使用Trident Protect 保护应用程序

Trident

NetApp
February 05, 2026

目录

使用Trident Protect 保护应用程序

了解Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强了由NetApp ONTAP存储系统和NetApp Trident CSI 存储供应器支持的有状态 Kubernetes 应用程序的功能和可用性。Trident Protect 简化了跨公有云和本地环境的容器化工作负载的管理、保护和迁移。它还通过其 API 和 CLI 提供自动化功能。

您可以通过创建自定义资源 (CR) 或使用Trident Protect CLI 来使用Trident Protect 保护应用程序。

下一步是什么?

您可以先了解Trident Protect 的相关要求，然后再进行安装：

- "Trident保护要求"

安装Trident Protect

Trident保护要求

首先，请验证您的运行环境、应用程序集群、应用程序和许可证是否准备就绪。确保您的环境满足部署和运行Trident Protect 的这些要求。

Trident Protect Kubernetes 集群兼容性

Trident Protect 与各种完全托管和自托管的 Kubernetes 产品兼容，包括：

- Amazon Elelic Kubelnetes Service (EKS)
- Google Kubernetes Engine （ GKEE ）
- Microsoft Azure Kubernetes Service （ AKS ）
- Red Hat OpenShift
- SUSE试用者
- VMware Tanzu产品组合
- 上游Kubernetes

确保安装Trident Protect 的集群已配置运行中的快照控制器和相关的 CRD。要安装快照控制器，请参阅 "这些说明"。

Trident Protect 存储后端兼容性

Trident Protect 支持以下存储后端：

- 适用于 NetApp ONTAP 的 Amazon FSX

- Cloud Volumes ONTAP
- ONTAP存储阵列
- Google Cloud NetApp卷
- Azure NetApp Files

确保存储后端满足以下要求：

- 确保连接到集群的NetApp存储使用的是Astra Trident 24.02或更高版本(建议使用Trident 24.10)。
  - 如果Astra Trident的版本低于24.06.1、而您计划使用NetApp SnapMirror灾难恢复功能、则需要手动启用Astra Control配置程序。
- 确保已安装最新的Astra控件配置程序(从Astra Trident 24.06.1开始、默认情况下已安装并启用)。
- 确保您有一个NetApp ONTAP存储后端。
- 确保已配置用于存储备份的对象存储分段。
- 创建您计划用于应用程序或应用程序数据管理操作的任何应用程序命名空间。Trident Protect 不会为您创建这些命名空间；如果您在自定义资源中指定了不存在的命名空间，则操作将失败。

NAS经济型卷的要求

Trident Protect 支持对 nas-economy 卷进行备份和恢复操作。目前不支持将快照、克隆和SnapMirror复制到 nas-economy 卷。您需要为计划与Trident Protect 一起使用的每个 nas-economy 卷启用快照目录。

某些应用程序与使用Snapshot目录的卷不兼容。对于这些应用程序、您需要通过在ONTAP存储系统上运行以下命令来隐藏Snapshot目录：

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

您可以通过对每个NAS经济型卷运行以下命令来启用Snapshot目录、并将其替换 `<volume-UUID>`为要更改的卷的UUID：

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

通过将Trident后端配置选项设置为，您可以默认为 `true`新卷启用快照目录 `snapshotDir`。现有卷不受影响。

使用**KubeVirt VM**保护数据

Trident Protect 24.10 和 24.10.1 及更高版本在保护运行在 KubeVirt VM 上的应用程序时，行为有所不同。对于这两个版本，您都可以在数据保护操作期间启用或禁用文件系统冻结和解冻结。

对于所有Trident Protect 版本，要在 OpenShift 环境中启用或禁用自动冻结功能，您可能需要授予应用程序命名空间特权权限。例如：

```
oc adm policy add-scc-to-user privileged -z default -n
<application-namespace>
```

Trident Protect 24.10

Trident Protect 24.10 在数据保护操作期间不会自动确保 KubeVirt VM 文件系统的一致性状态。如果您想使用Trident Protect 24.10 保护您的 KubeVirt VM 数据，则需要在执行数据保护操作之前手动启用文件系统的冻结/解冻功能。这样可以确保文件系统处于一致状态。

您可以配置Trident Protect 24.10 来管理数据保护操作期间 VM 文件系统的冻结和解冻。"正在配置虚拟化"然后使用以下命令：

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

Trident Protect 24.10.1 及更高版本

从 Trident Protect 24.10.1 开始，Trident Protect 在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。或者，您可以使用以下命令禁用此自动行为：

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

SnapMirror复制的要求

NetApp SnapMirror可与Trident Protect 配合使用，适用于以下ONTAP解决方案：

- NetApp ASA
- NetApp AFF
- NetApp FAS
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- 适用于 NetApp ONTAP 的 Amazon FSX

SnapMirror复制的ONTAP集群要求

如果您计划使用SnapMirror复制、请确保ONTAP集群满足以下要求：

- * Astra Control Provisioner 或Trident *：使用ONTAP作为后端的源 Kubernetes 集群和目标 Kubernetes 集群上必须同时存在Astra Control Provisioner 或Trident 。Trident Protect 支持使用NetApp SnapMirror技术进行复制，该技术使用以下驱动程序支持的存储类：
  - `ontap-nas`

- ontap-san

- 许可证：必须在源和目标ONTAP集群上启用使用数据保护包的ONTAP SnapMirror异步许可证。有关详细信息、请参见 "ONTAP 中的SnapMirror许可概述" 。

SnapMirror复制的对等注意事项

如果您计划使用存储后端对等、请确保您的环境满足以下要求：

- 集群和**SVM**：ONTAP存储后端必须建立对等状态。有关详细信息、请参见 "集群和 SVM 对等概述" 。

  确保两个ONTAP集群之间的复制关系中使用的SVM名称是唯一的。

- **Astra**控件配置程序或**Trident**和**SVM**：对等远程SVM必须可供目标集群上的Astra控件配置程序或Trident使用。
- 托管后端：您需要在Trident Protect 中添加和管理ONTAP存储后端，以创建复制关系。
- **NVMe over TCP**： Trident Protect 不支持使用 NVMe over TCP 协议的存储后端的NetApp SnapMirror复制。

用于SnapMirror复制的Trident / ONTAP配置

Trident Protect 要求您至少配置一个支持源集群和目标集群复制的存储后端。如果源集群和目标集群相同，为了获得最佳弹性，目标应用程序应该使用与源应用程序不同的存储后端。

安装并配置Trident Protect

如果您的环境满足Trident Protect 的要求，您可以按照以下步骤在集群上安装Trident Protect。您可以从NetApp获取Trident Protect，或者从您自己的私有注册表中安装它。如果您的集群无法访问互联网，从私有注册表安装会很有帮助。

默认情况下， Trident Protect 会收集有助于处理您可能提交的任何NetApp支持案例的支持信息，包括集群和受管应用程序的日志、指标和拓扑信息。Trident Protect 会按计划每日向NetApp发送这些支持包。安装Trident Protect 时，您可以选择禁用此支持包集合。您可以手动操作"生成支持包"随时。

安装Trident Protect

从NetApp安装Trident Protect

步骤

1. 添加Trident Helm存储库：

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

2. 安装Trident Protect CRD：

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

3. 使用 Helm 通过以下命令之一安装Trident Protect。代替 `<name_of_cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

   - 正常安装Trident Protect：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.1
--create-namespace --namespace trident-protect
```

   - 安装Trident Protect 并禁用每日定期上传的Trident Protect AutoSupport支持包：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect
```

从私有注册表安装Trident Protect

如果您的 Kubernetes 集群无法访问互联网，您可以从私有镜像仓库安装Trident Protect。在这些示例中，请将括号中的值替换为您环境中的信息：

步骤

1. 将以下映像提取到本地计算机、更新标记、然后将其推送到您的私人注册表：

```
netapp/controller:24.10.1
netapp/restic:24.10.1
netapp/kopia:24.10.1
netapp/trident-autosupport:24.10.0
netapp/exechook:24.10.1
netapp/resourcebackup:24.10.1
netapp/resourcerestore:24.10.1
netapp/resourcedelete:24.10.1
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例如：

```
docker pull netapp/controller:24.10.1
```

```
docker tag netapp/controller:24.10.1 <private-registry-
url>/controller:24.10.1
```

```
docker push <private-registry-url>/controller:24.10.1
```

2. 创建Trident Protect 系统命名空间：

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

3. 登录到注册表：

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

4. 创建用于私人注册表身份验证的拉机密：

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

5. 添加Trident Helm存储库：

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

6. 创建一个名为的文件 `protectValues.yaml`。请确保其中包含以下Trident Protect 设置：

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

7. 安装Trident Protect CRD：

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1 --create-namespace --namespace
trident-protect
```

8. 使用 Helm 通过以下命令之一安装Trident Protect。代替 `<name_of_cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

   - 正常安装Trident Protect：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.1
--create-namespace --namespace trident-protect -f
protectValues.yaml
```

   - 安装Trident Protect 并禁用每日定期上传的Trident Protect AutoSupport支持包：

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.1 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

指定Trident Protect 容器资源限制

安装Trident Protect 后，您可以使用配置文件来指定Trident Protect 容器的资源限制。设置资源限制可以控制Trident Protect 操作消耗集群资源的程度。

步骤

1. 创建一个名为的文件 `resourceLimits.yaml`。
2. 根据您的环境需求，在文件中填充Trident Protect 容器的资源限制选项。

   以下示例配置文件显示了可用设置、并包含每个资源限制的默认vaule：

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
```

```
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. 应用文件中的值 `resourceLimits.yaml`：

```
helm upgrade trident-protect -n trident-protect -f <resourceLimits.yaml>
--reuse-values
```

安装Trident Protect CLI 插件

您可以使用Trident Protect 命令行插件，它是Trident的一个扩展。 `tridentctl`用于创建和与Trident Protect 自定义资源 (CR) 交互的实用程序。

安装Trident Protect CLI 插件

在使用命令行实用程序之前、您需要将其安装在用于访问集群的计算机上。根据您的计算机使用的是x64 CPU还是ARM CPU、执行以下步骤。

下载适用于**Linux amd64 CPU**的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-linux-amd64
```

下载适用于**Linux ARM64 CPU**的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-linux-arm64
```

下载适用于**Mac amd64 CPU**的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-macos-amd64
```

下载适用于**Mac ARM64 CPU**的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.1/tridentctl-protect-macos-arm64
```

1. 为插件二进制文件启用执行权限：

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

2. 将插件二进制文件复制到路径变量中定义的位置。例如、 `/usr/bin`或 `/usr/local/bin`(您可能需要提升Privileges)：

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

3. 您也可以将插件二进制文件复制到主目录中的某个位置。在这种情况下、建议确保此位置属于您的路径变量：

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

通过将插件复制到路径变量中的某个位置、您可以通过在任意位置键入或 `tridentctl protect`来使用此插件 `tridentctl-protect。

查看**Trident**命令行界面插件帮助

您可以使用内置插件的帮助功能获取有关插件功能的详细帮助：

步骤

1. 使用帮助功能查看使用指南：

```
tridentctl-protect help
```

启用命令自动完成

安装Trident Protect CLI 插件后，您可以为某些命令启用自动补全功能。

为**bash shell**启用自动完成

步骤

1. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/tridentctl-completion.bash
```

2. 在主目录中创建一个新目录以包含该脚本：

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

3. 将下载的脚本移动到 `~/.bash/completions`目录：

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

4. 将以下行添加到 `~/.bashrc`主目录中的文件：

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

为**Z shell**启用自动完成

步骤

1. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.1/tridentctl-completion.zsh
```

2. 在主目录中创建一个新目录以包含该脚本：

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

3. 将下载的脚本移动到 `~/.zsh/completions`目录：

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

4. 将以下行添加到 `~/.zprofile`主目录中的文件：

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

结果

下次shell登录时、您可以使用带有trdentcd-protect插件的命令自动完成。

管理Trident Protect

管理Trident Protect 授权和访问控制

Trident Protect 使用 Kubernetes 的基于角色的访问控制 (RBAC) 模型。默认情况下，Trident Protect 提供一个系统命名空间及其关联的默认服务帐户。如果您的组织拥有众多用户或特定的安全需求，则可以使用Trident Protect 的 RBAC 功能来更精细地控制对资源和命名空间的访问。

集群管理员始终可以访问默认命名空间中的资源 `trident-protect`、也可以访问所有其他命名空间中的资源。要控制对资源和应用程序的访问、您需要创建更多的名分并将资源和应用程序添加到这些名分。

请注意、任何用户都不能在默认命名空间中创建应用程序数据管理CRS `trident-protect`。您需要在应用程序命名空间中创建应用程序数据管理CRS (最佳做法是、在与其关联的应用程序相同的命名空间中创建应用程序数据管理CRS)。

只有管理员才能访问具有特权的Trident Protect 自定义资源对象，其中包括：

- **AppVault**：需要存储分段凭据数据
- **AutoSupportBundle**：收集指标、日志和其他敏感的Trident Protect数据
- *AutoSupportBundleSchedule：管理日志收集计划

作为最佳实践、请使用RBAC限制管理员对有权限的对象的访问。

有关RBAC如何控制对资源和称表的访问的详细信息，请参阅 "Kubbernetes RBAC文档"。

有关服务帐户的信息，请参见 "Kubbernetes服务帐户文档"。

示例：管理两组用户的访问权限

例如、一个组织有一个集群管理员、一组工程用户和一组营销用户。集群管理员应完成以下任务、以创建一个环境、在此环境中、工程组和营销组各自只能访问分配给各自命名区域的资源。

第1步：创建一个命名空间以包含每个组的资源

通过创建命名空间、您可以从逻辑上分离资源、并更好地控制谁有权访问这些资源。

步骤

1. 为工程组创建命名空间：

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

2. 为营销组创建命名空间：

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

第2步：创建新的服务帐户、以便与每个命名空间中的资源进行交互

您创建的每个新命名空间都会附带一个默认服务帐户、但您应为每组用户创建一个服务帐户、以便将来根据需要在各个组之间进一步划分Privileges。

步骤

1. 为工程组创建服务帐户：

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

2. 为营销组创建服务帐户：

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

第3步：为每个新服务帐户创建一个密钥

服务帐户密钥用于向服务帐户进行身份验证、如果泄露、可以轻松删除和重新创建。

步骤

1. 为工程服务帐户创建一个密钥：

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. 为营销服务帐户创建密钥：

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

第4步：创建RoleBinding对象以将ClusterRole对象绑定到每个新服务帐户

安装Trident Protect 时会创建一个默认的 ClusterRole 对象。您可以通过创建和应用 RoleBinding 对象将此 ClusterRole 绑定到服务帐户。

步骤

1. 将ClusterRole绑定到工程服务帐户：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. 将ClusterRole绑定到营销服务帐户：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

第5步：测试权限

测试权限是否正确。

步骤

1. 确认工程用户可以访问工程资源：

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

2. 确认工程用户无法访问营销资源：

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

第6步：授予对AppVault对象的访问权限

要执行备份和快照等数据管理任务、集群管理员需要向各个用户授予对AppVault对象的访问权限。

步骤

1. 创建并应用AppVault和机密组合YAML文件、以授予用户对AppVault的访问权限。例如、以下CR将授予用户对AppVault的访问权限 `eng-user`：

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 创建并应用角色CR、使集群管理员能够授予对命名空间中特定资源的访问权限。例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. 创建并应用RoleBinding CR以将权限绑定到用户eng-user。例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 验证权限是否正确。

   a. 尝试检索所有名称库的AppVault对象信息：

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

   您应看到类似于以下内容的输出：

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. 测试用户是否可以获取他们现在有权访问的AppVault信息：

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

您应看到类似于以下内容的输出：

```
yes
```

结果

您为其授予了AppVault权限的用户应该能够使用授权的AppVault对象执行应用程序数据管理操作、并且不能访问分配的命名区之外的任何资源、也不能创建他们无权访问的新资源。

生成**Trident Protect** 支持包

Trident Protect 使管理员能够生成包含对NetApp支持有用的信息的捆绑包，包括有关受管理集群和应用程序的日志、指标和拓扑信息。如果您已连接到互联网，则可以使用自定义资源 (CR) 文件将支持包上传到NetApp支持站点 (NSS)。

使用**CR**创建支持包

步骤

1. 创建自定义资源(CR)文件并将其命名(例如 `trident-protect-support-bundle.yaml`)。
2. 配置以下属性：
   - 。**metadata.name**:(*required_*)此自定义资源的名称；请为您的环境选择一个唯一且合理的名称。
   - **spec.t触发 器Type**：(*required*)用于确定是立即生成支持包、还是按计划生成支持包。计划在UTC时间中午12点生成捆绑包。可能值：
     - 已计划
     - 手动
   - **spec.u倍 加载已启用**：(可 选)控制是否应在生成支持包后将其上传到NetApp支持站点。如果未指定，则默认为 `false`。可能值：
     - true
     - false (默认)
   - **spec.dataWindowStart**：(可 选) RFC 3339格式的日期字符串，指定支持包中包含的数据窗口应开始的日期和时间。如果未指定、则默认为24小时前。您可以指定的最早窗口日期是7天前。

     YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. 使用正确的值填充文件后 `astra-support-bundle.yaml` 、应用CR：

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

使用命令行界面创建支持包

步骤

1. 创建支持包、将括号中的值替换为您环境中的信息。 `trigger-type`确定是否立即创建分发包，或者是否由计划决定了创建时间，可以是 `Manual`或 `Scheduled。默认设置为 `Manual。

   例如：

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

升级Trident保护

您可以将Trident Protect 升级到最新版本，以享受新功能或修复错误。

要升级Trident Protect，请执行以下步骤。

步骤

1. 更新Trident Helm存储库：

```
helm repo update
```

2. 升级Trident Protect CRD：

```
helm upgrade trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.1  --namespace trident-protect
```

3. 升级Trident保护：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2410.1 --namespace trident-protect
```

管理和保护应用程序

使用**Trident Protect AppVault** 对象来管理存储桶。

Trident Protect 的存储桶自定义资源 (CR) 被称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含存储桶在保护操作（例如备份、快照、恢复操作和SnapMirror复制）中使用的必要配置。只有管理员才能创建应用保险库。

密钥生成和**AppVault**定义示例

定义AppVault CR时、您需要包含凭据才能访问由提供程序托管的资源。根据提供程序的不同、为凭据生成密钥的方式也会有所不同。以下是多个提供程序的命令行密钥生成示例、然后是每个提供程序的AppVault定义示例。

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

AppVault CR示例

您可以使用以下CR示例为每个云提供程序创建AppVault对象。

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  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-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  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-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

StorageGRID S3

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-
971f-ac4a83621922
  namespace: trident-protect
spec:
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

使用**Trident Protect CLI** 创建 **AppVault** 的示例

您可以使用以下命令行界面命令示例为每个提供程序创建AppVault CRS。

Google Cloud

```
tridentctl-protect create vault GCP my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3 (AWS)

```
tridentctl-protect create vault AWS <vault-name> --bucket <bucket-name>
--secret  <secret-name>  --endpoint <s3-endpoint>
```

Microsoft Azure

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

通用 S3

```
tridentctl-protect create vault GenericS3 <vault-name> --bucket
<bucket-name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

StorageGRID S3

```
tridentctl-protect create vault StorageGridS3 s3vault --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

使用AppVault浏览器查看AppVault信息

您可以使用Trident Protect CLI 插件查看有关集群上创建的 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
   ```

   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` 。

删除**AppVault**

您可以随时删除AppVault对象。

> 在删除AppVault对象之前、请勿 `finalizers`删除AppVault CR中的密钥。如果这样做、可能会导致AppVault存储分段中有残留数据、集群中会出现孤立资源。

开始之前

确保已删除存储在关联存储分段中的所有快照和备份。

使用**Kubbernetes**命令行界面删除**AppVault**

1. 删除AppVault对象、替换 `appvault_name`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、替换 `appvault_name`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 和关联的 AppVault CR 来定义要使用Trident Protect 管理的应用程序。

创建**AppVault CR**

您需要创建一个 AppVault CR，该 CR 将在对应用程序执行数据保护操作时使用，并且 AppVault CR 需要位于安装了Trident Protect 的集群上。AppVault CR 是针对您的特定环境的；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 来定义要管理的应用程序。

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**:(*required_*)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_*)使用命名空间标签或命名空间名称来指定应用程序资源所在的命名空间。应用程序命名空间必须属于此列表。

        YAML示例：

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

2. 创建应用程序CR以匹配您的环境后、请应用CR。例如：

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

使用命令行界面添加应用程序

步骤

1. 创建并应用应用应用程序定义、将括号中的值替换为您环境中的信息。您可以使用逗号分隔列表和以下示例中所示的参数在应用程序定义中包括名称和资源：

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

使用**Trident Protect** 保护应用程序

您可以使用自动保护策略或临时保护策略，通过拍摄快照和备份来保护Trident Protect 管理的所有应用程序。

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。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

创建按需快照

您可以随时创建按需快照。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：要创建快照的应用程序的Kubernetes名称。
   - **spec.appVaultRef**：(*required*)应存储快照内容(元数据)的AppVault的名称。
   - **spec.relaimPolicy**：(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，快照也 `Retain`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. 使用正确的值填充文件后 `trident-protect-snapshot-cr.yaml` 、应用CR：

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

使用命令行界面创建快照

步骤

1. 创建快照、将括号中的值替换为您环境中的信息。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。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要备份的应用程序的Kubernetes名称。
   - **spec.appVaultRef**：(*required*)应存储备份内容的AppVault的名称。
   - **spec.dataMover**：(可 选)一个字符串，指示用于备份操作的备份工具。可能值(区分大小写)：
     - `Restic`
     - `Kopia` (默认)
   - **spic.relaimPolicy**：(可 选)定义了从备份申请中释放备份时会发生什么情况。可能值：
     - `Delete`
     - `Retain` (默认)
   - **Spec.snapshotRef**：(可 选)：要用作备份源的快照的名称。如果不提供此参数、则会创建和备份临时快照。

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

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

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

使用命令行界面创建备份

步骤

1. 创建备份、将括号中的值替换为您环境中的信息。例如：

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

创建数据保护计划

保护策略通过按定义的计划创建快照，备份或这两者来保护应用程序。您可以选择每小时，每天，每周和每月创建快照和备份，并且可以指定要保留的副本数。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.dataMover**：(可 选)一个字符串，指示用于备份操作的备份工具。可能值(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**：(*required*)应存储备份内容的AppVault的名称。
   - *spec.backupretention *：要保留的备份数。零表示不应创建任何备份。
   - *spec.snapshotretention *：要保留的快照数。零表示不应创建任何快照。
   - 。**spec.granularity**:计划的运行频率。可能值以及必需的关联字段：
     - `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`)
   - **spec.dayOfMonth**：(可 选)计划应运行的日期(1 - 31)。如果粒度设置为，则需要此字段 `monthly`。
   - **spec.dayOfWeek**：(可 选)计划应运行的日期(0到7)。值0或7表示星期日。如果粒度设置为，则需要此字段 `weekly`。
   - *spec.hour *：(可 选)计划应运行的时间(0 - 23)。如果粒度设置为、或，则需要此字段 `daily` `weekly` `monthly`。
   - *spec.minute：(可 选)计划应运行的分钟(0 - 59)。如果粒度设置为、、或，则需要此字段 `hourly` `daily` `weekly` `monthly`。

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"
```

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

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

使用命令行界面创建计划

步骤

1. 创建保护计划、将括号中的值替换为您环境中的信息。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> --data-mover
<kopia_or_restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain> -n <application_namespace>
```

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。

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、已完成或失败的备份操作的状态。

步骤

1. 使用以下命令检索备份操作的状态、将括号中的值替换为环境中的信息：

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

为**azure-ANF-files (NetApp)**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 存储类且在Trident 24.06 之前创建的存储后端启用节省空间的备份和还原功能。此功能适用于 NFSv4 卷，并且不会占用容量池中的额外空间。

开始之前

确保满足以下要求：

- 您已安装Trident Protect。
- 您已在Trident Protect中定义了一个应用程序。在您完成此步骤之前，此应用程序的保护功能将受到限制。
- 您已 `azure-netapp-files` 选择作为存储后端的默认存储类。

展开以了解配置步骤

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-pv-files且与应用程序关联的NetApp启用Snapshot目录：

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

   b. 确认已为每个关联PV启用Snapshot目录：

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

   响应：

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未启用快照目录， Trident Protect 将选择常规备份功能，该功能会在备份过程中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来创建与被备份卷大小相同的临时卷。

结果

该应用程序已准备好使用Trident Protect 进行备份和恢复。每个 PVC 也可供其他应用程序用于备份和恢复。

使用Trident Protect 恢复应用程序

您可以使用Trident Protect 从快照或备份中恢复您的应用程序。将应用程序恢复到同一集群时，从现有快照恢复速度会更快。

> 还原应用程序时、为该应用程序配置的所有执行挂钩都会随该应用程序还原。如果存在还原后执行挂钩、则它会在还原操作中自动运行。

还原和故障转移操作期间的命名空间标注和标签

在还原和故障转移操作期间、目标命名空间中的标签和标注会与源命名空间中的标签和标注相匹配。此时将添加源命名空间中目标命名空间中不存在的标签或标注、并覆盖已存在的任何标签或标注、以便与源命名空间中的值匹配。仅存在于目标命名空间上的标签或标注保持不变。

> 如果您使用RedHat OpenShift、请务必注意命名空间标注在OpenShift环境中的关键作用。命名空间标注可确保还原的Pod遵循OpenShift安全上下文约束(SCC)定义的适当权限和安全配置、并可在没有权限问题的情况下访问卷。有关详细信息，请参阅 "OpenShift安全上下文约束文档"。

在执行还原或故障转移操作之前、您可以通过设置Kubornetes环境变量来防止目标命名空间中的特定标注被覆

盖 RESTORE_SKIP_NAMESPACE_ANNOTATIONS。例如：

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例显示了一个源和目标命名空间、每个命名空间都具有不同的标注和标签。您可以查看目标命名空间在操作前后的状态、以及标注和标签在目标命名空间中的组合或覆盖方式。

在执行还原或故障转移操作之前

下表说明了执行还原或故障转移操作之前示例源和目标名称卷的状态：

命名空间	标注	标签
命名空间ns-1 (源)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE"	• 环境=生产 • 合规性=HIPAA • name=nS-1
命名空间ns-2 (目标)	• 标注.One/键："TRUE" • 标注三个/项："false"	• Role=database

还原操作之后

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。已添加某些密钥、某些密钥已被覆盖、并且 `name`标签已更新以与目标命名空间匹配：

命名空间	标注	标签
命名空间ns-2 (目标)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE" • 标注三个/项："false"	• name=nS-2 • 合规性=HIPAA • 环境=生产 • Role=database

从备份还原到其他命名空间

当您使用 BackupRestore CR 将备份还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用

程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或者制定保护计划。

> 将备份还原到具有现有资源的其他命名空间不会更改与备份中的资源共享名称的任何资源。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、或者将备份还原到新命名空间。

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；请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**：(*required*)存储备份内容的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。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。
   - **spec.storageClassMapping**：还原操作的源存储类到目标存储类的映射。将和 `sourceStorageClass`替换 `destinationStorageClass`为您环境中的信息。

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

3. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。

- **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
- **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
- *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。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. 使用正确的值填充文件后 `trident-protect-backup-restore-cr.yaml` 、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、将括号中的值替换为环境中的信息。此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的 `source1:dest1,source2:dest2`卷。例如：

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

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。

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；请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**：(*required*)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. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
       - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
       - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
       - *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。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. 使用正确的值填充文件后 `trident-protect-backup-ipr-cr.yaml` 、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、将括号中的值替换为环境中的信息。 `backup`参数使用格式为的命名空间和备份名称 `<namespace>/<name>`。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

如果原始集群出现问题、您可以将备份还原到其他集群。

- 开始之前 *

确保满足以下前提条件：

- 目标集群已安装Trident Protect。
- 目标集群可以访问与存储备份的源集群相同的AppVault的分段路径。

步骤

1. 使用Trident Protect CLI 插件检查目标集群上 AppVault CR 的可用性：

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

确保目标集群上存在用于应用程序还原的命名空间。

2. 从目标集群查看可用AppVault的备份内容：

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

运行此命令可显示AppVault中的可用备份、包括其原始集群、相应的应用程序名称、时间戳和归档路径。

示例输出：

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

3. 使用AppVault名称和归档路径将应用程序还原到目标集群：

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：(*required*)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}'
```

   如果BackupRestore CR不可用、您可以使用步骤2中提到的命令查看备份内容。

   - 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

     例如：

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

3. 使用正确的值填充文件后 `trident-protect-backup-restore-cr.yaml` 、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、将括号中的值替换为环境中的信息。命名空间映射参数使用冒号分隔的命名空间将源命名空间映射到正确的目标命名空间、格式为SOURCE1：dest1、Source2：dest2。例如：

```
tridentctl-protect create backuprestore <restore_name> --namespace
-mapping <source_to_destination_namespace_mapping> --appvault
<appvault_name> --path <backup_path> -n <application_namespace>
--context <destination_cluster_name>
```

从快照还原到其他命名空间

您可以使用自定义资源 (CR) 文件从快照恢复数据，恢复到不同的命名空间或原始源命名空间。当您使用 SnapshotRestore CR 将快照还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或者制定保护计划。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：(*required*)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 <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。
   - **spec.storageClassMapping**：还原操作的源存储类到目标存储类的映射。将和 `sourceStorageClass`替换 `destinationStorageClass`为您环境中的信息。

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

3. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。

- **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
- **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
- *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。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. 使用正确的值填充文件后 `trident-protect-snapshot-restore-cr.yaml` 、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、将括号中的值替换为环境中的信息。
   - `snapshot`参数使用格式为的命名空间和快照名称 `<namespace>/<name>`。
   - 此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的`source1:dest1,source2:dest2`卷。

     例如：

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

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：(*required*)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
```

3. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
       - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
       - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
       - *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。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. 使用正确的值填充文件后 `trident-protect-snapshot-ipr-cr.yaml` 、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、将括号中的值替换为环境中的信息。例如：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name>
--snapshot <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、已完成或失败的还原操作的状态。

步骤

1. 使用以下命令检索还原操作的状态、将括号中的值替换为环境中的信息：

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

使用NetApp SnapMirror和Trident Protect 复制应用程序

使用Trident Protect，您可以利用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，无论是在同一集群内还是在不同集群之间。

还原和故障转移操作期间的命名空间标注和标签

在还原和故障转移操作期间、目标命名空间中的标签和标注会与源命名空间中的标签和标注相匹配。此时将添加源命名空间中目标命名空间中不存在的标签或标注、并覆盖已存在的任何标签或标注、以便与源命名空间中的值匹配。仅存在于目标命名空间上的标签或标注保持不变。

> 如果您使用RedHat OpenShift、请务必注意命名空间标注在OpenShift环境中的关键作用。命名空间标注可确保还原的Pod遵循OpenShift安全上下文约束(SCC)定义的适当权限和安全配置、并可在没有权限问题的情况下访问卷。有关详细信息，请参阅 "OpenShift安全上下文约束文档"。

在执行还原或故障转移操作之前、您可以通过设置Kubornetes环境变量来防止目标命名空间中的特定标注被覆盖 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。例如：

```
kubectl set env -n trident-protect deploy/trident-protect-controller-
manager
RESTORE_SKIP_NAMESPACE_ANNOTATIONS=<annotation_key_to_skip_1>,<annotation_
key_to_skip_2>
```

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例显示了一个源和目标命名空间、每个命名空间都具有不同的标注和标签。您可以查看目标命名空间在操作前后的状态、以及标注和标签在目标命名空间中的组合或覆盖方式。

在执行还原或故障转移操作之前

下表说明了执行还原或故障转移操作之前示例源和目标名称卷的状态：

命名空间	标注	标签
命名空间ns-1 (源)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE"	• 环境=生产 • 合规性=HIPAA • name=nS-1
命名空间ns-2 (目标)	• 标注.One/键："TRUE" • 标注三个/项："false"	• Role=database

还原操作之后

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。已添加某些密钥、某些密钥已被覆盖、并且`name`标签已更新以与目标命名空间匹配：

命名空间	标注	标签
命名空间ns-2 (目标)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE" • 标注三个/项："false"	• name=nS-2 • 合规性=HIPAA • 环境=生产 • Role=database

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。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

设置复制关系

设置复制关系涉及以下方面：

- 选择Trident Protect 拍摄应用程序快照的频率（包括应用程序的 Kubernetes 资源以及应用程序每个卷的卷快照）。
- 选择复制计划(包括Kubbernetes资源以及永久性卷数据)
- 设置创建快照的时间

步骤

1. 在源集群上为源应用程序创建AppVault。根据您的存储提供程序、修改中的示例"AppVault自定义资源"以适合您的环境：

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。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
- 。**spec.providerConfig**:(*required_*)存储使用指定提供程序访问AppVault所需的配置。为您的提供商选择一个BucketName和任何其他必要的详细信息。请记下所选的值、因为复制关系所需的其他CR文件会引用这些值。有关其他提供商的AppVault CRS示例、请参见"AppVault自定义资源"。
- 。**spec.providerCredentials**:(*required_*)存储对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_*)表示凭据值应来自密钥。
    - **key**:(*required*)要从中选择的密钥的有效密钥。
    - **name**:(*required*)包含此字段值的机密的名称。必须位于同一命名空间中。
  - * spec.providerCredentials.secretAccessKey*:(*required_*)用于访问提供程序的访问密钥。名称*应与*。spec.providerCredentials.valueFromSecret.name*。
- 。**spec.providerType**:(*required_*)用于确定提供备份的内容；例如、NetApp ONTAP S3、通用S3、Google Cloud或Microsoft Azure。可能值：
  - aws
  - azure
  - GCP
  - 常规S3
  - ONTAP S3
  - StorageGRID S3

c. 使用正确的值填充文件后 `trident-protect-appvault-primary-source.yaml` 、应用CR：

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

使用命令行界面创建**AppVault**

a. 创建AppVault、将括号中的值替换为您环境中的信息：

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

2. 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**:(*required_*)应用程序自定义资源的名称。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
- 。**spec.includedNamespaces**:(*required_*)一个由命名区域和关联标签组成的数组。使用命名空间名称、并可选择通过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属于此数组。

  示例**YAML**：

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

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

使用命令行界面创建源应用程序

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)为源应用程序创建快照。此快照将用作目标集群上应用程序的基础。如果跳过此步骤、则需要等待运行下一个计划快照、以便获得最新快照。

使用**CR**创建快照

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**：(*required*)此值必须与源应用程序的AppVault的metadata.name字段匹配。
- **spec.ApplicationRef**：(*required*)此值必须与源应用程序CR的metadata.name字段匹配。
- **spec.backup**䜣 刭：(*required*)此字段为必填字段、且值必须设置为0。
- *spec.enabled *：必须设置为true。
- 。**spec.granularity**:必须设置为 `Custom`。
- **spec.rec**发 规则：定义UTC时间的开始日期和重复间隔。
- **spec.snapshot**䜣 刭：必须设置为2。

YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-
46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

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

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

使用命令行界面创建快照

a. 创建快照、将括号中的值替换为您环境中的信息。例如：

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

4. 在目标集群上创建一个与源集群上应用的AppVault CR相同的源应用程序AppVault CR，并将其命名为(例如 `trident-protect-appvault-primary-destination.yaml`)。

5. 应用CR：

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

6. 在目标集群上为目标应用程序创建AppVault。根据您的存储提供程序、修改中的示例"AppVault自定义资源"以适合您的环境：

   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。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
      - 。**spec.providerConfig**:(*required_*)存储使用指定提供程序访问AppVault所需的配置。为您的提供商选择 `bucketName`以及任何其他必要的详细信息。请记下所选的值、因为复制关系所需的其他CR文件会引用这些值。有关其他提供商的AppVault CRS示例、请参见"AppVault自定义资源"。
      - 。**spec.providerCredentials**:(*required_*)存储对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_*)表示凭据值应来自密钥。
          - **key**:(*required*)要从中选择的密钥的有效密钥。
          - **name**:(*required*)包含此字段值的机密的名称。必须位于同一命名空间中。
        - * spec.providerCredentials.secretAccessKey*:(*required_*)用于访问提供程序的访问密钥。名称*应与*。spec.providerCredentials.valueFromSecret.name*。
      - 。**spec.providerType**:(*required_*)用于确定提供备份的内容；例如、NetApp ONTAP S3、通用S3、Google Cloud或Microsoft Azure。可能值：
        - aws
        - azure
        - GCP
        - 常规S3
        - ONTAP S3
        - StorageGRID S3

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

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

7. 创建App镜像 关系CR文件：

使用**CR**创建**App**镜像 关系

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) App镜像 关系自定义资源的名称。
- 。**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**:(*required_*)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**:(*required*)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.storageClassName**：(*required*)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建立对等关系的ONTAP Storage VM。
- **spec.rec**发 规则：定义UTC时间的开始日期和重复间隔。

YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

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

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

使用命令行界面创建**App**镜像 关系

a. 创建并应用App镜像 关系对象、将括号中的值替换为环境中的信息。例如：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --recurrence-rule <rule> --source-app
<my_source_app> --source-app-vault <my_source_app_vault> -n
<application_namespace>
```

8. (可 选)检查复制关系的状态：

```
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。此过程会停止复制关系，并将应用程序在目标集群上联机。如果源集群上的应用程序正在运行， Trident Protect 不会停止该应用程序。

步骤

1. 打开AppMirectorRelationship 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)在故障转移应用程序上创建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

重新同步操作将重新建立复制关系。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将被丢弃。

此过程会先停止目标集群上的应用程序、然后再重新建立复制。

故障转移期间写入目标应用程序的所有数据都将丢失。

步骤

1. 创建源应用程序的快照。
2. 打开AppMirectorRelationship 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. 如果您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。任何保留的计划都会导致卷快照失败。

反向重新同步故障转移复制关系

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保留下来。

步骤

1. 删除初始目标集群上的App镜像 关系CR。这会使目标成为源。如果新目标集群上仍有任何保护计划、请将其删除。
2. 通过将最初用于设置复制关系的CR文件应用于对等集群来设置复制关系。
3. 确保每个集群上的AppVault CRS均已准备就绪。
4. 在另一个集群上设置复制关系、并配置反向值。

反转应用程序复制方向

当您反转复制方向时， 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位置，然后再故障转移到目标应用程序。

在这种情况下、您将交换源和目标。

步骤

1. 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Sh关机Snapshot 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**：(*required*)此值必须与源应用程序CR文件的metadata.name字段匹配。

YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

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

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

使用命令行界面创建关闭快照

a. 创建关闭快照、将括号中的值替换为环境中的信息。例如：

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

2. 快照完成后、获取快照的状态：

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

3. 使用以下命令查找*shutdownSnapshot．statues.appArchivePath*的值，并记录文件路径的最后一部分(也称为基本名称；这将是最后一个斜杠后面的所有内容)：

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

4. 执行从目标集群到源集群的故障转移、并进行以下更改：

> 在故障转移过程的第2步中、将字段包含 `spec.promotedSnapshot`在App镜像 关系CR文件中、并将其值设置为您在上述第3步中记录的基本名称。

5. 执行中的反向重新同步步骤[反向重新同步故障转移复制关系]。
6. 在新的源集群上启用保护计划。

结果

反向复制会导致以下操作：

- 系统会为原始源应用程序的Kubbernetes资源创建一个快照。
- 通过删除原始源应用程序的Kubernetes资源(保留PVC和PV)、可以正常停止原始源应用程序的Pod。
- 关闭Pod后、将为应用程序的卷创建快照并进行复制。
- SnapMirror关系将中断、从而使目标卷做好读/写准备。
- 此应用程序的Kubornetes资源将使用在初始源应用程序关闭后复制的卷数据从关闭前的快照中还原。
- 反向重新建立复制。

将应用程序故障恢复到原始源集群

使用Trident Protect，您可以通过以下步骤序列在故障转移操作后实现“故障恢复”。在此恢复原始复制方向的工作流程中， Trident Protect 会将任何应用程序更改复制（重新同步）回原始源应用程序，然后再反转复制方向。

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、因为这会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复制关系]步骤。

2. 执行[反转应用程序复制方向]步骤。

删除复制关系

您可以随时删除复制关系。删除应用程序复制关系后、会导致两个单独的应用程序之间没有关系。

步骤

1. 删除App镜像 关系CR：

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

使用Trident Protect 迁移应用程序

您可以通过将备份或快照数据还原到其他集群或存储类来在集群或存储类之间迁移应用程序。

> 迁移应用程序时、为该应用程序配置的所有执行挂钩都会随该应用程序一起迁移。如果存在还原后执行挂钩、则它会在还原操作中自动运行。

备份和还原操作

要在以下情况下执行备份和还原操作、您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还原到同一集群。

步骤

1. 执行以下操作之一：
   a. "创建快照"(英文)
   b. "创建备份"(英文)
2. 在同一集群上、根据您是创建了快照还是备份、执行以下操作之一：
   a. "从快照还原数据"(英文)
   b. "从备份还原数据"(英文)

克隆到其他集群

要将应用程序克隆到不同的集群（执行跨集群克隆），请在源集群上创建备份，然后将备份还原到不同的集群。请确保目标集群上已安装Trident Protect。

> 您可以使用在不同集群之间复制应用程序"SnapMirror 复制"。

步骤

1. "创建备份"(英文)

2. 确保已在目标集群上为包含备份的对象存储分段配置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

您可以通过将快照还原到不同的目标存储类来将应用程序从一个存储类迁移到另一个存储类。

例如(从还原CR中排除密钥)：

```
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；请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**：(*required*)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。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

     ```
     ---
     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. (可选)如果您只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcesionCriteri**：(筛选时需要) `include or exclude`用于包含或排除在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.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
       - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
       - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。

- *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。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. 使用正确的值填充文件后 `trident-protect-snapshot-restore-cr.yaml` 、应用CR：

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

使用命令行界面还原快照

步骤

1. 将快照还原到其他命名空间、将括号中的值替换为环境中的信息。

   - `snapshot`参数使用格式为的命名空间和快照名称 `<namespace>/<name>`。
   - 此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的 `source1:dest1,source2:dest2`卷。

     例如：

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

管理Trident Protect 执行钩子

执行挂钩是一种自定义操作、您可以将其配置为与受管应用程序的数据保护操作结合运行。例如、如果您有一个数据库应用程序、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。这样可以确保应用程序一致的快照。

执行挂钩的类型

Trident Protect 支持以下几种执行钩子类型，具体取决于它们的运行时机：

- 预快照
- 快照后
- 预备份
- 备份后
- 还原后
- 故障转移后

执行顺序

运行数据保护操作时、执行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会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挂钩、但操作前这些挂钩的执行顺序既不能保证也不可配置。
2. 如果适用，则会发生文件系统冻结。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。
3. 执行数据保护操作。
4. 如果适用、冻结的文件系统将被解除冻结。
5. 任何适用的自定义操作后执行挂钩都会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后挂机、但这些挂机在操作后的执行顺序既不能保证也不可配置。

如果创建多个相同类型的执行挂钩(例如、预快照)、则无法保证这些挂钩的执行顺序。但是、可以保证不同类型的挂钩的执行顺序。例如、以下是具有所有不同类型挂钩的配置的执行顺序：

1. 已执行预快照挂钩
2. 已执行后快照挂钩
3. 已执行备份前的挂钩
4. 已执行备份后挂钩

只有在运行不使用现有快照的备份时、上述顺序示例才适用。

在生产环境中启用执行钩脚本之前，应始终对其进行测试。您可以使用 "kubectl exec" 命令方便地测试脚本。在生产环境中启用执行挂钩后、请测试生成的快照和备份、以确保它们一致。为此、您可以将应用程序克隆到临时命名空间、还原快照或备份、然后测试应用程序。

有关自定义执行挂钩的重要注意事项

在为应用程序规划执行挂钩时，请考虑以下几点。

- 执行挂钩必须使用脚本执行操作。许多执行挂钩可以引用同一个脚本。
- Trident Protect 要求执行钩子使用的脚本以可执行 shell 脚本的格式编写。
- 脚本大小限制为96 KB。
- Trident Protect 使用执行钩子设置和任何匹配条件来确定哪些钩子适用于快照、备份或恢复操作。

> 由于执行挂钩通常会减少或完全禁用其运行的应用程序的功能，因此您应始终尽量缩短自定义执行挂钩运行所需的时间。如果使用关联的执行挂钩启动备份或快照操作、但随后将其取消、则在备份或快照操作已开始时、仍允许运行这些挂钩。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。对于在所有容器上使用相同容器映像的应用程序、筛选器非常有用、但可能会将每个映像用于不同的用途(例如Elasticsearch)。通过筛选器、您可以创建执行挂钩在某些容器上运行的方案、但不一定是所有相同的容器上运行的方案。如果为单个执行钩创建多个筛选器、则这些筛选器将与逻辑运算符和运算符结合使用。每个执行连接最多可以有10个活动筛选器。

添加到执行挂钩的每个过滤器都使用正则表达式来匹配集群中的容器。当钩子与容器匹配时，钩子将在该容器上运行其关联的脚本。过滤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过滤器。有关Trident Protect 在执行钩子过滤器中支持的正则表达式语法的详细信息，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之后运行的执行挂钩、并且还原或克隆源和目标位于不同的命名空间中、则命名空间筛选器仅会应用于目标命名空间。

执行钩示例

请访问 "NetApp Verda GitHub项目" 、下载适用于Apache cassandr和Elascearch等常见应用程序的真实执行挂钩。您还可以查看示例并了解如何构建自己的自定义执行挂钩。

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钩子。您需要拥有所有者、管理员或成员权限才能创建执行钩子。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要运行执行挂钩的应用程序的Kubernetes名称。
   - *spec.stage *：(*required*)一个字符串，指示执行挂钩应在操作期间的哪个阶段运行。可能值：
     - 预
     - 发布
   - **spec.action**：(*required*)一个字符串，指示执行挂钩将执行的操作，假设指定的任何执行挂钩过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - *spec.enabled *：(可 选)指示此执行挂钩是启用还是禁用。如果未指定、则默认值为true。
   - **spec.hookSource**：(*required*)包含base64编码的挂钩脚本的字符串。
   - *spec.timeout *：(可 选)一个数字，用于定义允许执行挂钩运行多长时间(以分钟为单位)。最小值为1分钟、如果未指定、则默认值为25分钟。
   - **spic.arge**件：(可 选)可为执行挂钩指定的YAML参数列表。
   - *spec.匹 配Criteria：(可 选)标准键值对的可选列表，每个对构成执行挂钩筛选器。每个执行挂钩最多可以添加10个筛选器。
   - **spec.**匹 配**Cricera.type**：(可 选)标识执行挂钩筛选器类型的字符串。可能值：
     - 内容管理器映像
     - 内容名
     - 播客名称
     - PodLabel
     - NamespaceName
   - **spec.**匹 配**Cricery.value**：(可 选)用于标识执行挂钩筛选器值的字符串或正则表达式。

     YAML示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. 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、将括号中的值替换为环境中的信息。例如：

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

卸载Trident Protect

如果您要从试用版升级到完整版产品，可能需要移除Trident Protect 组件。

要移除Trident Protect，请执行以下步骤。

步骤

1. 删除Trident Protect CR 文件：

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

2. 移除Trident保护：

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

3. 移除Trident Protect 命名空间：

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