目标集群的 pod 内运行 Trident Protect CLI。

- 确保 AWS 会话令牌过期时间足以进行任何长期运行的还原操作。如果令牌在还原操作期间过期，则操作可能会失败。
 - 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
 - 有关 AWS 资源凭据的详细信息，请参见 ["AWS 文档"](#)。

步骤

1. 使用 Trident Protect CLI 插件验证目标集群上是否存在 AppVault CR:

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



如果目标集群上不存在 AppVault CR，请按照 ["使用 Trident Protect AppVault 对象管理存储桶"](#) 中的步骤创建它。

2. 查看目标集群上可用的 AppVault 的备份内容，并记录 `appArchivePath` 要还原的备份:

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

运行此命令会显示 AppVault 中的可用备份，包括其原始群集、相应的应用程序名称、时间戳和存档路径。

输出示例:

```
+-----+-----+-----+-----+
+-----+-----+
| CLUSTER | APP | TYPE | NAME | TIMESTAMP |
| PATH |
+-----+-----+-----+-----+
+-----+-----+
| production1 | wordpress | backup | wordpress-bkup-1 | 2024-10-30
08:37:40 (UTC) | backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2 | 2024-10-30
08:37:40 (UTC) | backuppath2 |
+-----+-----+-----+-----+
+-----+-----+
```

3. 使用 AppVault 名称和存档路径将应用程序还原到目标集群:



使用 CR 时，请确保目标集群上存在用于应用程序还原的命名空间。

使用 CR

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-restore-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appVaultRef:** (必需) 存储备份内容的 AppVault 的名称。
 - **spec.appArchivePath:** (*Required*) AppVault 内存储备份内容的路径。使用步骤 2 中的命令查看备份内容并找到 ``appArchivePath`` 要还原的备份。
 - **spec.destinationApplicationName:** (可选) 已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
 - **spec.namespaceMapping:** 还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`` 和 ``my-destination-namespace`。

例如：

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

3. 使用正确的值填充 ``trident-protect-backup-restore-cr.yaml`` 文件后，应用 CR：

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

使用 CLI

1. 使用以下命令还原应用程序，用环境中的信息替换括号中的值。namespace-mapping 参数使用冒号分隔的命名空间将源命名空间映射到格式为 `source1:dest1,source2:dest2` 的正确目标命名空间。例如：

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--destination-app-name <custom_app_name> \
--context <destination_cluster_name> \
```

```
-n <application_namespace>
```

从快照还原到其他命名空间

您可以使用自定义资源 (CR) 文件将数据从快照还原到其他命名空间或原始源命名空间。使用 SnapshotRestore CR 将快照还原到其他命名空间时，Trident Protect 会在新命名空间中还原应用程序，并为还原的应用程序创建应用程序 CR。要保护还原的应用程序，请创建按需备份或快照，或建立保护计划。



- SnapshotRestore 支持 `spec.storageClassMapping`` 属性，但仅当源和目标存储类使用相同的存储后端时。如果尝试还原到使用不同存储后端的 ``StorageClass`，还原操作将失败。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
- 有关 AWS 资源凭据的详细信息，请参见 ["AWS IAM 文档"](#)。

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-restore-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appVaultRef:** (必需) 存储快照内容的 AppVault 的名称。
 - **spec.appArchivePath:** AppVault 中存储快照内容的路径。您可以使用以下命令查找此路径：

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

- **spec.destinationApplicationName:** (可选) 已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
- **spec.namespaceMapping:** 还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`` 和 ``my-destination-namespace`。

```
---  
apiVersion: protect.trident.netapp.io/v1  
kind: SnapshotRestore  
metadata:
```

```

name: my-cr-name
namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-snapshot-path
  namespaceMapping: [{"source": "my-source-namespace",
"destination": "my-destination-namespace"}]

```

3. (Optional) 如果仅需要选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：



Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了持久卷声明资源，且该资源具有关联的 pod，则 Trident Protect 也将还原该关联的 pod。

- **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include` 或 `Exclude` 来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
 - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作匹配，并且每个元素（group、kind、version）内的字段将作为 AND 操作匹配。
 - **resourceMatchers[].group**：(Optional) 要筛选的资源的组。
 - **resourceMatchers[].kind**：(Optional) 要筛选的资源的类型。
 - **resourceMatchers[].version**：(Optional) 要筛选的资源的版本。
 - **resourceMatchers[].names**：(可选) 要过滤的资源的 Kubernetes metadata.name 字段中的名称。
 - **resourceMatchers[].namespaces**：(Optional) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
 - **resourceMatchers[].labelSelectors**：(Optional) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 ["Kubernetes 文档"](#) 中所定义。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. 使用正确的值填充 `trident-protect-snapshot-restore-cr.yaml` 文件后，应用 CR：

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

使用 CLI

步骤

1. 将快照还原到其他命名空间，用环境中的信息替换括号中的值。
 - snapshot 参数使用格式为 <namespace>/<name> 的命名空间和快照名称。
 - namespace-mapping 参数使用冒号分隔的命名空间将源命名空间映射到格式为 source1:dest1,source2:dest2 的正确目标命名空间。

例如：

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name <custom_app_name> \
-n <application_namespace>
```

从快照还原到原始命名空间

您可以随时将快照还原到原始命名空间。当您执行就地恢复时，Trident Protect 会自动管理保护计划和正在进行的操作，以防止无效的恢复点：

- 在开始还原之前，将禁用应用程序的所有已启用的保护计划。这会阻止在还原应用程序资源时运行定时备份或快照。
- 成功完成还原后，仅重新启用还原之前启用的计划。已禁用的计划将保持禁用状态。
- 在恢复开始之前，任何正在进行的备份或快照操作都将被取消。如果操作未在 5 分钟内取消，还原将继续，并在还原 CR 状态下记录警告。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 ["AWS API 文档"](#)。
- 有关 AWS 资源凭据的详细信息，请参见 ["AWS IAM 文档"](#)。

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-ipr-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name**: (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appVaultRef**: (*必需*) 存储快照内容的 AppVault 的名称。
 - **spec.appArchivePath**: AppVault 中存储快照内容的路径。您可以使用以下命令查找此路径：

```
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
```

+ *(Optional)* 如果仅需要选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：

+ 注意：Trident Protect 会自动选择某些资源，因为这些资源与您所选资源之间存在关联关系。例如，如果您选择了持久卷声明资源，且该资源具有关联的 pod，则 Trident Protect 也将还原该关联的 pod。

+ * **resourceFilter.resourceSelectionCriteria**: (筛选必需) 使用 `Include`` 或 `Exclude`` 以包含或排除在 `resourceMatchers` 中定义的资源。添加以下 `resourceMatchers` 参数以定义要包含或排除的资源： **resourceFilter.resourceMatchers: resourceMatcher** 对象数组。如果在此数组中定义多个元素，则它们将作为 `OR` 操作匹配，并且每个元素 (`group`、`kind`、`version`) 内的字段将作为 `AND` 操作匹配。* **resourceMatchers[].group**: (*Optional*) 要筛选的资源的组。* **resourceMatchers[].kind**: (*Optional*) 要筛选的资源类型。* **resourceMatchers[].version**: (*Optional*) 要筛选的资源的版本。* **resourceMatchers[].names**: (*Optional*) 要筛选的资源的 `Kubernetes metadata.name` 字段中的名称。* **resourceMatchers[].namespaces**: (*Optional*) 要筛选的资源的 `Kubernetes metadata.name` 字段中的命名空间。* **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 `Kubernetes metadata.name` 字段中的标签选择器字符串，如 ["Kubernetes 文档"](#) 中所定义。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"]
```

+ .使用正确的值填充 `trident-protect-snapshot-ipr-cr.yaml` 文件后，应用 CR：

+

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

使用 CLI

步骤

1. 将快照还原到原始命名空间，用环境中的信息替换括号中的值。例如：

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

检查还原操作的状态

您可以使用命令行检查正在进行、已完成或已失败的还原操作的状态。

步骤

1. 使用以下命令可检索还原操作的状态，用环境中的信息替换括号中的值：

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

使用高级 Trident Protect 还原设置

您可以使用高级设置（如注释、命名空间设置和存储选项）自定义还原操作，以满足您的特定要求。

还原和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释匹配。将添加目标命名空间中不存在的源命名空间中的标签或注释，并覆盖已存在的任何标签或注释，以匹配源命名空间中的值。仅存在于目标命名空间上的标签或注释保持不变。



如果使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的关键作用。命名空间注释可确保还原的 Pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。有关详细信息，请参见 ["OpenShift 安全上下文约束文档"](#)。

在执行还原或故障转移操作之前，可以通过设置 Kubernetes 环境变量 `'RESTORE_SKIP_NAMESPACE_ANNOTATIONS'` 来防止目标命名空间中的特定注释被覆盖。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
```



执行还原或故障切换操作时，在 `'restoreSkipNamespaceAnnotations'` 和 `'restoreSkipNamespaceLabels'` 中指定的任何命名空间批注和标签都将从还原或故障切换操作中排除。确保在初始 Helm 安装过程中配置了这些设置。要了解更多信息，请参阅 ["配置其他 Trident Protect helm 图表设置"](#)。

如果使用带有 `--create-namespace` 标志的 Helm 安装源应用程序，则会对 `name` 标签键进行特殊处理。在还原或故障转移过程中，Trident Protect 会将此标签复制到目标命名空间，但如果来自源的值与源命名空间匹配，则将值更新为目标命名空间值。如果此值与源命名空间不匹配，则将其复制到目标命名空间，不进行任何更改。

示例

以下示例显示了源和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作之前和之后的目标命名空间的状态，以及如何在目标命名空间中组合或覆盖注释和标签。

还原或故障转移操作之前

下表显示了还原或故障转移操作之前的示例源和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	<ul style="list-style-type: none"> • annotation.one/key: "updatedvalue" • annotation.two/key: "true" 	<ul style="list-style-type: none"> • environment=production • 合规性=hipaa • name=ns-1
命名空间 ns-2（目标）	<ul style="list-style-type: none"> • annotation.one/key: "true" • annotation.three/key: "false" 	<ul style="list-style-type: none"> • 角色=数据库

还原操作后

下表显示了还原或故障转移操作后示例目标命名空间的状态。已添加一些键，一些键已被覆盖，并且已更新 name 标签以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	<ul style="list-style-type: none"> • annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false" 	<ul style="list-style-type: none"> • name=ns-2 • 合规性=hipaa • environment=production • 角色=数据库

支持的字段

本节描述可用于还原操作的其他字段。

存储类映射

该 `spec.storageClassMapping` 属性定义从源应用程序中存在的存储类到目标集群上的新存储类的映射。您可以在具有不同存储类的集群之间迁移应用程序或更改 BackupRestore 操作的存储后端时使用此功能。

示例：

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

支持的注释

本节列出了用于在系统中配置各种行为的支持注释。如果用户未明确设置注释，系统将使用默认值。

标注	类型	说明	默认值
protect.trident.netapp.io/data-mover-timeout-sec	string	允许数据移动器操作停止的最长时间（以秒为单位）。	"300"

标注	类型	说明	默认值
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	string	Kopia 内容缓存的最大大小限制（以兆字节为单位）。	"1000"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	在操作失败之前，等待任何新创建的 PersistentVolumeClaims (PVC) 到达 `Bound` 阶段的最长时间（以秒为单位）。适用于所有还原 CR 类型（BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore）。如果您的存储后端或集群通常需要更多时间，请使用更高的值。	"1200"（20 分钟）

使用自定义资源还原资源时修改资源

资源转换使您能够在还原资源时对其进行修改。这在还原版本需要与原始版本不同时非常有用，例如，在将应用程序还原到其他集群时更新 ConfigMap 中的主机名。

资源修改的工作原理

`SnapshotRestore`、`BackupRestore`、`AppMirrorRelationship` 以及其他恢复资源中的 `transformations` 字段允许您在恢复过程中修改 Kubernetes 资源。当需要对恢复后的应用程序进行与原始应用程序不同的配置时（例如，更改主机名、注册表 URL、资源限制或环境变量），此功能非常有用。

资源转换使用 "RFC 6902 JSON Patch" 操作和 "RFC 6901 JSON 指针" 路径来定位和修改 Kubernetes 资源中的特定字段。

下面显示了包含资源转换的还原对象的基本结构：

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: my-restore
  namespace: target-namespace
spec:
  appVaultRef: my-vault
  appArchivePath: /path/to/snapshot
  namespaceMapping:
    - source: source-ns
      destination: target-ns
  transformations:
    - resource:
        kind: ConfigMap          # Required: resource kind
        group: ""              # Optional: API group (empty for core
resources)
```

```

version: ""           # Optional: API version
name: ""              # Optional: specific resource name
operations:
  - op: replace        # Operation type
    path: "/data/key"  # JSON Pointer path
    value: "new-value" # New value (for add/replace/test)

```

支持的资源

您可以将资源转换与符合以下标准的资源一起使用：

- **kind** (必填)：Kubernetes 资源类型（例如，ConfigMap、Deployment、Pod）
- **group** (可选)：API 组（例如，apps、route.openshift.io）——省略核心资源
- **version** (可选)：API 版本（例如，v1、v1beta1）
- 名称 (可选)：仅按名称应用于特定资源



当前不支持修改 PersistentVolumeClaims 和命名空间。

支持的操作

您可以使用以下操作修改资源：add、copy、move、remove、replace`和`test。

向资源添加值

使用 add 操作在指定路径上添加新字段或值。您可以将数据添加到对象或数组。

使用 CR

以下示例将节点选择器添加到`Deployment`资源：

```

transformations:
  - resource:
      kind: Deployment
    operations:
      - op: add
        path: "/spec/template/spec/nodeSelector"
        value:
          "topology.kubernetes.io/zone": "us-east-1a"
          disktype: "ssd"

```

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
```

```
--snapshot <namespace/snapshot-name> \
--namespace-mapping <source-ns>:<dest-ns> \
--transformation
'apps,v1,Deployment:add{"path":"/spec/template/spec/nodeSelector","value":{"topology.kubernetes.io/zone":"us-east-1a","disktype":"ssd"}}'
```

复制资源中的值

使用 `copy` 操作将同一资源中的值从一个路径复制到另一个路径。源未更改。

使用 CR

以下示例复制 ConfigMap 对象的数据键：

```
transformations:
- resource:
  kind: ConfigMap
operations:
- op: copy
  from: "/data/source-key"
  path: "/data/backup-key"
```

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
--snapshot <namespace/snapshot-name> \
--namespace-mapping <source-ns>:<dest-ns> \
--transformation ',v1,ConfigMap:copy{"from":"/data/source-key","path":"/data/backup-key"}'
```

在资源内移动值

使用 `move` 操作将同一资源内的值从一个路径移动到另一个路径。源被删除，值被放置在目的地。

使用 CR

以下示例重命名 ConfigMap 对象的数据键：

```
transformations:
- resource:
  kind: ConfigMap
```

```
operations:
  - op: move
    from: "/data/OLD_KEY"
    path: "/data/NEW_KEY"
```

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:move{"from":"/data/OLD_KEY","path":"/data/NEW_KEY"}'
```

从资源中删除值

使用 `remove` 操作可删除指定路径中的字段或值。

使用 CR

以下示例从 `ConfigMap` 资源中删除批注：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: remove
        path: "/metadata/annotations/kubectl.kubernetes.io~1last-
        applied-configuration"
```



在以上示例的路径中，`~1` 是 `/` 的 JSON 指针转义序列。

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:remove{"path":"/metadata/annotations/kubectl.kubernetes.
io~1last-applied-configuration"}'
```

替换资源中的值

使用该 `replace` 操作替换指定路径下资源中的现有值。JSON 路径必须已存在。

使用 CR

以下示例更改 `Route` 对象的主机名：

```
transformations:
- resource:
  kind: Route
  group: route.openshift.io
  operations:
  - op: replace
    path: "/spec/host"
    value: "prod.example.com"
```

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
--snapshot <namespace/snapshot-name> \
--namespace-mapping <source-ns>:<dest-ns> \
--transformation
'route.openshift.io,v1,Route:replace{"path":"/spec/host","value":"prod.
example.com"}'
```

测试资源修改

使用 `test` 操作验证路径上的值是否与预期值匹配，然后再应用其他操作。如果测试失败，则回滚整个修改。

使用 CR

仅当 environment 设置为 staging 时，以下示例才会更新 database-host：

```
transformations:
- resource:
  kind: ConfigMap
  operations:
  - op: test
    path: "/data/environment"
    value: "staging"
  - op: replace
    path: "/data/database-host"
```

```
value: "prod-db.example.com"
```

使用 CLI

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:test{"path":"/data/environment","value":"staging"},repla
ce{"path":"/data/database-host","value":"prod-db.example.com"}'
```

使用 NetApp SnapMirror 和 Trident Protect 复制应用程序

使用 Trident Protect，您可以使用 NetApp SnapMirror 技术的异步复制功能将数据和应用程序更改从一个存储后端复制到另一个存储后端，在同一群集上或在不同群集之间。

还原和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释匹配。将添加目标命名空间中不存在的源命名空间中的标签或注释，并覆盖已存在的任何标签或注释，以匹配源命名空间中的值。仅存在于目标命名空间上的标签或注释保持不变。



如果使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的关键作用。命名空间注释可确保还原的 Pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。有关详细信息，请参见 ["OpenShift 安全上下文约束文档"](#)。

在执行还原或故障转移操作之前，可以通过设置 Kubernetes 环境变量 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS` 来防止目标命名空间中的特定注释被覆盖。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
```



执行还原或故障切换操作时，在 `restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 中指定的任何命名空间批注和标签都将从还原或故障切换操作中排除。确保在初始 Helm 安装过程中配置了这些设置。要了解更多信息，请参阅 ["配置其他 Trident Protect helm 图表设置"](#)。

如果使用带有 `--create-namespace` 标志的 Helm 安装源应用程序，则会对 `name` 标签键进行特殊处理。在还原或故障转移过程中，Trident Protect 会将此标签复制到目标命名空间，但如果来自源的值与源命名空间匹配，则将值更新为目标命名空间值。如果此值与源命名空间不匹配，则将其复制到目标命名空间，不进行任何更改。

示例

以下示例显示了源和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作之前和之后的目标命名空间的状态，以及如何在目标命名空间中组合或覆盖注释和标签。

还原或故障转移操作之前

下表显示了还原或故障转移操作之前的示例源和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	<ul style="list-style-type: none">• <code>annotation.one/key: "updatedvalue"</code>• <code>annotation.two/key: "true"</code>	<ul style="list-style-type: none">• <code>environment=production</code>• <code>合规性=hipaa</code>• <code>name=ns-1</code>
命名空间 ns-2（目标）	<ul style="list-style-type: none">• <code>annotation.one/key: "true"</code>• <code>annotation.three/key: "false"</code>	<ul style="list-style-type: none">• <code>角色=数据库</code>

还原操作后

下表显示了还原或故障转移操作后示例目标命名空间的状态。已添加一些键，一些键已被覆盖，并且已更新 `name` 标签以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	<ul style="list-style-type: none">• <code>annotation.one/key: "updatedvalue"</code>• <code>annotation.two/key: "true"</code>• <code>annotation.three/key: "false"</code>	<ul style="list-style-type: none">• <code>name=ns-2</code>• <code>合规性=hipaa</code>• <code>environment=production</code>• <code>角色=数据库</code>



您可以配置 Trident Protect 在数据保护操作期间冻结和解冻文件系统。[详细了解如何使用 Trident Protect 配置文件系统冻结](#)。

在故障转移和反向操作期间执行挂钩

在使用 AppMirror 关系来保护您的应用程序时，在故障转移和反向操作期间，您应该注意与执行挂钩相关的特定行为。

- 在故障转移过程中，执行挂钩会自动从源集群复制到目标集群。您无需手动重新创建它们。故障转移后，执行挂钩出现在应用程序上，并将在任何相关操作期间执行。
- 在反向或反向重新同步期间，应用程序上的任何现有执行挂钩都将被删除。当源应用程序成为目标应用程序时，这些执行挂钩无效并被删除以防止其执行。

要了解有关执行钩子的更多信息，请参阅 ["管理 Trident Protect 执行挂钩"](#)。

设置复制关系

设置复制关系包括以下内容：

- 选择您希望 Trident Protect 拍摄应用快照的频率（包括应用程序的 Kubernetes 资源以及每个应用程序卷的卷快照）
- 选择复制计划（包括 Kubernetes 资源和持久卷数据）
- 设置拍摄快照的时间

步骤

1. 在源集群上，为源应用程序创建 AppVault。根据您的存储提供商，修改 ["AppVault 自定义资源"](#) 中的示例以适应您的环境：

使用 CR 创建 AppVault

- a. 创建自定义资源 (CR) 文件并将其命名（例如，`trident-protect-appvault-primary-source.yaml`）。
- b. 配置以下属性：
 - **metadata.name:** (*Required*) AppVault 自定义资源的名称。记下您选择的名称，因为复制关系所需的其他 CR 文件引用此值。
 - **spec.providerConfig:** （必需）存储使用指定提供程序访问 AppVault 所需的配置。为您的提供程序选择 `bucketName` 和任何其他必要的详细信息。记下您选择的值，因为复制关系所需的其他 CR 文件会引用这些值。有关其他提供程序的 AppVault CR 示例，请参阅 ["AppVault 自定义资源"](#)。
 - **spec.providerCredentials:** (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
 - **spec.providerCredentials.valueFromSecret:** (*Required*) 指示凭据值应来自机密。
 - **key:** (*Required*) 要从中选择的 secret 的有效 key。
 - **name:** (*Required*) 包含此字段值的密钥的名称。必须位于同一命名空间中。
 - **spec.providerCredentials.secretAccessKey:** （必需）用于访问提供程序的访问密钥。**name** 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。
 - **spec.providerType:** （必需）确定提供备份的内容；例如，NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能的值：
 - `aws`
 - `azure`
 - `gcp`
 - `generic-s3`
 - `ontap-s3`
 - `storagegrid-s3`
- c. 使用正确的值填充 `trident-protect-appvault-primary-source.yaml` 文件后，应用 CR：

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

```
trident-protect
```

使用 CLI 创建 AppVault

- a. 创建 AppVault，用环境中的信息替换括号中的值：

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

2. 在源群集上，创建源应用程序 CR：

使用 CR 创建源应用程序

- a. 创建自定义资源 (CR) 文件并将其命名（例如，trident-protect-app-source.yaml）。

- b. 配置以下属性：

- **metadata.name**：（必需）应用程序自定义资源的名称。记下您选择的名称，因为复制关系所需的其他 CR 文件引用此值。
- **spec.includedNamespaces**：（必需）命名空间和相关标签的数组。使用命名空间名称，并可选择使用标签缩小命名空间的范围，以指定此处列出的命名空间中存在的资源。应用程序命名空间必须是此数组的一部分。

YAML 示例：

```
---  
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. 使用正确的值填充 `trident-protect-app-source.yaml` 文件后，应用 CR：

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

使用 CLI 创建源应用程序

- a. 创建源应用程序。例如：

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

3. (可选) 在源集群上, 拍摄源应用程序的快照。此快照用作目标集群上应用程序的基础。如果跳过此步骤, 则需要等待下一个计划快照运行, 以便获得最近的快照。要创建按需快照, 请参阅 ["创建按需快照"](#)。
4. 在源群集上, 创建复制计划 CR:

除了下面提供的时间表外, 建议创建一个单独的每日快照时间表, 保留期为 7 天, 以维护对等 ONTAP 群集之间的共同快照。这可确保快照最多可用 7 天, 但可以根据用户要求自定义保留期。



如果发生故障转移, 系统可以使用这些快照最多 7 天进行反向操作。这种方法使反向过程更快、更高效, 因为只会传输自上次快照以来所做的更改, 而不会传输所有数据。

如果应用程序的现有计划已满足所需的保留要求, 则不需要其他计划。

使用 CR 创建复制计划

a. 为源应用程序创建复制计划:

- i. 创建自定义资源 (CR) 文件并将其命名 (例如, `trident-protect-schedule.yaml`)。
- ii. 配置以下属性:
 - **metadata.name:** (*Required*) 计划自定义资源的名称。
 - **spec.appVaultRef:** (必需) 此值必须与源应用程序的 AppVault 的 `metadata.name` 字段匹配。
 - **spec.applicationRef:** (必需) 此值必须与源应用程序 CR 的 `metadata.name` 字段匹配。
 - **spec.backupRetention:** (*Required*) 此字段为必填字段, 必须将值设置为 0。
 - **spec.enabled:** 必须设置为 `true`。
 - **spec.granularity:** 必须设置为 `Custom`。
 - **spec.recurrenceRule:** 以 UTC 时间定义开始日期和重复间隔。
 - **spec.snapshotRetention:** 必须设置为 2。

示例 YAML:

```
---
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. 使用正确的值填充 `trident-protect-schedule.yaml` 文件后，应用 CR：

```

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

```

使用 CLI 创建复制计划

- a. 创建复制计划，用环境中的信息替换括号中的值：

```

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>

```

示例：

```

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. 在目标群集上，创建与在源群集上应用的 AppVault CR 相同的源应用程序 AppVault CR，并将其命名（例如，trident-protect-appvault-primary-destination.yaml）。

6. 应用 CR：

```

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

```

7. 在目标集群上为目标应用程序创建目标 AppVault CR。根据您的存储提供商，修改 "AppVault 自定义资源" 中的示例以适应您的环境：

- a. 创建自定义资源 (CR) 文件并将其命名（例如，trident-protect-appvault-secondary-destination.yaml）。

b. 配置以下属性：

- **metadata.name:** (*Required*) AppVault 自定义资源的名称。记下您选择的名称，因为复制关系所需的其它 CR 文件引用此值。
- **spec.providerConfig:** (必需) 存储使用指定提供程序访问 AppVault 所需的配置。为您的提供程序选择 `bucketName` 和任何其他必要的详细信息。记下您选择的值，因为复制关系所需的其它 CR 文件会引用这些值。有关其他提供程序的 AppVault CR 示例，请参阅 ["AppVault 自定义资源"](#)。
- **spec.providerCredentials:** (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
 - **spec.providerCredentials.valueFromSecret:** (*Required*) 指示凭据值应来自机密。
 - **key:** (*Required*) 要从中选择的 secret 的有效 key。
 - **name:** (*Required*) 包含此字段值的密钥的名称。必须位于同一命名空间中。
 - **spec.providerCredentials.secretAccessKey:** (必需) 用于访问提供程序的访问密钥。name 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。
- **spec.providerType:** (必需) 确定提供备份的内容；例如，NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能的值：
 - `aws`
 - `azure`
 - `gcp`
 - `generic-s3`
 - `ontap-s3`
 - `storagegrid-s3`

c. 使用正确的值填充 `trident-protect-appvault-secondary-destination.yaml` 文件后，应用 CR：

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

8. 在目标集群上，创建 AppMirrorRelationship CR 文件。



使用 CR 时，请在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

使用 CR 创建 AppMirrorRelationship

a. 创建自定义资源 (CR) 文件并将其命名 (例如，`trident-protect-relationship.yaml`)。

b. 配置以下属性：

- **metadata.name:** (必需) AppMirrorRelationship 自定义资源的名称。
- **spec.destinationAppVaultRef:** (*Required*) 此值必须与目标集群上目标应用程序的 AppVault 名称匹配。
- **spec.namespaceMapping:** (必需) 目标和源命名空间必须与相应应用程序 CR 中定义的应用程序命名空间匹配。

- **spec.sourceAppVaultRef:** (*Required*) 此值必须与源应用程序的 AppVault 名称匹配。
- **spec.sourceApplicationName:** (必需) 此值必须与您在源应用程序 CR 中定义的源应用程序的名称匹配。
- **spec.sourceApplicationUID:** (必需) 此值必须与您在源应用程序 CR 中定义的源应用程序的 UID 匹配。
- **spec.storageClassName:** (可选) 选择集群上有效存储类的名称。存储类必须链接到与源环境对等的 ONTAP 存储虚拟机。如果未提供存储类，则默认情况下将使用集群上的默认存储类。
- **spec.storageClassMapping:** (*Optional*) 将源存储类映射到同一 AMR 中各个 PVC 的目标存储类。每个条目都需要一个 `source`` 和 ``destination`` 存储类名称。设置后，``storageClassMapping`` 将完全覆盖 ``storageClassName``；没有匹配源条目的 PVC 使用集群默认存储类，而不是 `storageClassName`。

在 AppMirrorRelationship CR 中使用 ``storageClassMapping`` 时，请注意以下限制：

- `storageClassMapping` 完全覆盖 `storageClassName`。当两个字段都设置时，`storageClassName` 将被忽略。
- 源存储类在 `storageClassMapping`` 中没有匹配条目的 PVC 将使用集群默认存储类，而非 ``storageClassName``。
- 每个条目都需要同时提供 ``source`` 和 ``destination`` 值。
- 更改目标存储类可能会强制删除并重新创建现有目标 PVC。
- **spec.recurrenceRule:** 以 UTC 时间定义开始日期和重复间隔。

示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-relationship.yaml` 文件后，应用 CR：

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

使用 CLI 创建 AppMirrorRelationship

- a. 创建并应用 AppMirrorRelationship 对象，用环境中的信息替换括号中的值：

```
tridentctl-protect create appmirrorrelationship  
<name_of_appmirrorrelationship> --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> --storageclass-mapping  
<src_class:dst_class> -n <application_namespace>
```



使用 `--storageclass-mapping` 将单个源存储类映射到目标存储类。指定一个或多个以逗号分隔的 `source:destination` 对（例如，`"class-a:class-b,class-c:class-d"`）。使用 `\*:destination-class` 作为通配符，将所有未映射的 PVC 映射到单个目标存储类。设置 `--storageclass-mapping` 后，`--storage-class` 将被忽略。

使用单个目标存储类的示例 (**--storage-class**)：

```
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. (可选) 在目标集群上，检查复制关系的状态和状态：

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

故障转移到目标集群

使用 Trident Protect，您可以将复制的应用程序故障转移到目标集群。此过程停止复制关系，并使应用程序在目标集群上联机。Trident Protect 不会停止源集群上的应用程序（如果它运行正常）。

步骤

1. 在目标群集上，编辑 AppMirrorRelationship CR 文件（例如，trident-protect-relationship.yaml）并将 **spec.desiredState** 的值更改为 Promoted。
2. 保存 CR 文件。
3. 应用 CR：

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

4. （可选）创建故障转移应用程序所需的任何保护计划。
5. （可选）检查复制关系的状态：

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

重新同步故障转移复制关系

重新同步操作会重新建立复制关系。执行重新同步操作后，原始源应用程序将成为正在运行的应用程序，对目标集群上正在运行的应用程序所做的任何更改都将被丢弃。

此过程在重新建立复制之前停止目标集群上的应用。



在故障转移期间写入目标应用程序的任何数据都将丢失。

步骤

1. 可选：在源集群上，创建源应用程序的快照。这可确保捕获源集群的最新更改。
2. 在目标群集上，编辑 AppMirrorRelationship CR 文件（例如，trident-protect-relationship.yaml）并将 spec.desiredState 的值更改为 Established。
3. 保存 CR 文件。
4. 应用 CR：

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

5. 如果在目标集群上创建了任何保护计划来保护故障转移应用程序，请将其删除。任何剩余的计划都会导致卷快照失败。

反向重新同步故障转移的复制关系

当您反向重新同步故障转移复制关系时，目标应用程序将成为源应用程序，而源应用程序将成为目标。在故障转

移期间对目标应用程序所做的更改将被保留。

步骤

1. 在原始目标集群上，删除 AppMirrorRelationship CR。这会导致目标成为源。如果新目标集群上还有任何保护计划，请将其删除。
2. 通过应用最初用于设置关系的 CR 文件到相反的集群来设置复制关系。
3. 确保新目标（原始源集群）配置了两个 AppVault CR。
4. 在相反的集群上设置复制关系，配置相反方向的值。

反向应用程序复制方向

当您反向复制方向时，Trident Protect 会将应用程序移动到目标存储后端，同时继续复制回原始源存储后端。Trident Protect 会停止源应用程序并将数据复制到目标，然后再故障切换到目标应用程序。

在这种情况下，您将交换源和目标。

步骤

1. 在源集群上，创建关闭快照：

使用 **CR** 创建关闭快照

- a. 禁用源应用程序的保护策略计划。
- b. 创建 ShutdownSnapshot CR 文件：
 - i. 创建自定义资源 (CR) 文件并将其命名（例如，trident-protect-shutdownsnapshot.yaml）。
 - ii. 配置以下属性：
 - **metadata.name**: (*Required*) 自定义资源的名称。
 - **spec.AppVaultRef**: (*Required*) 此值必须与源应用程序的 AppVault 的 metadata.name 字段匹配。
 - **spec.ApplicationRef**: (*必需*) 此值必须与源应用程序 CR 文件的 metadata.name 字段匹配。

示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-shutdownsnapshot.yaml` 文件后，应用 CR：

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

使用 **CLI** 创建关闭快照

- a. 创建关机快照，用环境中的信息替换括号中的值。例如：

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

2. 在源集群上，关闭快照完成后，获取关闭快照的状态：

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

3. 在源集群上，使用以下命令查找 **shutdownsnapshot.status.appArchivePath** 的值，并记录文件路径的最后一部分（也称为基准名；这将是最后一个斜杠之后的所有内容）：

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

4. 执行从新目标集群到新源集群的故障切换，并进行以下更改：



在故障转移过程的第 2 步中，将 `spec.promotedSnapshot` 字段包含在 AppMirrorRelationship CR 文件中，并将其值设置为您在上述第 3 步中记录的基准名。

5. 执行 [\[反向重新同步故障转移的复制关系\]](#) 中的反向重新同步步骤。
6. 在新源集群上启用保护计划。

结果

由于反向复制，发生以下操作：

- 为初始源应用的 Kubernetes 资源拍摄一个快照。
- 通过删除应用程序的 Kubernetes 资源（将 PVC 和 PV 保留到位），可以优雅地停止原始源应用程序的 Pod。
- 关闭 Pod 后，将拍摄并复制应用卷的快照。

- SnapMirror 关系已中断，使目标卷准备好读/写。
- 使用原始源应用程序关闭后复制的卷数据，从预关闭快照中还原应用程序的 Kubernetes 资源。
- 反向重新建立复制。

将应用程序故障回切至原始源群集

使用 Trident Protect，您可以使用以下操作序列在故障转移操作后实现"回切"。在此工作流程中，为了还原原始复制方向，Trident Protect 会在反转复制方向之前将任何应用程序更改复制（重新同步）回原始源应用程序。

此过程从已完成故障转移到目标的关系开始，涉及以下步骤：

- 从故障转移状态开始。
- 反向重新同步复制关系。



请勿执行正常的重新同步操作，因为这将丢弃在故障转移过程中写入目标集群的数据。

- 反转复制方向。

步骤

1. 执行 [\[反向重新同步故障转移的复制关系\]](#) 步骤。
2. 执行 [\[反向应用程序复制方向\]](#) 步骤。

删除复制关系

您可以随时删除复制关系。当您删除应用程序复制关系时，会产生两个独立的应用程序，它们之间没有关系。

步骤

1. 在当前目标集群上，删除 AppMirrorRelationship CR：

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

使用 Trident Protect 迁移应用程序

您可以通过恢复备份数据在集群之间迁移应用程序或迁移到不同的存储类。



迁移应用程序时，为应用程序配置的所有执行挂钩都会随应用程序一起迁移。如果存在还原后执行挂钩，它将作为还原操作的一部分自动运行。

备份和还原操作

要为以下方案执行备份和还原操作，可以自动执行特定的备份和还原任务。

克隆到同一集群

要将应用程序克隆到同一集群，请创建快照或备份并将数据恢复到同一集群。

步骤

1. 执行以下操作之一：
 - a. "创建快照"。
 - b. "创建备份"。
2. 在同一集群上，根据您创建的是快照还是备份，执行以下操作之一：
 - a. "从快照还原数据"。
 - b. "从备份还原数据"。

克隆到不同的集群

若要将应用程序克隆到其他集群（执行跨集群克隆），请在源集群上创建备份，然后将备份还原到其他集群。确保已在目标集群上安装 Trident Protect。



您可以使用 "SnapMirror 复制" 在不同集群之间复制应用程序。

步骤

1. "创建备份"。
2. 确保已在目标集群上配置了包含备份的对象存储桶的 AppVault CR。
3. 在目标集群上，"从备份还原数据"。

将应用程序从一个存储类迁移到另一个存储类

您可以通过将备份还原到目标存储类，将应用程序从一个存储类迁移到其他存储类。

例如（从还原 CR 中排除机密）：

```
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 还原快照

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-restore-cr.yaml`。
2. 在创建的文件中，配置以下属性：
 - **metadata.name**: (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.appArchivePath**: AppVault 中存储快照内容的路径。您可以使用以下命令查找此路径：

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

- **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
- **spec.namespaceMapping**: 还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace` 和 `my-destination-namespace`。

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

3. 或者，如果只需要选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：
 - **resourceFilter.resourceSelectionCriteria**: (筛选必需) 使用 `include` 或 `exclude` 来包括或排除在 `resourceMatchers` 中定义的资源。添加以下 `resourceMatchers` 参数以定义要包括或排除的资源：
 - **resourceFilter.resourceMatchers**: `resourceMatcher` 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作匹配，并且每个元素 (`group`、`kind`、`version`) 内的字段将作为 AND 操作匹配。
 - **resourceMatchers[].group**: (*Optional*) 要筛选的资源的组。
 - **resourceMatchers[].kind**: (*Optional*) 要筛选的资源的类型。
 - **resourceMatchers[].version**: (*Optional*) 要筛选的资源的版本。
 - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes `metadata.name` 字段

中的名称。

- **resourceMatchers[].namespaces:** (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors:** (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "[Kubernetes 文档](#)" 中所定义。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. 使用正确的值填充 `trident-protect-snapshot-restore-cr.yaml` 文件后，应用 CR：

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

使用 CLI 还原快照

步骤

1. 将快照还原到其他命名空间，用环境中的信息替换括号中的值。

- snapshot 参数使用格式为 <namespace>/<name> 的命名空间和快照名称。
- namespace-mapping 参数使用冒号分隔的命名空间将源命名空间映射到格式为 source1:dest1,source2:dest2 的正确目标命名空间。

例如：

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

使用 Trident Protect 保护 KubeVirt 虚拟机

Trident Protect 支持您备份和恢复运行在 OpenShift Virtualization 上的 KubeVirt 虚拟机 (VM)。您可以一次性保护一个或多个命名空间中的所有 VM，也可以使用 `includedVirtualMachines` 字段按名称指定特定的 VM。Trident Protect 会在备份期间自动收集所有依赖的 VM 资源，在恢复期间，您可以恢复整个归档，也可以选择性地恢复单个 VM。

KubeVirt 保护模式

Trident Protect 支持两种方式来定义要保护的 KubeVirt 虚拟机。定义应用程序时，请选择一种模式：

模式	字段	说明
基于命名空间	<code>includedNamespaces</code>	保护一个或多个命名空间中的所有匹配资源。在命名空间条目上使用 <code>'labelSelector'</code> 来控制动态包含哪些 VM。
VM 范围	<code>includedVirtualMachines</code>	通过命名空间和名称保护特定 KubeVirt 虚拟机。当 VM 分布在多个命名空间中或不希望保护所有命名空间资源时，请使用此模式。

您必须在 `ApplicationSpec` 中指定恰好一个字段 (`includedNamespaces` 或 `includedVirtualMachines`)。您不能在同一个应用程序中同时使用两者。

这两种模式都支持额外的筛选：

- 使用 `'resourceFilter'` 可包括或排除特定资源。
- 使用 `'includedClusterScopedResources'` 可包含兼容的集群范围的资源，例如 `'StorageClass'` 对象。



Trident Protect 可以在数据保护操作期间冻结和解冻 KubeVirt VM 文件系统，以确保一致性。有关配置详细信息，请参见 ["使用 KubeVirt VM 保护数据"](#)。

备份和快照期间收集的资源

当您为包含 KubeVirt 虚拟机的应用程序创建备份或快照时，Trident Protect 会收集选定的虚拟机资源并自动发现其依赖资源。

集合行为适用于两种情况：

- 使用 `'includedVirtualMachines'` 的 VM 范围应用程序。
- 包含 KubeVirt `VirtualMachine` 资源的基于命名空间的应用程序。

VM 范围的集合

对于 VM 范围的应用程序，Trident Protect 会从 `includedVirtualMachines` 中列出的每个 `VirtualMachine` 开始，并发现还原该 VM 所需的依赖资源。

资源类型	范围	如何发现
VirtualMachine	命名空间	从 `includedVirtualMachines` 列表中直接匹配。
VirtualMachineInstance	命名空间	与 VM 相同的命名空间和名称（如果 VM 正在运行）。
VirtualMachineInstanceType	命名空间	由 `specinstancetype` 在 `kind: VirtualMachineInstanceType` 时引用。
VirtualMachineClusterInstanceType	集群	在 kind 未设置或为 kind: VirtualMachineClusterInstanceType 时由 specinstancetype 引用。
VirtualMachinePreference	命名空间	由 `spec.preference` 在 `kind: VirtualMachinePreference` 时引用。
VirtualMachineClusterPreference	集群	在 kind 未设置或为 kind: VirtualMachineClusterPreference 时由 spec.preference 引用。
DataVolume	命名空间	被引用 spec.template.spec.volumes[].dataVolume.name。
PersistentVolumeClaim	命名空间	从 DataVolume 支持的 PVC 中发现，这些 PVC 使用与 DataVolume 相同的名称； persistentVolumeClaim.claimName 中的直接 PVC 引用； memoryDump.claimName 中的内存转储 PVC；以及 ephemeral.persistentVolumeClaim.claimName 中的临时 PVC。
PersistentVolume	集群	通过使用 `spec.volumeName` 从每个已收集的 PVC 的后备 PV 中发现。
Secret	命名空间	从 `secret.secretName` 中的卷机密发现；containerDisk 映像拉取机密；CloudInitNoCloud 和 CloudInitConfigDrive 用户数据和网络数据机密引用；Sysprep 机密；KernelBoot 容器映像拉取机密；以及 AccessCredentials SSH 或密码机密。
ConfigMap	命名空间	从卷 ConfigMaps 中发现，位于 configMap.name 和 Sysprep ConfigMaps。
ServiceAccount	命名空间	从 `serviceAccount.serviceAccountName` 卷引用中发现。
NetworkAttachmentDefinition	命名空间	当网络名称不包含 / 时，从 spec.template.spec.networks[].multus.networkName 中的 Multus 网络中发现。仅收集相同命名空间的 NetworkAttachmentDefinition 引用。跳过跨命名空间引用。
Pod	命名空间	已为 virt-launcher Pods 通过使用标签选择器发现 kubevirt.io=virt-launcher,vm.kubevirt.io/name=<vmName>。
存储类、Trident 卷和 Trident 后端	集群	收集方式与基于命名空间的应用程序相同，基于为 VM 发现的 PVC。



收集所有 VM 范围的命名空间资源后，Trident Protect 将应用标准 resourceFilter（如果已配置）。然后，集群范围的资源收集将正常进行。

基于命名空间的集合

对于基于命名空间的应用程序，Trident Protect 会扫描选定的命名空间并收集所有匹配的资源。如果发现任何 KubeVirt VirtualMachine 对象，Trident Protect 将自动收集其依赖资源，包括实例类型、首选项和 virt-launcher Pod，与 VM 范围内的应用程序相同。

这意味着，即使使用 `includedNamespaces` 定义了应用程序，您的备份或快照中也会包含特定于 VM 的资源。

使用基于命名空间的应用程序定义保护虚拟机

创建基于命名空间的应用程序时，Trident Protect 会扫描选定的命名空间并收集所有资源，包括任何 KubeVirt VM。对于找到的每个虚拟机，Trident Protect 还会收集备份和还原该虚拟机所需的依赖资源。

基于命名空间的应用程序 CR 示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
```

创建应用程序后，请使用标准的 Trident Protect 工作流进行备份和快照：

- 有关快照 YAML 示例，请参见 ["创建按需快照"](#)。
- 有关备份 YAML 示例，请参见 ["创建按需备份"](#)。

还原基于命名空间的虚拟机应用程序的行为

由于基于命名空间的应用程序从所选命名空间收集资源，因此备份或快照可以包括 KubeVirt 运行时对象和 KubeVirt 托管元数据。其中某些对象不应直接还原，因为它们表示来自源集群的运行时状态。

在恢复期间，Trident Protect 应用 KubeVirt 感知转换来删除源集群元数据并跳过临时 KubeVirt 资源。这有助于 KubeVirt 在恢复的环境中重新创建虚拟机。

资源	转换
VirtualMachine	删除以 `kubevirt.io/` 开头的标签和批注。
VirtualMachineInstance	删除以 `kubevirt.io/` 开头的标签和批注。
VirtualMachineInstanceMigration	忽略而不还原。
VirtualMachineSnapshot	忽略而不还原。
VirtualMachineSnapshotContent	忽略而不还原。
Pod (virt-launcher)	当 Pod 具有 `kubevirt.io=virt-launcher` 标签时跳过。



Trident Protect 以定义的顺序恢复 KubeVirt 资源，以避免依赖性问题。CDI DataVolumes (cdi.kubevirt.io) 将首先还原，以便在创建虚拟机对象之前提供虚拟机磁盘数据。之后，VirtualMachines (kubevirt.io) 将在 VirtualMachineInstances 之前还原。

使用标签选择器实现动态虚拟机保护

如果要在不列出每个虚拟机名称的情况下动态保护虚拟机，请使用带有标签选择器的基于命名空间的应用程序定义。

只有 VirtualMachine CR 需要标签。依赖资源（例如 DataVolumes、PVC、Secrets 和 ConfigMaps）会自动收集。

步骤

1. 使用共享标签标记虚拟机 CR:

```
kubectl label virtualmachine <vm_name_1> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_2> app=<label_value> -n
<vm_namespace>
```

2. 使用 `includedNamespaces` 和 `labelSelector` 创建应用程序 CR:

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
    labelSelector:
      matchLabels:
        app: <label_value>
```

3. 使用标准 Trident Protect 工作流程创建备份和还原。

其他详细信息:

- 您可以随时通过更改标签来添加虚拟机或从保护中删除虚拟机:

```
kubectl label virtualmachine <vm_name_to_add> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_to_remove> app
=<different_label_value> -n <vm_namespace>
```

- 您还可以在多个命名空间中使用标签选择器：

```
spec:
  includedNamespaces:
    - namespace: <vm_namespace_1>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
    - namespace: <vm_namespace_2>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
```

使用 includedVirtualMachines 保护特定 VM

如果要保护一个或多个命名空间中的 VirtualMachines 命名集，并从存档中选择性地还原特定虚拟机，请使用 `includedVirtualMachines` 此选项。

应用程序和还原 CR 都使用相同的 `includedVirtualMachines` 选择器格式。

在不同上下文中使用此选择器：

- 在应用程序 CR 中，它定义了受保护的虚拟机。
- 在还原CR中，它定义了从备份或快照存档还原的虚拟机。



在还原 CR 时，`includedVirtualMachines` 是可选的。如果不包括此字段，则会恢复整个存档。指定后，仅还原列出的虚拟机及其从属资源。

下面的 YAML 显示了共享选择器结构：

```
includedVirtualMachines:
  - namespace: <vm_namespace_1>
    names:
      - <vm_name_1>
      - <vm_name_2>
  - namespace: <vm_namespace_2>
    names:
      - <vm_name_3>
```



`includedVirtualMachines` 中的多个条目可以引用不同的命名空间，从而支持多命名空间 VM 感知应用程序定义和还原。

字段	类型	说明
namespace	string	包含目标虚拟机对象的 Kubernetes 命名空间。

字段	类型	说明
名称	字符串列表	该命名空间中的一个或多个虚拟机名称。

基于 **VM** 的应用程序示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace_1>
      names:
        - <vm_name_1>
        - <vm_name_2>
    - namespace: <vm_namespace_2>
      names:
        - <vm_name_3>
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

还原特定 KubeVirt VM

您可以在还原 CR 上使用 `includedVirtualMachines`，以仅还原备份、快照或复制快照存档中的选定 KubeVirt 虚拟机。

在还原 CR 上指定 `includedVirtualMachines` 时，Trident Protect 仅还原列出的 `VirtualMachines` 及其从属资源，例如 `VirtualMachineInstance`、`DataVolume`、`PersistentVolumeClaim`、`PersistentVolume`、`Secret`、`ConfigMap`、`ServiceAccount` 以及其他与 VM 相关的对象。

如果未在还原 CR 上指定 `includedVirtualMachines`，则 Trident Protect 将根据还原 CR 配置还原完整存档。

支持的恢复 **CR** 和 **CLI** 命令

下表列出了支持 `includedVirtualMachines` 的恢复 CR 及其相应的 CLI 命令。

还原 CR	CLI 命令	说明
BackupRestore	<code>tridentctl-protect create backuprestore</code>	从备份还原到新的或不同的命名空间映射。支持 <code>namespaceMapping</code> 和 <code>storageClassMapping</code> 。

还原 CR	CLI 命令	说明
BackupInplaceRestore	tridentctl-protect create backupinplacerestore	从备份还原到原始命名空间或多个命名空间。
SnapshotRestore	tridentctl-protect create snapshotrestore	从快照还原到新的或不同的命名空间映射。支持 namespaceMapping`和`storageClassMapping。
SnapshotInplaceRestore	tridentctl-protect create snapshotinplace restore	从快照还原到原始命名空间或多个命名空间。
ReplicateSnapshotRestore	仅限 YAML，或使用受支持的 CLI 命令（如果在您的环境中可用）	从复制的快照还原。支持 namespaceMapping、`storageClassMapping`以及使用`inPlaceRestore`标志进行就地还原。



在恢复命令中，tridentctl-protect CLI 没有用于`includedVirtualMachines`的标志。要恢复特定的 VM，您必须使用`kubectl apply`直接应用恢复 CR YAML 文件。您可以从头开始编写 YAML，也可以结合使用 CLI 和`--dry-run --output yaml`来生成初始文件，在`spec`下添加`includedVirtualMachines`字段，然后应用该文件。

VM 范围的还原行为

恢复 CR 上的`includedVirtualMachines`字段使用与应用程序 CR 相同的选择器格式：

```
includedVirtualMachines:
- namespace: <source_vm_namespace>
  names:
    - <vm_name_1>
    - <vm_name_2>
```

对于还原 CR，includedVirtualMachines 中的命名空间值是指备份、快照或复制快照存档中存在的源命名空间。

如果使用 namespaceMapping，Trident Protect 首先从源存档中选择请求的虚拟机，然后在还原期间应用命名空间映射。如果使用 storageClassMapping，Trident Protect 会在还原转换阶段应用存储类映射。

使用 CLI 生成还原 YAML

您可以使用 tridentctl-protect`生成基本还原 CR，然后编辑 YAML 以添加`includedVirtualMachines。

```
tridentctl-protect create <restore_command> <restore_name> \
  <restore_options> \
```

```
--dry-run --output yaml > <restore_name>.yaml
```

编辑生成的 YAML 并在 spec 下添加 includedVirtualMachines 字段，然后应用该文件：

```
kubectl apply -f <restore_name>.yaml
```

示例：定义基于 **VM** 的应用程序

以下应用程序保护两个命名空间中的三个特定虚拟机。

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: production-vms
  namespace: prod
spec:
  includedVirtualMachines:
    - namespace: prod
      names:
        - app-server
        - database-server
    - namespace: monitoring
      names:
        - prometheus-vm
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

示例：创建备份

为基于 VM 的应用程序创建备份时，需使用标准备份 CR。备份 CR 上不需要特定于 VM 的字段。

```
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  name: daily-backup
  namespace: prod
spec:
  applicationRef: production-vms
  appVaultRef: s3-vault
```

示例：从备份还原单个虚拟机

以下示例仅从备份存档还原 database-server。存档中的其他虚拟机（例如 app-server 和 prometheus-vm）将被跳过。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: restore-db-only
  namespace: prod-vms-restore
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  namespaceMapping:
    - source: prod
      destination: prod-dr
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f restore-db-only.yaml
```

仅 database-server 及其依赖资源（例如 DataVolumes、PVC、PV、Secrets、ConfigMaps 和相关 KubeVirt 资源）将被还原。

示例：从备份原位还原单个虚拟机

以下示例仅将 database-server 从备份存档还原到其原始命名空间。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: ipr-db-only
  namespace: prod
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f ipr-db-only.yaml
```

示例：从快照还原单个虚拟机

以下示例仅从快照存档还原 ubuntu-vm-blue-a627be38。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: blue-vm-snap-restore
  namespace: blue
spec:
  appVaultRef: my-appvault
  appArchivePath: snapshots/protectctl-blue-vm/snap-1
  includedVirtualMachines:
    - namespace: blue
      names:
        - ubuntu-vm-blue-a627be38
```

```
kubectl apply -f blue-vm-snap-restore.yaml
```

将虚拟机还原与其他筛选器相结合

您可以在同一还原 CR 中将 `includedVirtualMachines` 与 `resourceFilter`、`namespaceMapping` 和 `storageClassMapping` 结合使用，以微调还原的内容。

添加 `resourceFilter` 时，Trident Protect 首先选择 `includedVirtualMachines` 中列出的虚拟机及其依赖资源。然后，它应用资源过滤器来进一步缩小这些结果中包含或排除的内容。

```
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace>
      names:
        - <vm_name>
  resourceFilter:
    resourceSelectionCriteria: Include
    resourceMatchers:
      - kinds: ["PersistentVolumeClaim"]
        names: ["<pvc_name>"]
```

对于 `BackupRestore`、`SnapshotRestore` 和 `ReplicateSnapshotRestore`，`namespaceMapping` 和 `storageClassMapping` 与 `includedVirtualMachines` 协同工作。`includedVirtualMachines` 中的命名空间引用对应于存档中显示的源命名空间。命名空间和存储类映射在过滤完成后的还原转换阶段应用。

有关还原行为详细信息，请参阅 ["使用 Trident Protect 还原应用程序"](#)。

使用 CLI 创建 VM 范围的应用程序

使用 `--virtual-machines` 标志将应用程序的范围限定为特定 KubeVirt 虚拟机，而不是整个命名空间。
`--virtual-machines` 标志与 `--namespaces` 互斥。

步骤

1. 通过运行以下命令之一创建应用程序，将括号中的值替换为您环境中的信息。`--virtual-machines` 标志的格式为 `<namespace>(<vm_name>)`。对于多个虚拟机，请在括号内使用逗号分隔的名称：

- 命名空间中的单个 VM：

```
tridentctl-protect create application <application_name> \  
-n <application_namespace> \  
--virtual-machines "<vm_namespace>(<vm_name>)"
```

- 跨命名空间的多个虚拟机：

```
tridentctl-protect create application <application_name> \  
-n <application_namespace> \  
--virtual-machines "<vm_namespace_1>(<vm_name_1>,<vm_name_2>  
<vm_namespace_2>(<vm_name_3>)"
```

相关信息

- ["定义用于管理的应用程序"](#)
- ["保护应用程序"](#)
- ["使用 Trident Protect 还原应用程序"](#)
- ["使用高级还原设置"](#)
- ["安装 Trident Protect CLI 插件"](#)
- ["使用 AppVault 对象管理存储桶"](#)

管理 Trident Protect 执行挂钩

执行挂钩是一种自定义操作，您可以将其配置为与托管应用的数据保护操作一起运行。例如，如果您有数据库应用程序，则可以使用执行挂钩在快照之前暂停所有数据库事务，并在快照完成后恢复事务。这可确保应用程序一致性快照。

执行挂钩的类型

Trident Protect 支持以下类型的执行挂钩，具体取决于它们可以运行的时间：

- 快照前
- 快照后
- 备份前
- 备份后
- 恢复后
- 故障转移后

执行顺序

运行数据保护操作时，执行挂钩事件按以下顺序发生：

1. 任何适用的自定义操作前执行挂钩都在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义操作前钩子，但操作之前这些钩子的执行顺序既不保证也不可配置。
2. 如果适用，文件系统会冻结。["详细了解如何使用 Trident Protect 配置文件系统冻结"](#)。
3. 执行数据保护操作。
4. 已冻结的文件系统已解冻（如适用）。
5. 任何适用的自定义操作后执行挂钩都在适当的容器上运行。您可以根据需要创建和运行尽可能多的自定义操作后挂钩，但操作后这些挂钩的执行顺序既不保证也不可配置。

如果创建相同类型的多个执行挂钩（例如，pre-snapshot），则无法保证这些挂钩的执行顺序。但是，不同类型的钩子的执行顺序是有保证的。例如，以下是具有所有不同类型钩子的配置的执行顺序：

1. 已执行快照前挂钩
2. 已执行快照后挂钩
3. 已执行备份前挂钩
4. 已执行备份后挂钩



前面的顺序示例仅适用于运行不使用现有 snapshot 的备份时。



在生产环境中启用执行挂钩脚本之前，应始终对其进行测试。您可以使用 'kubectl exec' 命令来方便地测试脚本。在生产环境中启用执行挂钩后，测试生成的快照和备份以确保它们是一致的。为此，可以将应用克隆到临时命名空间，还原快照或备份，然后测试应用。



如果快照前执行挂钩添加、更改或删除 Kubernetes 资源，这些更改将包含在快照或备份以及任何后续还原操作中。

关于自定义执行挂钩的重要说明

为应用规划执行挂钩时，请考虑以下内容。

- 执行挂钩必须使用脚本来执行操作。许多执行挂钩可以引用相同的脚本。
- Trident Protect 要求执行挂钩使用的脚本以可执行 shell 脚本的格式编写。
- 脚本大小限制为 96KB。

- Trident Protect 使用执行挂钩设置和任何匹配标准来确定哪些挂钩适用于快照、备份或还原操作。



由于执行挂钩通常会减少或完全禁用它们所运行的应用程序的功能，因此应始终尝试最大限度地减少自定义执行挂钩的运行时间。如果使用关联的执行挂钩启动备份或快照操作，但随后取消该操作，则如果备份或快照操作已经开始，则仍允许运行挂钩。这意味着备份后执行挂钩中使用的逻辑不能假定备份已完成。

执行挂钩筛选器

在为应用程序添加或编辑执行挂钩时，可以向执行挂钩添加筛选器，以管理挂钩将匹配的容器。筛选器对于在所有容器上使用相同容器映像的应用程序非常有用，但可能会将每个映像用于不同的目的（例如 Elasticsearch）。筛选器允许您创建执行挂钩在某些但不一定所有相同的容器上运行的场景。如果为单个执行挂钩创建多个筛选器，它们将与逻辑 AND 运算符组合。每个执行挂钩最多可以有 10 个活动筛选器。

添加到执行挂钩的每个筛选器都使用正则表达式来匹配集群中的容器。当挂钩与容器匹配时，挂钩将在该容器上运行其关联的脚本。筛选器的正则表达式使用正则表达式 2 (RE2) 语法，该语法不支持创建从匹配列表中排除容器的筛选器。有关 Trident Protect 在执行挂钩筛选器中支持正则表达式的语法的信息，请参阅 ["正则表达式 2 \(RE2\) 语法支持"](#)。



如果将命名空间筛选器添加到在还原或克隆操作后运行的执行挂钩中，并且还原或克隆源和目标在不同命名空间中，则命名空间筛选器仅应用于目标命名空间。

执行挂钩示例

访问 ["NetApp Verda GitHub 项目"](#) 以下载流行应用程序（如 Apache Cassandra 和 Elasticsearch）的实际执行挂钩。您还可以查看示例并获得构建自己的自定义执行挂钩的想法。



Verda 存储库中的脚本按原样提供，不受 NetApp 的正式支持。NetApp 支持仅涵盖 Trident Protect 中的执行挂钩框架。

创建执行挂钩

您可以使用 Trident Protect 为应用程序创建自定义执行挂钩。您需要拥有所有者、管理员或成员权限才能创建执行挂钩。

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-hook.yaml`。
2. 配置以下属性以匹配您的 Trident Protect 环境和集群配置：
 - **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.applicationRef:** (*Required*) 要为其运行执行挂钩的应用程序的 Kubernetes 名称。
 - **spec.stage:** (*Required*) 指示执行挂钩应在操作期间运行哪个阶段的字符串。可能的值：
 - 之前
 - 发布

- **spec.action:** (*Required*) 指示执行挂钩将采取哪些操作的字符串，假设指定的任何执行挂钩过滤器都匹配。可能的值：
 - Snapshot
 - 备份
 - 还原
 - 故障转移
- **spec.enabled:** (*Optional*) 指示是启用还是禁用此执行挂钩。如果未指定，则默认值为 true。
- **spec.hookSource:** (*Required*) 包含 base64 编码的钩子脚本的字符串。
- **spec.timeout:** (*Optional*) 一个数字，定义允许执行挂钩运行的时间，以分钟为单位。最小值为 1 分钟，如果未指定，默认值为 25 分钟。
- **spec.arguments:** (*Optional*) 可以为执行挂钩指定的参数的 YAML 列表。
- **spec.matchingCriteria:** (可选) 条件键值对的可选列表，每对组成一个执行挂钩过滤器。每个执行挂钩最多可添加 10 个筛选器。
- **spec.matchingCriteria.type:** (*Optional*) 标识执行挂钩筛选器类型的字符串。可能的值：
 - ContainerImage
 - ContainerName
 - PodName
 - PodLabel
 - NamespaceName
- **spec.matchingCriteria.value:** (*Optional*) 标识执行挂钩过滤器值的字符串或正则表达式。

示例 YAML:

```
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. 使用正确的值填充 CR 文件后，应用 CR：

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

使用 CLI

步骤

1. 创建执行挂钩，用环境中的信息替换括号中的值。例如：

```
tridentctl-protect create exehook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

手动运行执行挂钩

您可以出于测试目的手动运行执行挂钩，也可以在出现故障后手动重新运行挂钩。您需要拥有所有者、管理员或成员权限才能手动运行执行挂钩。

手动运行执行挂钩包括两个基本步骤：

1. 创建资源备份，收集资源并创建资源的备份，确定挂钩将在何处运行
2. 针对备份运行执行挂钩

▼ 步骤 1：创建资源备份

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-resource-backup.yaml`。

2. 配置以下属性以匹配您的 Trident Protect 环境和集群配置：

- **metadata.name:** (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
- **spec.applicationRef:** (必需) 要为其创建资源备份的应用程序的 Kubernetes 名称。
- **spec.appVaultRef:** (必需) 存储备份内容的 AppVault 的名称。
- **spec.appArchivePath:** AppVault 中存储备份内容的路径。您可以使用以下命令查找此路径：

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

示例 YAML：

```
---  
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. 使用正确的值填充 CR 文件后，应用 CR：

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

使用 CLI

步骤

1. 创建备份，用环境中的信息替换括号中的值。例如：

```
tridentctl protect create resourcebackup <my_backup_name> --app  
<my_app_name> --appvault <my_appvault_name> -n <my_app_namespace>  
--app-archive-path <app_archive_path>
```

2. 查看备份的状态。您可以重复使用此示例命令，直到操作完成：

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

3. 验证备份是否成功：

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

▼ 步骤 2：运行执行挂钩

使用 CR

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-hook-run.yaml`。
2. 配置以下属性以匹配您的 Trident Protect 环境和集群配置：
 - **metadata.name**： (*Required*) 此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
 - **spec.applicationRef**： (*Required*) 确保此值与您在步骤 1 中创建的 ResourceBackup CR 中的应用程序名称相匹配。
 - **spec.appVaultRef**： (必填) 确保此值与您在第 1 步创建的 ResourceBackup CR 中的 appVaultRef 相匹配。
 - **spec.appArchivePath**： 确保该值与您在第 1 步创建的 ResourceBackup CR 中的 appArchivePath 相匹配。

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

- **spec.action**： (*Required*) 指示执行挂钩将采取哪些操作的字符串，假设指定的任何执行挂钩过滤器都匹配。可能的值：
 - Snapshot
 - 备份
 - 还原
 - 故障转移
- **spec.stage**： (*Required*) 指示执行挂钩应在操作期间运行哪个阶段的字符串。此挂钩运行不会在任何其他阶段运行挂钩。可能的值：
 - 之前
 - 发布

示例 YAML：

```
---  
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. 使用正确的值填充 CR 文件后，应用 CR：

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

使用 CLI

步骤

1. 创建手动执行挂钩运行请求：

```
tridentctl protect create exehooksruntime <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. 检查执行挂钩运行的状态。您可以重复运行此命令，直到操作完成：

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

3. 描述 exehooksruntime 对象以查看最终详细信息和状态：

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

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 <http://www.netapp.com/TM> 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。