管理Trident Protect

Trident

NetApp
January 15, 2026

目录

管理Trident Protect . . . 1
管理Trident Protect 授权和访问控制 . . . 1
示例：管理两组用户的访问权限 . . . 1
监控Trident保护资源 . . . 7
步骤 1：安装监控工具 . . . 7
步骤 2：配置监控工具以协同工作 . . . 10
步骤 3：配置警报和警报目标 . . . 11
生成Trident Protect 支持包 . . . 12
监视和检索支持包 . . . 14
升级Trident保护 . . . 14

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
```