使用Trident Protect 保护应用程序

Trident

NetApp
February 02, 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

- Trident Protect备份仅支持Linux计算节点。Windows 计算节点不支持备份操作。
- 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"。
- 确保至少存在一个 VolumeSnapshotClass。更多信息，请参阅"VolumeSnapshotClass"。

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 Trident 24.02 或更新版本（建议使用 Trident 24.10）。
- 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 在数据保护操作期间为 KubeVirt 虚拟机提供文件系统冻结和解冻功能，以确保数据一致性。虚拟机冻结操作的配置方法和默认行为在Trident Protect 的不同版本中有所不同，较新的版本通过 Helm chart 参数提供了简化的配置。

在恢复操作期间，任何 `VirtualMachineSnapshots`为虚拟机 (VM) 创建的数据不会被恢复。

Trident Protect 25.10 及更新版本

Trident Protect 在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统，以确保一致性。从Trident Protect 25.10 开始，您可以使用以下方法禁用此行为： `vm.freeze` Helm Chart 安装过程中的参数。该参数默认启用。

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

Trident Protect 24.10.1 至 25.06

从Trident Protect 24.10.1 开始， Trident Protect 会在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。您也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
```

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
```

SnapMirror复制的要求

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

- 内部NetApp FAS、AFF和ASA集群
- 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：

- * NetApp Trident *：使用ONTAP作为后端服务的源 Kubernetes 集群和目标 Kubernetes 集群上都必须存在NetApp 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`：NFS
- `ontap-san`：iSCSI
- `ontap-san`：足球俱乐部
- `ontap-san`：NVMe/TCP（要求最低 ONTAP 版本 9.15.1）

- 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" 。

  从ONTAP 9.10.1开始、所有许可证均以NetApp许可证文件(NLG)的形式提供、NLG是一个支持多项功能的文件。有关详细信息、请参见 "ONTAP One附带的许可证" 。

  仅支持 SnapMirror 异步保护。

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。

- **NetApp Trident** 和 **SVM**：对等远程 SVM 必须可供目标集群上的 NetApp 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。

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。

SnapMirror复制的 Kubernetes 集群要求

确保您的 Kubernetes 集群满足以下要求：

- **AppVault** 可访问性：源集群和目标集群都必须具有网络访问权限，才能从 AppVault 读取和写入应用程序对象复制。
- 网络连接：配置防火墙规则、存储桶权限和 IP 允许列表，以实现跨 WAN 的集群和 AppVault 之间的通信。

许多企业环境在 WAN 连接中实施严格的防火墙策略。在配置复制之前，请与您的基础设施团队验证这些网络要求。

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。

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. 使用 Helm 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：

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

3. （可选）要启用调试日志记录（建议用于故障排除），请使用：

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

调试日志记录有助于NetApp支持人员排除故障，而无需更改日志级别或重现问题。

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：

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

例如：

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

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

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

要获取 Helm Chart，首先需要在可以访问互联网的计算机上下载 Helm Chart。 `helm pull trident-protect --version 100.2510.0 --repo https://netapp.github.io/trident-protect-helm-chart`然后复制结果 `trident-protect-100.2510.0.tgz 将文件复制到您的离线环境并进行安装 `helm install trident-protect ./trident-protect-100.2510.0.tgz 而不是在最后一步中使用存储库引用。

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：

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

这 `imageRegistry`和 `imagePullSecrets`这些值适用于所有组件图像，包括 `resourcebackup`和 `resourcerestore`。如果您将镜像推送到注册表中的特定存储库路径（例如， `example.com:443/my-repo`），请在注册表字段中包含完整路径。这将确保所有图像都从此处提取。 `<private-registry-url>/<image-name>:<tag>`。

7. 使用 Helm 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：

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

8. （可选）要启用调试日志记录（建议用于故障排除），请使用：

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

调试日志记录有助于NetApp支持人员排除故障，而无需更改日志级别或重现问题。

有关其他 Helm Chart 配置选项，包括AutoSupport设置和命名空间过滤，请参阅 "定制Trident Protect 安装"。

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/25.10.0/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/25.10.0/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/25.10.0/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/25.10.0/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. 创建完成脚本：

```
tridentctl-protect completion bash > 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. 创建完成脚本：

```
tridentctl-protect completion zsh > 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。

定制Trident Protect 安装

您可以自定义Trident Protect 的默认配置，以满足您环境的特定要求。

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值：

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

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

3. 应用文件中的值 `resourceLimits.yaml`：

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

自定义安全上下文约束

安装Trident Protect 后，您可以使用配置文件来修改Trident Protect 容器的 OpenShift 安全上下文约束 (SCC)。这些约束定义了 Red Hat OpenShift 集群中 pod 的安全限制。

步骤

1. 创建一个名为的文件 `sccconfig.yaml`。
2. 将scc选项添加到文件中、然后根据环境的需要修改参数。

   以下示例显示了SCC选项参数的默认值：

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

   下表介绍了SCC选项的参数：

参数	Description	Default
创建	确定是否可以创建SCC资源。只有在将设置为 `true`且Helm安装过程标识OpenShift环境时、才会创建SCC资源 `scc.create`。如果不在OpenShift上运行，或者如果 `scc.create`设置为 `false`，则不会创建任何SCC资源。	true
name	指定SCC的名称。	Trident保护作业
优先级	定义SCC的优先级。优先级值较高的SCC会在优先级值较低的SCC之前进行评估。	1

3. 应用文件中的值 `sccconfig.yaml`：

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

此操作会将默认值替换为文件中指定的值 `sccconfig.yaml`。

配置其他Trident Protect 舵图设置

您可以自定义AutoSupport设置和命名空间过滤以满足您的特定要求。下表描述了可用的配置参数：

参数	Type	Description
自动支持代理	string	为NetApp AutoSupport连接配置代理 URL。使用此功能通过代理服务器路由支持包上传。例子：`http://my.proxy.url` 。
自动支持.不安全	boolean	设置为时跳过AutoSupport代理连接的 TLS 验证 `true`。仅用于不安全的代理连接。 （默认： `false` ）
自动支持已启用	boolean	启用或禁用每日Trident Protect AutoSupport捆绑包上传。设置为 `false`每日定时上传功能已禁用，但您仍然可以手动生成支持包。 （默认： `true` ）
恢复跳过命名空间注释	string	要从备份和恢复操作中排除的命名空间注释的逗号分隔列表。允许您根据注释过滤命名空间。
restoreSkipNamespaceLabels	string	要从备份和恢复操作中排除的命名空间标签的逗号分隔列表。允许您根据标签过滤命名空间。

您可以使用 YAML 配置文件或命令行标志配置这些选项：

使用 **YAML** 文件

步骤

1. 创建配置文件并命名 `values.yaml`。
2. 在您创建的文件中，添加您想要自定义的配置选项。

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

3. 填充后 `values.yaml`具有正确值的文件，应用配置文件：

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

使用 **CLI** 标志

步骤

1. 使用以下命令 `--set`标志来指定单个参数：

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

将**Trident Protect Pod** 限制在特定节点上

您可以使用 Kubernetes nodeSelector 节点选择约束，根据节点标签来控制哪些节点有资格运行Trident Protect pod。默认情况下， Trident Protect 仅限于运行 Linux 的节点。您可以根据需要进一步自定义这些限制条件。

步骤

1. 创建一个名为的文件 `nodeSelectorConfig.yaml`。
2. 将nodeSelector选项添加到文件中、然后修改文件以添加或更改节点标签、从而根据环境需求进行限制。例如、以下文件包含默认操作系统限制、但也针对特定区域和应用程序名称：

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. 应用文件中的值 `nodeSelectorConfig.yaml`：

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

此操作会将默认限制替换为您在文件中指定的限制 `nodeSelectorConfig.yaml`。

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。

监控Trident保护资源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 开源工具来监控Trident Protect 保护的资源的健康状况。

kube-state-metrics 服务从 Kubernetes API 通信生成指标。将其与Trident Protect 结合使用，可以显示有关环境中资源状态的有用信息。

Prometheus 是一个工具包，它可以接收 kube-state-metrics 生成的数据，并将其呈现为关于这些对象的易于阅读的信息。kube-state-metrics 和 Prometheus 共同提供了一种方法，让您可以监控使用Trident Protect 管理的资源的健康状况和状态。

警报管理器是一项服务、可接收Prometheus等工具发送的警报、并将其路由到您配置的目标。

> 这些步骤中包含的配置和指导仅为示例；您需要对其进行自定义以符合您的环境。有关具体说明和支持、请参见以下官方文档：
>
> - "Kube-state-metrics 文档"
> - "Prometheus文档"
> - "记录"

第1步：安装监控工具

要在Trident Protect 中启用资源监控，您需要安装和配置 kube-state-metrics、Promethus 和 Alertmanager。

安装Kube-state-metrics

您可以使用Helm安装Kube-state-metrics。

步骤

1. 添加Kube-state-metrics Helm图表。例如：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. 将 Prometheus ServiceMonitor CRD 应用到集群：

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

3. 为Helm图表创建配置文件(例如 `metrics-config.yaml`)。您可以根据您的环境自定义以下示例配置：

metrics-config.yaml：Kube-state-metrics Helm图表配置

```
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

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

# Collect metrics from all namespaces
namespaces: ""

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

4. 通过部署Helm图表来安装Kube-state-metrics。例如：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. 按照以下说明配置 kube-state-metrics，以生成Trident Protect 使用的自定义资源的指标： "Kube-state-metrics自定义资源文档" 。

安装 Prometheus

您可以按照中的说明安装Prometheus "Prometheus文档"。

安装活动管理器

您可以按照中的说明安装提示管理器 "记录"。

第2步：配置监控工具以协同工作

安装监控工具后、您需要将其配置为协同工作。

步骤

1. 将Kube-state-metrics与Prometheus集成。编辑Prometheus配置文件(`prometheus.yaml`)并添加Kube-state-metrics服务信息。例如：

   prometheus.yaml：kube-state-metrics 服务与 Prometheus 的集成

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

2. 配置Prometheus以将警报路由到警报管理器。编辑Prometheus配置文件(`prometheus.yaml`)并添加以下部分：

prometheus.yaml：向 **Alertmanager** 发送警报

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

结果

Prometheus现在可以从Kube-state-metrics收集指标、并可向警报管理器发送警报。现在、您可以配置触发警报的条件以及警报的发送位置。

第3步：配置警报和警报目标

将这些工具配置为协同工作后、您需要配置触发警报的信息类型以及警报的发送位置。

警报示例：备份失败

以下示例定义了备份自定义资源的状态设置为5秒或更长时间时触发的严重警报 `Error`。您可以自定义此示例以匹配您的环境、并将此YAML段包含在您的配置文件中 `prometheus.yaml`：

rules.yaml：定义失败备份的 **Prometheus** 警报

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

将警报管理器配置为向其他通道发送警报

通过在文件中指定相应的配置、您可以将警报管理器配置为向其他通道发送通知、例如电子邮件、PagerDty、Microsoft团队或其他通知服务 `alertmanager.yaml`。

以下示例将配置警报管理器、以便向SI延 时信道发送通知。要根据您的环境自定义此示例、请将此密钥的值替换 `api_url`为您的环境中使用的Slackwebhook URL：

alertmanager.yaml：向 Slack 频道发送警报

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

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. 填充后 `trident-protect-support-bundle.yaml`具有正确值的文件，应用 CR：

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

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> -n trident-protect
```

监视和检索支持包

使用任一方法创建支持包后，您可以监视其生成进度并将其检索到本地系统。

步骤

1. 等待 `status.generationState`到达 `Completed`状态。您可以使用以下命令监控生成进度：

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

2. 将支持包检索到您的本地系统。从已完成的AutoSupport包中获取复制命令：

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

找到 `kubectl cp`从输出中执行命令并运行它，用您喜欢的本地目录替换目标参数。

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。

- 从版本 24.10 升级时，升级期间运行的快照可能会失败。此失败不会阻止将来创建快照（无论是手动快照还是计划快照）。如果升级期间快照失败，您可以手动创建新快照以确保应用程序受到保护。

  为避免潜在的故障，您可以在升级前禁用所有快照计划，然后在升级后重新启用。但是，这会导致升级期间丢失所有计划的快照。

- 对于私有镜像仓库安装，请确保目标版本所需的 Helm Chart 和镜像在您的私有镜像仓库中可用，并验证您的自定义 Helm 值与新 Chart 版本兼容。更多信息，请参阅"从私有注册表安装Trident Protect"。

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：

> 如果您是从 25.06 之前的版本升级，则需要执行此步骤，因为 CRD 现在已包含在Trident Protect Helm 图表中。

a. 运行此命令将 CRD 的管理从 `trident-protect-crds`到 `trident-protect`：

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

b. 运行此命令删除 `trident-protect-crds`图表：

> 不要卸载 `trident-protect-crds`图表使用 Helm，因为这可能会删除您的 CRD 和任何相关数据。

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

3. 升级Trident保护：

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

> 您可以通过添加以下内容来配置升级期间的日志级别。 `--set logLevel=debug`升级命令。默认日志级别为 `warn`。建议启用调试日志记录进行故障排除，因为它可以帮助NetApp支持人员诊断问题，而无需更改日志级别或重现问题。

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。

在应用程序上执行数据保护操作时，您需要手动或从命令行创建 AppVault CR。AppVaultCR 特定于您的环境，您可以使用本页上的示例作为创建 AppVault CR 的指南。

> 确保 AppVault CR 位于安装了Trident Protect 的集群上。如果 AppVault CR 不存在或您无法访问，命令行会显示错误。

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 和数据移动器可以向提供商和任何相关资源进行身份验证。

数据移动工具存储库密码

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 密钥。如果您不指定密钥， Trident Protect 将使用默认密码。

- 手动创建 AppVault CR 时，使用 **spec.dataMoverPasswordSecretRef** 字段指定密钥。
- 使用Trident Protect CLI 创建 AppVault 对象时，请使用 `--data-mover-password-secret-ref`用于指定密钥的参数。

创建数据移动工具存储库密码密钥

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

- 根据所使用的数据移动工具、您只需包含该数据移动工具的相应密码即可。例如、如果您正在使用Restic、并且将来不打算使用KONIA、则在创建密钥时只能包含Restic密码。
- 请将密码保存在安全的地方。您将需要它来还原同一集群或其他集群上的数据。如果集群或`trident-protect`命名空间被删除后，没有密码您将无法恢复备份或快照。

使用CR

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

使用命令行界面

```
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

当您访问与 S3 兼容的存储（例如 Amazon S3、通用 S3）时， "StorageGRID S3" ， 或者 "ONTAP S3"使用Trident Protect 时，您需要确保提供的用户凭据具有访问存储桶的必要权限。以下是授予使用Trident Protect 进行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 策略的更多信息，请参阅 "Amazon S3 文档" 。

用于 Amazon S3（AWS）身份验证的 EKS Pod Identity

Trident Protect 支持 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 凭证存储在 Kubernetes 机密中。

EKS Pod Identity 与Trident Protect 的要求

在将 EKS Pod Identity 与Trident Protect 结合使用之前，请确保以下事项：

- 您的 EKS 集群已启用 Pod Identity。
- 您已创建具有必要的 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 角色与服务账户关联的详细说明，请参阅 "AWS EKS Pod Identity 文档"。

AppVault 配置 使用 EKS Pod Identity 时，请使用以下配置配置您的 AppVault CR `useIAM: true`标记而不是明确的凭证：

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

云提供商的**AppVault**密钥生成示例

定义 AppVault CR 时，您需要包含凭证以访问提供商托管的资源，除非您使用 IAM 身份验证。如何生成凭证密钥将根据提供商的不同而有所不同。以下是几个提供商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证创建密钥。

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

以下是每个提供程序的AppVault定义示例。

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。

- 您可以选择指定一个包含用于Restic和Koria存储库加密的自定义密码的Kubernetes密钥。有关详细信息、请参见 [数据移动工具存储库密码] 。
- 对于Amazon S3 (AWS) AppVault对象、您可以选择指定sSession令牌、如果使用单点登录(SSO)进行身份验证、则此令牌非常有用。此令牌是在中为提供程序生成密钥时创建的云提供商的AppVault密钥生成示例。
- 对于S3 AppVault对象、您可以选择使用密钥为出站S3流量指定出口代理URL `spec.providerConfig.S3.proxyURL`。

Google Cloud

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

Amazon S3 (AWS)

```yaml
---
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 数据移动器的 EKS 环境，您可以删除 `providerCredentials` 部分并添加 `useIAM: true`根据 `s3`配置。

Microsoft Azure

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

通用 S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGRID S3

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

使用**Trident Protect CLI** 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。

- 您可以选择指定一个包含用于Restic和Koria存储库加密的自定义密码的Kubernetes密钥。有关详细信息、请参见 [数据移动工具存储库密码] 。
- 对于S3 AppVault对象、您可以选择使用参数为出站S3流量指定出口代理URL `--proxy-url <ip_address:port>`。

Google Cloud

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

Amazon S3 (AWS)

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

Microsoft Azure

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

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 `providerConfig.s3`配置选项

请参阅下表了解 S3 提供程序配置选项：

参数	Description	Default	示例
`providerConfig.s3.skipCertValidation`	禁用 SSL/TLS 证书验证。	false	“真”，“假”
`providerConfig.s3.secure`	启用与 S3 端点的安全 HTTPS 通信。	true	“真”，“假”
`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`	提供用于 SSL/TLS 验证的自定义根 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 标识。	false	true false

查看AppVault信息

您可以使用Trident Protect CLI 插件查看有关您在集群上创建的 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` 。

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

确保已删除要删除的AppVault正在使用的所有快照和备份CRS。

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属于此列表。标签选择器是可选的、可用于筛选每个指定命名空间中的资源。
      - 。**spec.includedClusterScopedResources**:(可 选)使用此属性指定要包含在应用程序定义中的集群范围资源。此属性允许您根据资源的组、版本、种类和标签来选择这些资源。
        - **groupVersionKind**：(*required*)指定集群范围资源的API组、版本和类型。
        - **labelSelecter**：(可 选)根据集群范围的资源的标签对其进行筛选。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (可选) 此注解仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。指定此应用程序在快照期间是否可以写入文件系统。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，并且在快照期间文件系统将被冻结。如果指定了注解，但应用程序定义中没有虚拟机，则忽略该注解。如未特别说明，则申请流程如下："全球Trident Protect 冷冻设置" 。

如果您需要在创建应用程序后应用此标注、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"
```

+
YAML示例：

+

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. （可选）添加包含或排除标有特定标签的资源的过滤：

   - **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"`。

> 当两者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`首先运行，然后 `labelSelector`应用于结果资源。

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"]
```

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
```

步骤

1. 使用以下示例之一创建并应用应用程序定义、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示例中显示的参数在应用程序定义中包括名称和资源。

   创建应用时，您可以选择使用注解来指定应用在快照期间是否可以写入文件系统。这仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。如果您将注释设置为 `true`该应用程序忽略全局设置，可以在快照期间写入文件系统。如果你把它设置为 `false`该应用程序忽略全局设置，导致文件系统在快照期间冻结。如果使用了注解，但应用程序定义中没有虚拟机，则该注解将被忽略。如果您不使用注解，应用程序将遵循以下规则："全球Trident Protect 冷冻设置" 。

   要在使用命令行界面创建应用程序时指定标注、可以使用标志。 `--annotation`

   - 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>
```

- 创建应用程序并为文件系统冻结行为配置本地应用程序设置：

```
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">
```

您可以使用 `--resource-filter-include`和 `--resource-filter-exclude`用于包含或排除资源的标志 `resourceSelectionCriteria`例如组、种类、版本、标签、名称和命名空间，如下例所示：

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

使用Trident Protect 保护应用程序

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。

如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、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；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。

如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、快照或克隆中。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的S3备份操作。如果令牌在备份操作期间过期、则操作可能会失败。

- 有关检查当前会话令牌到期时间的详细信息、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；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）：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。

   YAML示例：

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

3. 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> --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、所有备份均为增量备份。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、以最大程度地降低与还原相关的风险。

支持的备份注释

下表描述了创建备份 CR 时可以使用的注释：

标注	Type	Description	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` 创建非增量备份。最佳实践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/snapshot-completion-timeout	string	完成整个快照操作允许的最长时间。	60米
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	卷快照达到可用状态所需的最长时间。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	string	创建卷快照允许的最长时间。	5米
protect.trident.netapp.io/pvc-bind-timeout-sec	string	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位） `Bound`操作失败前的阶段。	1200（20分钟）

创建数据保护计划

保护策略通过按照定义的计划创建快照、备份或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，并可以指定要保留的副本数量。您可以使用 full-backup-rule 注释来安排非增量式完整备份。默认情况下，所有备份都是增量的。定期执行完整备份以及其间的增量备份有助于降低与恢复相关的风险。

- 您可以通过设置 `backupRetention`归零， `snapshotRetention`为大于零的值。环境 `snapshotRetention`为零意味着任何计划的备份仍将创建快照，但这些快照是临时的，并在备份完成后立即被删除。
- 如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、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；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**: (必需) 要保留的备份数量。零表示不应创建备份（仅快照）。
   - **spec.backupReclaimPolicy**: (可选) 确定如果备份 CR 在其保留期内被删除，则备份会发生什么情况。保留期过后，备份文件总是会被删除。可能的值（区分大小写）：
     - `Retain` (默认)
     - `Delete`
   - **spec.snapshotRetention**: (必需) 要保留的快照数量。零表示不创建任何快照。
   - **spec.snapshotReclaimPolicy**: (可选) 确定如果快照 CR 在其保留期内被删除，则快照会发生什么情况。保留期过后，快照总是会被删除。可能的值（区分大小写）：
     - `Retain`
     - `Delete`（默认）
   - 。**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`)
     - `Custom`
   - **spec.dayOfMonth**：（可选）计划应运行的月份日期（1 - 31）。如果粒度设置为 `Monthly`。该值必须以字符串形式提供。
   - **spec.dayOfWeek**：（可选）计划应运行的星期几（0 - 7）。值 0 或 7 表示星期日。如果粒度设置为 `Weekly`。该值必须以字符串形式提供。
   - **spec.hour**：（可选）计划应运行的小时数（0 - 23）。如果粒度设置为 `Daily`， `Weekly`， 或者 `Monthly`。该值必须以字符串形式提供。
   - **spec.minute**：（可选）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`， 或者 `Monthly`。该值必须以字符串形式提供。

     备份和快照计划的示例 YAML：

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

仅快照计划的示例 YAML：

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

3. 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> \
  --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>
```

以下选项可让您对日程安排进行更多控制：

- 完整备份计划：使用 `--full-backup-rule` 标记以安排非增量式完整备份。此标志仅适用于 `--granularity Daily`。可能的值：
  - `Always`每天创建完整备份。
  - 具体工作日：指定一个或多个日期，用逗号分隔（例如， `"Monday,Thursday"`）。有效值：星期一、星期二、星期三、星期四、星期五、星期六、星期日。

    这 `--full-backup-rule` 该标志位不适用于按小时、按周或按月划分的粒度。
- 仅快照计划：设置 `--backup-retention 0` 并指定一个大于零的值 `--snapshot-retention`。

支持的日程注释

下表描述了创建计划变更请求 (CR) 时可以使用的注释：

标注	Type	Description	默认值
protect.trident.netapp.io/full-backup-rule	string	指定安排完整备份的规则。你可以将其设置为 `Always` 您可以根据需要进行持续完整备份或自定义备份。例如，如果您选择按日粒度进行备份，则可以指定应进行完整备份的星期几（例如， `"Monday,Thursday"`）。有效的工作日值为：星期一、星期二、星期三、星期四、星期五、星期六、星期日。请注意，此注释只能用于已包含以下内容的日程表： `granularity` 设置为 `Daily`。	未设置（所有备份均为增量备份）
protect.trident.netapp.io/snapshot-completion-timeout	string	完成整个快照操作允许的最长时间。	60米
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	卷快照达到可用状态所需的最长时间。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	string	创建卷快照允许的最长时间。	5米
protect.trident.netapp.io/pvc-bind-timeout-sec	string	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位） `Bound`操作失败前的阶段。	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。

确保回收策略设置为 `Delete`从对象存储中删除所有备份数据。该策略的默认设置是 `Retain`以避免意外数据丢失。如果政策没有改变 `Delete`，备份数据将保留在对象存储中，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。

还原应用程序

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。
- 对于 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，对于 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来自定义恢复操作。欲了解更多信息，请参阅 "使用高级Trident Protect 恢复设置"。

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 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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"。

当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

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；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。

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

3. (可 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：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **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
```

使用命令行界面

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>
```

您可以随时将备份还原到原始命名空间。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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"。

当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

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；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：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **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
```

使用命令行界面

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。

- 当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。
- 使用 CR 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

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。
- 确保在运行 AppVault CR 时，本地环境可以连接到 AppVault CR 中定义的对象存储桶。 `tridentctl-protect get appvaultcontent`命令。如果网络限制阻止访问，请改为从目标集群上的 pod 内运行Trident Protect CLI。
- 确保AWS会话令牌到期时间足以执行任何长时间运行的还原操作。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到期时间的详细信息、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
```

使用命令行界面

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> \
--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。

- 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。如果您尝试恢复到 `StorageClass`如果使用不同的存储后端，则恢复操作将失败。
- 使用 CR 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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；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。

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

3. (可 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：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **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
```

使用命令行界面

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。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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；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：

> Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **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
```

使用命令行界面

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 <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}'
```

使用高级**Trident Protect** 恢复设置

您可以使用高级设置（例如注释、命名空间设置和存储选项）自定义恢复操作，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 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 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。欲了解更多信息，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：

```
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
```

> 执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置其他Trident Protect 舵图设置"。

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

支持的字段

本节介绍可用于恢复操作的其他字段。

存储类别映射

这 `spec.storageClassMapping`属性定义从源应用程序中的现有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别的集群之间迁移应用程序时或更改 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

本节列出了系统中支持配置各种行为的注解。如果用户未明确设置注解，系统将使用默认值。

标注	Type	Description	默认值
protected.trident.netapp.io/data-mover-timeout-sec	string	允许数据移动设备操作停止的最长时间（以秒为单位）。	“300”
protected.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	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位）`Bound`操作失败前的阶段。适用于所有恢复 CR 类型（备份恢复、备份就地恢复、快照恢复、快照就地恢复）。如果您的存储后端或集群经常需要更多时间，请使用更高的值。	1200（20分钟）

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 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 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。欲了解更多信息，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：

```
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
```

执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置其他Trident Protect 舵图设置"。

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"。

故障转移和反向操作期间的执行挂钩

当使用 AppMirror 关系保护您的应用程序时，您应该在故障转移和反转操作期间注意与执行挂钩相关的特定行为。

- 在故障转移期间，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，执行挂钩仍存在于应用程序中，并将在任何相关操作期间执行。
- 在反向同步或反向重新同步期间，应用程序上所有现有的执行钩子都将被移除。当源应用程序成为目标应用程序时，这些执行钩子将失效，并会被删除以阻止其执行。

要了解有关执行钩子的更多信息，请参阅"管理Trident Protect 执行钩子"。

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> -n
trident-protect
```

2. 在源集群上、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**:(*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）在源集群上，对源应用程序进行快照。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，则需要等待下一次计划快照运行，以便获得最新的快照。要创建按需快照，请参阅 "创建按需快照"。

4. 在源集群上，创建复制计划 CR：

除了下面提供的计划外，建议创建一个单独的每日快照计划，保留期为 7 天，以便在对等 ONTAP 集群之间维护通用快照。这可确保快照最多可用 7 天，但保留期可根据用户需求自定义。

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，而不是所有数据。

如果应用程序的现有计划已经满足所需的保留要求，则不需要额外的计划。

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.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
  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. 使用正确的值填充文件后 `trident-protect-schedule.yaml` 、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，并将括号中的值替换为您环境中的信息：

```
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. 在目标集群上，创建一个与您在源集群上应用的AppVault CR完全相同的源应用程序AppVault CR，并将其命名为(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 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 时才会自动创建命名空间。

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.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) 选择集群上有效的存储类的名称。存储类必须链接到与源环境建立对等连接的ONTAP存储 VM。如果未提供存储类，则默认使用集群上的默认存储类。
- **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 AppMirrorRelationship 对象，并将括号中的值替换为您环境中的信息：

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

- 示例： *

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

9. (可 选)在目标集群上、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. 在目标集群上，编辑AppMirorRelationship 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. 可选：在源集群上、创建源应用程序的快照。这样可确保捕获源集群的最新更改。
2. 在目标集群上，编辑AppMirorRelationship 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. 在初始目标集群上、删除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. 确保为新目标(初始源集群)配置了两个AppVault CRS。
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. 在源集群上、创建一个关闭快照：

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. 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*shutdownfapp.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. 在当前目标集群上、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

您可以通过恢复备份数据在集群之间或不同的存储类之间迁移您的应用程序。

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

您可以通过将备份还原到目标存储类，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。

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

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
```

使用命令行界面

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>
```

手动运行执行挂钩

您可以手动运行执行挂钩以进行测试、或者在发生故障后需要手动重新运行挂钩。要手动运行执行挂钩、您需要具有所有者、管理员或成员权限。

手动运行执行挂钩包含两个基本步骤：

1. 创建资源备份、此备份用于收集资源并为其创建备份、从而确定挂钩的运行位置
2. 对备份运行执行挂钩

第**1**步：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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要为其创建备份资源的应用程序的Kubernetes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}'
     ```

     YAML示例：

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

3. 使用正确的值填充CR文件后、应用CR：

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

使用命令行界面

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

2. 查看备份状态。您可以重复使用以下示例命令、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>
```

第**2**步：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；请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中的应用程序名称匹配。
   - **spec.appVaultRef**：(*required*)确保此值与您在步骤1中创建的ResourceBackup CR中的appVaultRef匹配。
   - **spec.appArchivePath**：确保此值与您在步骤1中创建的ResourceBackup CR中的appArchivePath匹配。

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

   - **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.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。此挂钩运行不会在任何其他阶段运行挂钩。可能值：
     - 预
     - 发布

     YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

3. 使用正确的值填充CR文件后、应用CR：

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

使用命令行界面

步骤

1. 创建手动执行挂钩运行请求：

```
tridentctl protect create exechooksrun <my_exec_hook_run_name>
-n <my_app_namespace> --action snapshot --stage <pre_or_post>
--app <my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. 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 exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. 描述exech本 运行对象以查看最终详细信息和状态：

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

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：

   25.06 及更高版本不需要此步骤。

   ```
   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
   ```