使用Trident Protect 保护应用程序

Trident

NetApp
January 15, 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：

- 亚马逊弹性 Kubernetes 服务 (EKS)
- 谷歌 Kubernetes 引擎 (GKE)
- Microsoft Azure Kubernetes 服务 (AKS)
- 红帽 OpenShift
- SUSE Rancher
- 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"。

Trident Protect 存储后端兼容性

Trident Protect 支持以下存储后端：

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP存储阵列
- Google Cloud NetApp Volumes
- Azure NetApp Files

请确保您的存储后端满足以下要求：

- 确保连接到集群的NetApp存储使用的是Trident 24.02 或更高版本（建议使用Trident 24.10）。
- 请确保您拥有NetApp ONTAP存储后端。
- 请确保您已配置用于存储备份的对象存储桶。
- 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-economy 容量要求

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快照目录的卷不兼容。对于这些应用，您需要通过在ONTAP存储系统上运行以下命令来隐藏快照目录：

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

您可以通过对每个 nas-economy 卷运行以下命令来启用快照目录，并将命令替换为 `<volume-UUID>`使用要更改的卷的 UUID：

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

您可以通过设置Trident后端配置选项，为新卷默认启用快照目录。 `snapshotDir`到 `true`。现有卷数不受影响。

使用 **KubeVirt** 虚拟机保护数据

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

在恢复操作期间，任何 `VirtualMachineSnapshots`为虚拟机 (VM) 创建的配置文件无法恢复。

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。"配置虚拟化"然后使用以下命令：

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

Trident Protect 24.10.1 及更高版本

从Trident Protect 24.10.1 开始， Trident Protect 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
```

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：

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

ONTAP集群对SnapMirror复制的要求

如果您计划使用SnapMirror复制功能，请确保您的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`：极品飞车
  - `ontap-san`：iSCSI
  - `ontap-san`：FC
  - `ontap-san`：NVMe/TCP（最低要求ONTAP版本 9.15.1）
- 许可证：使用数据保护包的ONTAP SnapMirror异步许可证必须在源 ONTAP 集群和目标ONTAP集群上启用。请参阅 "ONTAP中的SnapMirror许可概述"了解更多信息。

  从ONTAP 9.10.1 开始，所有许可证均以NetApp许可证文件 (NLF) 的形式交付，这是一个可以启用多种功能的单个文件。请参阅"ONTAP One 附带的许可证"了解更多信息。

  仅支持SnapMirror异步保护。

SnapMirror复制的对等连接注意事项

如果您计划使用存储后端对等互连，请确保您的环境满足以下要求：

- 集群和 **SVM**： ONTAP存储后端必须相互连接。请参阅 "集群和SVM对等连接概述"了解更多信息。

  确保两个ONTAP集群之间复制关系中使用的 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

请确保您的 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 读取数据和写入数据，以实现应用程序对象复制。
- 网络连接：配置防火墙规则、存储桶权限和 IP 允许列表，以启用两个集群和 AppVault 之间通过 WAN 进行通信。

许多企业环境在广域网连接中实施严格的防火墙策略。在配置复制之前，请与您的基础架构团队确认这些网络要求。

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 仓库：

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

从私有注册表安装**Trident Protect**

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

步骤

1. 将以下镜像拉取到本地计算机，更新标签，然后将其推送到您的私有镜像仓库：

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

例如：

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

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

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

2. 创建Trident Protect 系统命名空间：

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

3. 登录注册表：

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

4. 创建用于私有注册表身份验证的拉取密钥：

```
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 仓库：

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

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

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

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

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，需要将其安装在用于访问集群的计算机上。根据您的机器使用的是 x64 还是ARM CPU，请按照以下步骤操作。

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/25.06.0/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/25.06.0/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/25.06.0/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/25.06.0/tridentctl-protect-macos-arm64
```

1. 启用插件二进制文件的执行权限：

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

2. 将插件二进制文件复制到 PATH 环境变量中定义的位置。例如， `/usr/bin`或者 `/usr/local/bin`（您可能需要更高的权限）：

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

3. 您也可以选择将插件二进制文件复制到您主目录中的某个位置。在这种情况下，建议确保该位置已添加到您的 PATH 环境变量中：

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

将插件复制到 PATH 环境变量中的某个位置，即可通过键入以下命令来使用该插件。 `tridentctl-protect`或者 `tridentctl protect`从任何地点。

查看**Trident CLI** 插件帮助

您可以使用插件内置的帮助功能来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
```

启用命令自动补全

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** 启用自动补全功能

步骤

1. 下载完成脚本：

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

2. 在您的用户目录下创建一个新目录，用于存放脚本：

```
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`您主目录中的文件：

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

为 **Z shell** 启用自动补全

步骤

1. 下载完成脚本：

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

2. 在您的用户目录下创建一个新目录，用于存放脚本：

```
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`您主目录中的文件：

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

结果

下次登录 shell 时，您可以使用 tridentctl-protect 插件进行命令自动补全。

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. 应用以下值 `resourceLimits.yaml`文件：

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

自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. 在文件中添加 SCC 选项，并根据您的环境需要修改参数。

   以下示例显示了 SCC 选项的参数默认值：

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

下表描述了SCC选项的参数：

参数	描述	默认
create	确定是否可以创建 SCC 资源。仅当 `scc.create`设置为 `true`Helm 安装过程会识别 OpenShift 环境。如果不是在 OpenShift 上运行，或者如果 `scc.create`设置为 `false`不会创建 SCC 资源。	true
name	指定SCC的名称。	三叉戟保护工作
优先事项	确定 SCC 的优先级。优先级较高的SCC会优先于优先级较低的SCC进行评估。	1

3. 应用以下值 `sccconfig.yaml`文件：

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

这将用指定的值替换默认值。 `sccconfig.yaml`文件。

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	类型	描述
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	布尔值	设置为“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	布尔值	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` ）

参数	类型	描述
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. 创建一个配置文件并将其命名为 `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. 填写完之后 `values.yaml`将包含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`用于指定各个参数的标志：

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

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. 在文件中添加 nodeSelector 选项，并修改文件以添加或更改节点标签，从而根据您的环境需要进行限制。例如，以下文件包含默认的操作系统限制，但同时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. 应用以下值 `nodeSelectorConfig.yaml`文件：

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

这将默认限制替换为您在设置中指定的限制。 `nodeSelectorConfig.yaml`文件。

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。

集群管理员始终拥有对默认资源的访问权限。 `trident-protect`命名空间，并且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，您需要创建额外的命名空间，并将资源和应用程序添加到这些命名空间中。

请注意，默认情况下，任何用户都无法创建应用程序数据管理变更请求 (CR)。 `trident-protect`命名空间。您需要在应用程序命名空间中创建应用程序数据管理 CR（最佳实践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 CR）。

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**：需要存储桶凭证数据
- **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

最佳实践是使用基于角色的访问控制 (RBAC) 将对特权对象的访问限制在管理员范围内。

有关基于角色的访问控制 (RBAC) 如何管理对资源和命名空间的访问的更多信息，请参阅…… "Kubernetes RBAC 文档" 。

有关服务帐户的信息，请参阅 "Kubernetes 服务帐户文档"。

示例：管理两组用户的访问权限

例如，一个组织有集群管理员、一组工程用户和一组市场营销用户。集群管理员将完成以下任务，以创建一个环境，其中工程组和市场营销组各自只能访问分配给其各自命名空间的资源。

步骤 **1**：创建命名空间以包含每个组的资源

创建命名空间可以让你从逻辑上分离资源，并更好地控制谁可以访问这些资源。

步骤

1. 为工程组创建一个命名空间：

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

2. 为市场营销组创建命名空间：

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

步骤 **2**：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带有一个默认服务帐户，但您应该为每个用户组创建一个服务帐户，以便将来必要时可以进一步在组之间划分权限。

步骤

1. 为工程团队创建一个服务帐户：

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

2. 为市场营销团队创建一个服务帐户：

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

步骤 **3**：为每个新服务帐户创建一个密钥

服务帐户密钥用于对服务帐户进行身份验证，如果遭到泄露，可以轻松删除并重新创建。

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一个密钥：

```
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
```

步骤 **4**：创建 **RoleBinding** 对象，将 **ClusterRole** 对象绑定到每个新的服务帐户。

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. 将集群角色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. 将集群角色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
```

步骤 **5**：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市场营销资源：

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

步骤 **6**：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，集群管理员需要授予各个用户对 AppVault 对象的访问权限。

步骤

1. 创建并应用 AppVault 和密钥组合的 YAML 文件，以授予用户对 AppVault 的访问权限。例如，以下 CR 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. 创建并应用角色绑定 CR，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. 请确认权限是否正确。

   a. 尝试检索所有命名空间的 AppVault 对象信息：

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

   您应该会看到类似以下内容的输出：

```
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
```

您应该会看到类似以下内容的输出：

```
yes
```

结果

您授予 AppVault 权限的用户应该能够使用授权的 AppVault 对象进行应用程序数据管理操作，并且不应该能够访问分配的命名空间之外的任何资源，或者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。

Alertmanager 是一项服务，它可以接收 Prometheus 等工具发送的警报，并将它们路由到您配置的目标位置。

> 这些步骤中包含的配置和指导仅供参考；您需要根据自己的环境进行自定义。请参阅以下官方文档以获取具体说明和支持：
>
> - "kube-state-metrics 文档"
> - "普罗米修斯文档"
> - "Alertmanager 文档"

步骤 1：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 chart。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 chart 创建一个配置文件（例如， `metrics-config.yaml` ）。您可以根据自身环境自定义以下示例配置：

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

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

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

# Collect metrics from all namespaces
namespaces: ""

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

4. 通过部署 Helm chart 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

您可以按照以下说明安装 Prometheus： "普罗米修斯文档" 。

安装 Alertmanager

您可以按照以下说明安装 Alertmanager： "Alertmanager 文档" 。

步骤 2：配置监控工具以协同工作

安装完监控工具后，需要配置它们以使其协同工作。

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 将警报路由到 Alertmanager。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，并可以向 Alertmanager 发送警报。现在您可以配置哪些条件会触发警报以及警报应该发送到哪里。

步骤 **3**：配置警报和警报目标

配置好工具协同工作后，还需要配置哪些类型的信息会触发警报，以及警报应该发送到哪里。

警报示例：备份失败

以下示例定义了一个关键警报，当备份自定义资源的状态设置为“是”时，该警报将被触发。 `Error`持续5秒或更长时间。您可以自定义此示例以匹配您的环境，并将此 YAML 代码片段包含在您的项目中。 `prometheus.yaml`配置文件：

rules.yaml：定义备份失败的 **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."
```

配置 **Alertmanager** 以将警报发送到其他渠道

您可以配置 Alertmanager，使其将通知发送到其他渠道，例如电子邮件、PagerDuty、Microsoft Teams 或其他通知服务，只需在配置文件中指定相应的配置即可。 `alertmanager.yaml`文件。

以下示例配置 Alertmanager 向 Slack 频道发送通知。要根据您的环境自定义此示例，请替换以下值： `api_url`密钥包含您环境中使用的 Slack webhook 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) 文件并将其命名为（例如， `trident-protect-support-bundle.yaml` ）。
2. 配置以下属性：

   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.triggerType**: (*Required*) 确定支持包是立即生成还是按计划生成。计划的数据包生成时间为世界协调时凌晨 12 点。可能值：
     - 已计划
     - 手动
   - **spec.uploadEnabled**: (可选) 控制生成支持包后是否应将其上传到NetApp支持站点。如果未指定，则默认为 `false`。可能值：
     - true
     - false（默认值）
   - **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。您最早可以指定的日期范围是 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. 填写完之后 `trident-protect-support-bundle.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 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。这 `trigger-type`决定捆绑包是立即创建还是由计划安排决定创建时间，并且可以是 `Manual`或者 `Scheduled`。默认设置是 `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`从输出中读取命令并运行它，将目标参数替换为您首选的本地目录。

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 24.10 版本升级时，升级期间运行的快照可能会失败。此故障不会阻止将来创建快照，无论是手动创建还是计划创建。如果在升级过程中快照失败，您可以手动创建一个新的快照，以确保您的应用程序受到保护。
>
> 为避免潜在的故障，您可以在升级前禁用所有快照计划，并在升级后重新启用它们。但是，这会导致在升级期间错过任何计划的快照。

要升级Trident Protect，请执行以下步骤。

步骤

1. 更新Trident Helm 仓库：

```
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. 运行此命令以将 CRD 的管理权从 `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. 运行此命令以删除 Helm 密钥 `trident-protect-crds`图表：

不要卸载 `trident-protect-crds`使用 Helm 构建图表可能会删除您的 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 100.2506.0 --namespace trident-protect
```

管理和保护应用程序

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

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

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。

数据迁移器存储库密码

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，请使用 **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。

创建数据迁移存储库密码密钥

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。

- 根据您使用的数据传输工具，您只需输入该数据传输工具对应的密码即可。例如，如果您正在使用 Restic，并且将来不打算使用 Kopia，则在创建密钥时，您可以只包含 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`命名空间已被删除，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 **CLI**

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

S3 兼容存储 **IAM** 权限

当您访问与 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，请参阅以下示例： "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。欲了解更多信息，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 身份并将 IAM 角色与服务帐户关联的详细说明，请参阅 "AWS EKS Pod Identity 文档"。

AppVault 配置 使用 EKS Pod Identity 时，请配置您的 AppVault CR。 `useIAM: true`使用标志而不是显式凭据：

```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
```

适用于云提供商的 **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
```

亚马逊 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

以下是每个提供商的 AppVault 定义示例。

AppVault CR 示例

您可以使用以下 CR 示例为每个云提供商创建 AppVault 对象。

- 您可以选择指定一个 Kubernetes secret，其中包含 Restic 和 Kopia 存储库加密的自定义密码。请参阅[数据迁移器存储库密码]了解更多信息。
- 对于 Amazon S3 (AWS) AppVault 对象，您可以选择性地指定 sessionToken，如果您使用单点登录 (SSO) 进行身份验证，这将非常有用。当您为提供程序生成密钥时，将创建此令牌。适用于云提供商的 AppVault 密钥生成示例 。
- 对于 S3 AppVault 对象，您可以选择性地使用以下方式指定出站 S3 流量的出口代理 URL：`spec.providerConfig.S3.proxyURL`钥匙。

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
```

亚马逊 S3 (AWS)

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

对于使用 Pod Identity 和 Kopia 数据移动器的 EKS 环境，您可以移除 `providerCredentials` 部分并添加 `useIAM: true`在 `s3`改为配置。

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 CLI 命令示例为每个提供商创建 AppVault CR。

- 您可以选择指定一个 Kubernetes secret，其中包含 Restic 和 Kopia 存储库加密的自定义密码。请参阅[数据迁移器存储库密码]了解更多信息。
- 对于 S3 AppVault 对象，您可以选择性地使用以下方式指定出站 S3 流量的出口代理 URL：`--proxy-url <ip_address:port>`争论。

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
```

亚马逊 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
```

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，请使用该标志 `--show-paths`。

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

移除 **AppVault**

您可以随时删除 AppVault 对象。

> 不要移除 `finalizers`在删除 AppVault 对象之前，请在 AppVault CR 中输入密钥。如果这样做，可能会导致 AppVault 存储桶中残留数据，集群中出现孤立资源。

开始之前

请确保您已删除要删除的 AppVault 使用的所有快照和备份 CR。

使用 **Kubernetes CLI** 删除 **AppVault**

1. 移除 AppVault 对象，并替换 `appvault-name`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. 移除 AppVault 对象，并替换 `appvault-name`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) 文件并将其命名为（例如， `maria-app.yaml` ）。
   b. 配置以下属性：
      - **metadata.name**: (必填) 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**: (必需) 使用命名空间和标签选择器来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包含在此列表中。标签选择器是可选的，可用于筛选每个指定命名空间内的资源。
      - **spec.includedClusterScopedResources**: (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**: (必需) 指定集群范围资源的 API 组、版本和类型。
        - **labelSelector**: (可选) 根据标签筛选集群范围的资源。
      - **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" 。

如果需要在应用程序创建后应用此注解，可以使用以下命令：

```
kubectl annotate application -n <application CR namespace> <application CR name> protect.trident.netapp.io/skip-vm-freeze="true"
```

+
YAML 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）添加筛选条件，包含或排除带有特定标签的资源：

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：

     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。

       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。

- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

> 当两者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`先运行，然后 `labelSelector`应用于生成的资源。

例如：

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

2. 创建与您的环境相匹配的应用程序变更请求后，应用该变更请求。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的列表，并在示例中所示的参数中，将命名空间和资源包含在应用程序定义中。

   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" 。

   在使用 CLI 创建应用程序时，要指定注解，可以使用以下方法： `--annotation`旗帜。

   - 创建应用程序并使用文件系统冻结行为的全局设置：

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

- 创建应用程序并配置本地应用程序设置以控制文件系统冻结行为：

```
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`用于根据以下条件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。

如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

使用 **CR** 创建快照

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: 要创建快照的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储快照内容（元数据）的 AppVault 的名称。
   - **spec.reclaimPolicy**: (可选) 定义当快照 CR 被删除时，快照的 AppArchive 会发生什么情况。这意味着即使设置为 `Retain`快照将被删除。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. 填写完之后 `trident-protect-snapshot-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建快照

步骤

1. 创建快照，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。

如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

开始之前

确保 AWS 会话令牌过期时间足以满足任何长时间运行的 S3 备份操作。如果在备份操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储备份内容的 AppVault 的名称。
   - **spec.dataMover**: (可选) 指示要用于备份操作的备份工具的字符串。可能的值（区分大小写）：
     - `Restic`
     - `Kopia`（默认）
   - **spec.reclaimPolicy**: (可选) 定义从其声明中释放备份时会发生什么。可能值：
     - `Delete`
     - `Retain`（默认）
   - **spec.snapshotRef**: (可选): 要用作备份源的快照名称。如果未提供，则会创建并备份临时快照。
   - **metadata.annotations.protect.trident.netapp.io/full-backup** : (可选) 此注释用于指定备份是否应为非增量备份。默认情况下，所有备份都是增量的。但是，如果此注释设置为 `true`这样，备份就变成了非增量备份。如果未指定，备份将遵循默认的增量备份设置。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。

     YAML 示例：

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

3. 填写完之后 `trident-protect-backup-cr.yaml`将文件的值正确后，应用 CR：

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

使用命令行创建备份

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>
```

您也可以选择使用 `--full-backup`用于指定备份是否应为非增量备份的标志。默认情况下，所有备份都是增量的。使用此标志后，备份将变为非增量备份。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。

制定数据保护计划

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。

- 您只能通过设置来创建快照计划。 `backupRetention`归零和 `snapshotRetention`取大于零的值。环境 `snapshotRetention`将快照值设为零意味着任何计划备份仍会创建快照，但这些快照是临时的，会在备份完成后立即删除。
- 如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

使用变更请求 **(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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.dataMover**: (可选) 指示要用于备份操作的备份工具的字符串。可能的值（区分大小写）：
     - `Restic`
     - `Kopia`（默认）
   - **spec.applicationRef**: 要备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储备份内容的 AppVault 的名称。
   - **spec.backupRetention**：要保留的备份数量。零表示不应创建备份（仅快照）。
   - **spec.snapshotRetention**：要保留的快照数量。零表示不创建任何快照。
   - **spec.granularity**: 计划运行的频率。可能的值，以及相应的必填字段：
     - `Hourly`（需要您指定） `spec.minute` ）
     - `Daily`（需要您指定） `spec.minute`和 `spec.hour`）
     - `Weekly`（需要您指定） `spec.minute, spec.hour` ， 和 `spec.dayOfWeek`）
     - `Monthly`（需要您指定） `spec.minute, spec.hour` ， 和 `spec.dayOfMonth`）
     - `Custom`
   - **spec.dayOfMonth**：（可选）计划应运行的月份日期（1 - 31）。如果粒度设置为，则此字段为必填项。 `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，则此字段为必填项。 `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，则此字段为必填项。 `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，则此字段为必填项。 `Hourly` ， `Daily` ， `Weekly` ， 或者 `Monthly`。该值必须以字符串形式提供。
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (可选) 此注解用于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。

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

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. 填写完之后 `trident-protect-schedule-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建计划任务

步骤

1. 创建保护计划，将括号中的值替换为您环境中的信息。例如：

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

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

您可以设置 `--full-backup-rule`标记 `always`您可以根据需要进行持续完整备份或自定义备份。例如，如果您选择按天粒度进行备份，则可以指定应在哪些工作日进行完整备份。例如，使用 `--full-backup-rule "Monday,Thursday"`安排每周一和周四进行全面备份。

对于仅快照计划，请设置 `--backup-retention 0`并指定一个大于 `0` 的值 `--snapshot-retention`。

删除快照

删除不再需要的已安排或按需快照。

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。该策略的默认设置为： `Retain`避免意外数据丢失。如果政策不改变 `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

您可以使用命令行来检查正在进行、已完成或已失败的备份操作的状态。

步骤

1. 使用以下命令检索备份操作的状态，将方括号中的值替换为您环境中的信息：

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

启用 **Azure NetApp** 文件 **(ANF)** 操作的备份和还原

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

确保以下事项：

- 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。
- 你有 `azure-netapp-files`已选为存储后端的默认存储类。

展开查看配置步骤

1. 如果 ANF 卷是在升级到Trident 24.10 之前创建的，请在Trident中执行以下操作：

   a. 为每个基于 azure-netapp-files 且与应用程序关联的 PV 启用快照目录：

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

   b. 确认已为每个关联的 PV 启用快照目录：

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

   回复：

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未启用快照目录，则选择常规备份功能，该功能在备份过程中会暂时占用容量池中的空间。在这种情况下，请确保容量池中有足够的空间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。

恢复应用程序

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。

- 恢复应用程序时，为该应用程序配置的所有执行钩子都会随应用程序一起恢复。如果存在恢复后执行钩子，它将作为恢复操作的一部分自动运行。
- 支持从备份恢复到不同的命名空间或恢复到原始命名空间（适用于 qtree 卷）。但是，对于 qtree 卷，不支持从快照恢复到不同的命名空间或恢复到原始命名空间。
- 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"。

从备份还原到不同的命名空间

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不同命名空间不会更改与备份中资源同名的任何资源。要恢复备份中的所有资源，要么删除并重新创建目标命名空间，要么将备份恢复到新的命名空间。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

> 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 "科皮亚文件"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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

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

   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     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）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   > 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.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。

- **resourceMatchers[].version**: (可选) 要筛选的资源版本。
- **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "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. 填写完之后 `trident-protect-backup-restore-cr.yaml`将文件的值正确后，应用 CR：

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

使用 CLI

步骤

1. 将备份还原到不同的命名空间，并将括号中的值替换为您环境中的信息。这 `namespace-mapping`该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，格式如下：`source1:dest1,source2:dest2` 。例如：

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

您可以随时将备份恢复到原始命名空间。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

> 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 "科皮亚文件"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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

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

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

     例如：

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

3. （可选）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   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.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名

称。

- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "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. 填写完之后 `trident-protect-backup-ipr-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI**

步骤

1. 将备份恢复到原始命名空间，并将括号中的值替换为您环境中的信息。这 `backup`参数使用命名空间和备份名称，格式如下 `<namespace>/<name>`。例如：

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

如果原始集群出现问题，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 "科皮亚文件"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。

开始之前

确保满足以下先决条件：

- 目标集群已安装Trident Protect。
- 目标集群可以访问与源集群相同的 AppVault 的存储桶路径，备份就存储在该存储桶中。
- 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 会话令牌的过期时间足以满足任何长时间运行的恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。
  - 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
  - 请参阅 "AWS 文档"有关 AWS 资源凭证的更多信息。

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>
```

> 确保用于应用程序还原的命名空间存在于目标集群上。

2. 查看目标集群中可用 AppVault 的备份内容：

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

运行此命令将显示 AppVault 中可用的备份，包括其来源集群、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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内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 中提到的命令来查看备份内容。

   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     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. 填写完之后 `trident-protect-backup-restore-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI**

1. 使用以下命令恢复应用程序，将括号中的值替换为您环境中的信息。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>
```

从快照还原到不同的命名空间

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，但仅当源存储类和目标存储类使用相同的存储后端时才有效。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。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内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**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     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）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   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.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。

- **resourceMatchers[].version**: (可选) 要筛选的资源版本。
- **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "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. 填写完之后 `trident-protect-snapshot-restore-cr.yaml`将文件的值正确后，应用 CR：

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

使用 CLI

步骤

1. 将快照还原到不同的命名空间，并将括号中的值替换为您环境中的信息。
   - 这 `snapshot`参数使用命名空间和快照名称，格式如下 `<namespace>/<name>`。
   - 这 `namespace-mapping`该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，格式如下： `source1:dest1,source2:dest2` 。

     例如：

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

从快照恢复到原始命名空间

您可以随时将快照恢复到原始命名空间。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内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）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

> 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.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "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. 填写完之后 `trident-protect-snapshot-ipr-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI**

步骤

1. 将快照恢复到原始命名空间，并将括号中的值替换为您环境中的信息。例如：

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

检查还原操作的状态

您可以使用命令行来检查正在进行、已完成或已失败的还原操作的状态。

步骤

1. 使用以下命令检索恢复操作的状态，将方括号中的值替换为您环境中的信息：

```
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。

恢复和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释相匹配。源命名空间中不存在的目标命名空间中的标签或注释将被添加，并且任何已存在的标签或注释都将被覆盖以匹配源命名空间中的值。仅存在于目标命名空间中的标签或注释保持不变。

> 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"。

您可以通过设置 Kubernetes 环境变量来防止目标命名空间中的特定注解被覆盖。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在执行恢复或故障转移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

> 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AutoSupport和命名空间过滤选项"。

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

以下示例展示了源命名空间和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作前后目标命名空间的状态，以及目标命名空间中的注释和标签是如何组合或覆盖的。

在恢复或故障转移操作之前

下表说明了恢复或故障转移操作之前示例源命名空间和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 环境=生产 • 合规性=HIPAA • 名称=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

恢复操作后

下表说明了恢复或故障转移操作后示例目标命名空间的状态。有些键已被添加，有些键已被覆盖，并且 `name` 标签已更新，以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名称=ns-2 • 合规性=HIPAA • 环境=生产 • 角色=数据库

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`属性定义了从源应用程序中存在的存储类到目标集群上新存储类的映射。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。

例子：

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

支持的注释

本节列出了系统中用于配置各种行为的支持注解。如果用户没有明确设置注释，系统将使用默认值。

标注	类型	描述	默认值
protect.trident.netapp.io/data-mover-timeout-sec	string	数据移动器操作允许停止的最长时间（以秒为单位）。	“300”
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	string	Kopia 内容缓存的最大大小限制（以兆字节为单位）。	“1000”

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。

恢复和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释相匹配。源命名空间中不存在的目标命名空间中的标签或注释将被添加，并且任何已存在的标签或注释都将被覆盖以匹配源命名空间中的值。仅存在于目标命名空间中的标签或注释保持不变。

> 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"。

您可以通过设置 Kubernetes 环境变量来防止目标命名空间中的特定注解被覆盖。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在执行恢复或故障转移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

> 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AutoSupport和命名空间过滤选项"。

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

以下示例展示了源命名空间和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作前后目标命名空间的状态，以及目标命名空间中的注释和标签是如何组合或覆盖的。

在恢复或故障转移操作之前

下表说明了恢复或故障转移操作之前示例源命名空间和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 环境=生产 • 合规性=HIPAA • 名称=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

恢复操作后

下表说明了恢复或故障转移操作后示例目标命名空间的状态。有些键已被添加，有些键已被覆盖，并且 `name` 标签已更新，以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名称=ns-2 • 合规性=HIPAA • 环境=生产 • 角色=数据库

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钩子

使用 AppMirror 关系保护应用程序时，在故障转移和反向操作期间，您应该注意与执行钩子相关的特定行为。

- 故障转移期间，执行钩子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，应用程序上会存在执行钩子，这些钩子将在任何相关操作期间执行。
- 在反向同步或反向重同步期间，应用程序上任何现有的执行钩子都会被移除。当源应用程序变为目标应用程序时，这些执行钩子将不再有效，并会被删除以防止其执行。

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

建立复制关系

建立复制关系涉及以下步骤：

- 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 Kubernetes 资源以及持久卷数据）

- 设置拍摄快照的时间

步骤

1. 在源集群上，为源应用程序创建一个 AppVault。根据您的存储提供商，修改示例中的内容。"AppVault 自定义资源"为了适应您的环境：

使用 **CR** 创建 **AppVault**

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

b. 配置以下属性：

- **metadata.name**: (必填) 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**: (必需) 存储使用指定提供程序访问 AppVault 所需的配置。选择存储桶名称以及提供商所需的其他任何详细信息。请记下您选择的值，因为复制关系所需的其他 CR 文件会引用这些值。请参阅"AppVault 自定义资源"例如，AppVault CR 与其他提供商的合作案例。
- **spec.providerCredentials**: (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
  - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示凭据值应来自密钥。
    - **key**: (必填) 要从中选择的有效密钥。
    - **name**: (必填) 包含此字段值的密钥的名称。必须位于同一命名空间中。
  - **spec.providerCredentials.secretAccessKey**: (必需) 用于访问提供程序的访问密钥。 名称 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。
- **spec.providerType**: (必需) 确定备份的提供方；例如， NetApp ONTAP S3、通用 S3 、Google Cloud 或 Microsoft Azure。可能值：
  - AWS
  - 蔚蓝
  - 通用控制协议
  - 通用-s3
  - ontap-s3
  - 存储网格-s3

c. 填写完之后 `trident-protect-appvault-primary-source.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建 **AppVault**

a. 创建 AppVault，并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) 文件并将其命名为（例如， `trident-protect-app-source.yaml` ）。

b. 配置以下属性：

- **metadata.name**: (必填) 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**: (必需) 命名空间及其关联标签的数组。使用命名空间名称，并可选择使用标签缩小命名空间的范围，以指定此处列出的命名空间中存在的资源。应用程序命名空间必须包含在此数组中。

  YAML 示例：

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

c. 填写完之后 `trident-protect-app-source.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建源应用程序

a. 创建源应用程序。例如：

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

3. （可选）在源集群上，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之外，建议创建一个单独的每日快照计划，保留期为 7 天，以在对等ONTAP集群之间保持共同的快照。这样可以确保快照最多保留 7 天，但保留期限可以根据用户需求进行自定义。

如果发生故障转移，系统可以使用这些快照最多 7 天进行逆向操作。这种方法使得逆向过程更快、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，则无需制定其他计划。

使用 **CR** 创建复制计划

a. 为源应用程序创建复制计划：

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

   ii. 配置以下属性：

   - **metadata.name**: (必填) 计划自定义资源的名称。
   - **spec.appVaultRef**: (必需) 此值必须与源应用程序的 AppVault 的 metadata.name 字段匹配。
   - **spec.applicationRef**: (必需) 此值必须与源应用程序 CR 的 metadata.name 字段匹配。
   - **spec.backupRetention**: (*Required*) 此字段为必填项，其值必须设置为 0。
   - **spec.enabled**：必须设置为 true。
   - **spec.granularity**：必须设置为 `Custom`。
   - **spec.recurrenceRule**: 定义 UTC 时间的开始日期和重复间隔。
   - **spec.snapshotRetention**：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. 填写完之后 `trident-protect-schedule.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建复制计划

a. 创建复制计划，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>
```

例子：

```
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。根据您的存储提供商，修改示例中的内容。"AppVault 自定义资源"为了适应您的环境：
   a. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-appvault-secondary-destination.yaml` ）。
   b. 配置以下属性：
      - **metadata.name**: (必填) 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**: (必需) 存储使用指定提供程序访问 AppVault 所需的配置。选择一个 `bucketName`以及其他任何您需要提供给服务提供商的详细信息。请记下您选择的值，因为复制关系所需的其他 CR 文件会引用这些值。请参阅"AppVault 自定义资源"例如，AppVault CR 与其他提供商的合作案例。
      - **spec.providerCredentials**: (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
        - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示凭据值应来自密钥。
          - **key**: (必填) 要从中选择的有效密钥。
          - **name**: (必填) 包含此字段值的密钥的名称。必须位于同一命名空间中。
        - **spec.providerCredentials.secretAccessKey**: (必需) 用于访问提供程序的访问密钥。 名称 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。

- **spec.providerType**: (必需) 确定备份的提供方；例如， NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能值：
  - AWS
  - 蔚蓝
  - 通用控制协议
  - 通用-s3
  - ontap-s3
  - 存储网格-s3

c. 填写完之后 `trident-protect-appvault-secondary-destination.yaml`将文件的值正确后，应用 CR：

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

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

使用 **CR** 创建 **AppMirrorRelationship**

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

b. 配置以下属性：

- **metadata.name:**（必填）AppMirrorRelationship 自定义资源的名称。
- **spec.destinationAppVaultRef**: (必需) 此值必须与目标集群上目标应用程序的 AppVault 名称匹配。
- **spec.namespaceMapping**: (必需) 目标命名空间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**: (必需) 此值必须与源应用程序的 AppVault 名称匹配。
- **spec.sourceApplicationName**: (必需) 此值必须与您在源应用程序 CR 中定义的源应用程序的名称匹配。
- **spec.sourceApplicationUID**: （必需）此值必须与您在源应用程序 CR 中定义的源应用程序的 UID 匹配。
- **spec.storageClassName**: (可选) 选择集群上有效的存储类的名称。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.recurrenceRule**: 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. 填写完之后 `trident-protect-relationship.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建 **AppMirrorRelationship**

a. 创建并应用 AppMirrorRelationship 对象，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>
```

例子：

```
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 AppMirrorRelationship CR 文件（例如， `trident-protect-relationship.yaml` ）并将 **spec.desiredState** 的值更改为 `Promoted`。
2. 保存CR文件。

3. 应用 CR：

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

4. （可选）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
```

重新同步失败的复制关系

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

该过程会在重新建立复制之前停止目标集群上的应用程序。

故障转移期间写入目标应用程序的任何数据都将丢失。

步骤

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

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

5. 如果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。任何残留的计划都会导致卷快照失败。

反向重新同步失败的复制关系

当您反向同步故障转移复制关系时，目标应用程序将变为源应用程序，而源应用程序将变为目标应用程序。故障转移期间对目标应用程序所做的更改将被保留。

步骤

1. 在原始目标集群上，删除 AppMirrorRelationship CR。这导致目的地变成了出发地。如果新目标集群上还有任何剩余的保护计划，请将其删除。
2. 通过将最初用于建立关系的 CR 文件应用到相反的集群来建立复制关系。
3. 确保新目标（原始源集群）配置了两个 AppVault CR。
4. 在相反的集群上建立复制关系，配置反向的值。

反向应用程序复制方向

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，你交换了源地址和目标地址。

步骤

1. 在源集群上，创建关机快照：

使用 **CR** 创建关机快照

a. 禁用源应用程序的保护策略计划。

b. 创建 ShutdownSnapshot CR 文件：

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

ii. 配置以下属性：

- **metadata.name**: (必填) 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 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. 填写完之后 `trident-protect-shutdownsnapshot.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 创建关机快照

a. 创建关机快照，将括号中的值替换为您环境中的信息。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，关机快照完成后，获取关机快照的状态：

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

3. 在源集群上，使用以下命令查找 **shutdownsnapshot.status.appArchivePath** 的值，并记录文件路径的最后一部分（也称为基本名称；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. 从新的目标集群到新的源集群执行故障转移，并进行以下更改：

> 在故障转移流程的第 2 步中，包括： `spec.promotedSnapshot`在 AppMirrorRelationship CR 文件中，将该字段的值设置为您在上面的步骤 3 中记录的基本名称。

5. 执行反向重新同步步骤[反向重新同步失败的复制关系]。
6. 在新源集群上启用保护计划。

结果

由于反向复制，会发生以下操作：

- 对原始源应用程序的 Kubernetes 资源进行快照。
- 通过删除应用程序的 Kubernetes 资源（保留 PVC 和 PV），优雅地停止原始源应用程序的 pod。
- 在 pod 关闭后，会对应用程序的卷进行快照并进行复制。
- SnapMirror关系已断开，目标卷已准备好进行读/写操作。
- 该应用程序的 Kubernetes 资源是从关闭前的快照中恢复的，使用的是在原始源应用程序关闭后复制的卷数据。
- 复制过程以相反的方向重新建立。

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。

该过程从已完成故障转移至目标位置的关系开始，并涉及以下步骤：

- 从故障转移状态开始。
- 反向重新同步复制关系。

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

- 反转复制方向。

步骤

1. 执行[反向重新同步失败的复制关系]步骤。
2. 执行[反向应用程序复制方向]步骤。

删除复制关系

您可以随时删除复制关系。删除应用程序复制关系后，将生成两个彼此独立的应用程序，它们之间没有任何关系。

步骤

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

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

使用Trident Protect 迁移应用程序

您可以通过恢复备份数据在集群之间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钩子都会随应用程序一起迁移。如果存在恢复后执行钩子，它将作为恢复操作的一部分自动运行。

备份和恢复操作

针对以下场景，您可以自动执行特定的备份和恢复任务，以执行备份和恢复操作。

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，然后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"。
   b. "创建备份"。
2. 在同一集群上，根据您创建的是快照还是备份，执行以下操作之一：
   a. "从快照恢复数据"。
   b. "从备份中恢复数据"。

克隆到不同的集群

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。

您可以使用以下方法在不同的集群之间复制应用程序"SnapMirror 复制"。

步骤

1. "创建备份"。
2. 确保目标集群上已配置包含备份的对象存储桶的 AppVault CR。
3. 在目标集群上，"从备份中恢复数据" 。

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（不包括恢复 CR 中的密钥）：

```
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** 恢复快照

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. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储快照内容的路径。您可以使用以下命令查找此路径：

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

   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     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. 如果您需要仅选择要恢复的应用程序的某些资源，则可以添加筛选条件，以包含或排除带有特定标签的资源：
   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `include or exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "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. 填写完之后 `trident-protect-snapshot-restore-cr.yaml`将文件的值正确后，应用 CR：

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

使用 **CLI** 恢复快照

步骤

1. 将快照还原到不同的命名空间，并将括号中的值替换为您环境中的信息。

   - 这 `snapshot`参数使用命名空间和快照名称，格式如下 `<namespace>/<name>`。
   - 这 `namespace-mapping`该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，格式如下： `source1:dest1,source2:dest2` 。

     例如：

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

管理Trident Protect 执行钩子

执行钩子是一种自定义操作，您可以将其配置为与受管应用程序的数据保护操作一起运行。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。这确保了应用程序一致的快照。

执行钩子的类型

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
- 恢复后
- 故障转移后

执行顺序

当运行数据保护操作时，执行挂钩事件按以下顺序发生：

1. 任何适用的自定义预操作执行挂钩都在适当的容器上运行。您可以根据需要创建和运行任意数量的自定义预操作挂钩，但这些挂钩在操作之前的执行顺序既无法保证也无法配置。
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冻。
5. 任何适用的自定义后操作执行挂钩都在适当的容器上运行。您可以根据需要创建和运行任意数量的自定义后操作挂钩，但操作后这些挂钩的执行顺序既无法保证也无法配置。

如果您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，不同类型的钩子的执行顺序是有保证的。例如，以下是具有所有不同类型钩子的配置的执行顺序：

1. 快照前钩子执行
2. 快照后钩子执行
3. 执行备份前挂钩
4. 执行备份后钩子

前面的顺序示例仅适用于运行不使用现有快照的备份时。

在生产环境中启用执行挂钩脚本之前，您应该始终对其进行测试。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，测试生成的快照和备份以确保它们一致。您可以通过将应用程序克隆到临时命名空间、恢复快照或备份，然后测试应用程序来执行此操作。

如果快照前执行钩子添加、更改或删除 Kubernetes 资源，则这些更改将包含在快照或备份以及任何后续恢复操作中。

关于自定义执行钩子的重要说明

在为您的应用程序规划执行挂钩时，请考虑以下事项。

- 执行钩子必须使用脚本来执行操作。许多执行钩子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 96KB。
- 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。

由于执行钩子通常会减少或完全禁用其所针对的应用程序的功能，因此您应该始终尝试尽量减少自定义执行钩子的运行时间。如果您启动带有相关执行挂钩的备份或快照操作，但随后取消它，则如果备份或快照操作已经开始，则仍允许挂钩运行。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。

执行钩子过滤器

当您为应用程序添加或编辑执行挂钩时，您可以向执行挂钩添加过滤器来管理该挂钩将匹配哪些容器。过滤器对于在所有容器上使用相同容器镜像但可能将每个镜像用于不同目的的应用程序（例如 Elasticsearch）很有用。过滤器允许您创建执行挂钩在某些（但不一定是所有）相同的容器上运行的场景。如果为单个执行挂钩创建多个过滤器，它们将通过逻辑 AND 运算符组合在一起。每个执行挂钩最多可以有 10 个活动过滤器。

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"。

如果将命名空间过滤器添加到在恢复或克隆操作后运行的执行挂钩，并且恢复或克隆源和目标位于不同的命名空间中，则命名空间过滤器仅适用于目标命名空间。

执行钩子示例

访问 "NetApp Verda GitHub 项目"下载流行应用程序（如 Apache Cassandra 和 Elasticsearch）的真实执行挂钩。您还可以查看示例并获得构建您自己的自定义执行挂钩的想法。

创建执行钩子

您可以使用以下方法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**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要运行执行钩子的应用程序的 Kubernetes 名称。
   - **spec.stage**: (*Required*) 一个字符串，指示执行钩子应该在操作的哪个阶段运行。可能值：
     - 预
     - 发布
   - **spec.action**: (*Required*) 一个字符串，指示执行钩子将采取什么操作，假设指定的任何执行钩子过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.enabled**: (可选) 指示此执行钩子是否已启用或禁用。如果未指定，则默认值为 true。
   - **spec.hookSource**: (必需) 包含 base64 编码的 hook 脚本的字符串。
   - **spec.timeout**: (可选) 定义执行钩子允许运行的分钟数的数字。最小值为 1 分钟，如果未指定，则默认值为 25 分钟。
   - **spec.arguments**: (可选) 一个 YAML 列表，用于指定执行钩子的参数。
   - **spec.matchingCriteria**: (可选) 一个可选的条件键值对列表，每个键值对构成一个执行钩子过滤器。每个执行钩子最多可以添加 10 个过滤器。
   - **spec.matchingCriteria.type**: (可选) 用于标识执行钩过滤器类型的字符串。可能值：
     - 容器图像
     - 容器名称
     - Pod名称
     - PodLabel
     - 命名空间名称
   - **spec.matchingCriteria.value**: (可选) 用于标识执行钩过滤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. 在用正确的值填充 CR 文件后，应用 CR：

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

使用 **CLI**

步骤

1. 创建执行钩子，将括号中的值替换为您环境中的信息。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>
```

手动运行执行钩子

您可以手动运行执行钩子进行测试，或者在失败后需要手动重新运行钩子时也可以这样做。您需要拥有所有者、管理员或成员权限才能手动运行执行钩子。

手动运行执行钩子包含两个基本步骤：

1. 创建资源备份，该备份会收集资源并创建它们的备份，从而确定钩子函数的运行位置。
2. 针对备份运行执行钩子

步骤 **1**：创建资源备份

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-resource-backup.yaml`。
2. 配置以下属性以匹配您的Trident Protect 环境和集群配置：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要为其创建资源备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内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. 在用正确的值填充 CR 文件后，应用 CR：

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

使用 **CLI**

步骤

1. 创建备份，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。您可以重复使用此示例命令，直到操作完成：

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

3. 确认备份是否成功：

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

步骤 **2**：运行执行钩子

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-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**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 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**: (必需) 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，指示执行钩子将采取什么操作，假设指定的任何执行钩子过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.stage**: (*Required*) 一个字符串，指示执行钩子应该在操作的哪个阶段运行。这次钩子运行不会在任何其他阶段运行钩子。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. 在用正确的值填充 CR 文件后，应用 CR：

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

使用 **CLI**

步骤

1. 创建手动执行钩子运行请求：

```
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. 检查执行钩子运行状态。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 exechooksrun 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
```