管理 Trident Protect

Trident

NetApp
July 01, 2026

目录

管理 Trident Protect

管理 Trident Protect 授权和访问控制

Trident Protect 使用基于角色的访问控制 (RBAC) 的 Kubernetes 模型。默认情况下，Trident Protect 提供单个系统命名空间及其关联的默认服务帐户。如果您的组织有很多用户或特定的安全需求，您可以使用 Trident Protect 的 RBAC 功能对资源和命名空间的访问进行更精细的控制。

集群管理员始终可以访问默认 `trident-protect`命名空间中的资源，也可以访问所有其他命名空间中的资源。要控制对资源和应用程序的访问，需要创建其他命名空间，并向这些命名空间中添加资源和应用程序。

请注意，没有用户可以在默认 `trident-protect`命名空间中创建应用程序数据管理 CR。您需要在应用程序命名空间中创建应用程序数据管理 CR（作为最佳实践，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）。

> 只有管理员应有权访问特权 Trident Protect 自定义资源对象，其中包括：
>
> - **AppVault**：需要存储区凭据数据
> - **AutoSupportBundle**：收集指标、日志和其他敏感 Trident Protect 数据
> - **AutoSupportBundleSchedule**：管理日志收集计划
>
> 作为最佳做法，请使用 RBAC 来限制管理员对特权对象的访问。

有关 RBAC 如何规范对资源和命名空间的访问的详细信息，请参见 "Kubernetes RBAC 文档"。

有关服务帐户的详细信息，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，一个组织有一个集群管理员、一组工程用户和一组营销用户。集群管理员将完成以下任务，以创建一个环境，其中工程组和营销组各自只能访问分配给各自命名空间的资源。

步骤 **1**：创建一个命名空间以包含每个组的资源

创建命名空间可以在逻辑上分离资源，并更好地控制谁有权访问这些资源。

步骤

1. 为工程组创建命名空间：

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

2. 为营销组创建命名空间：

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

步骤 **2**：创建新的服务帐户以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，以便将来在必要时可以在组之间进一步划分权限。

步骤

1. 为工程组创建服务帐户：

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

2. 为营销组创建服务帐户：

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

步骤 **3**：为每个新服务帐户创建密码

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密钥：

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

2. 为营销服务帐户创建密钥：

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

步骤 **4**：创建一个 **RoleBinding** 对象以将此 **ClusterRole** 对象绑定到每个新服务帐户

当你安装 Trident Protect 时，会创建一个默认的 ClusterRole 对象。你可以通过创建并应用一个 RoleBinding 对象，将此 ClusterRole 绑定到服务账户。

步骤

1. 将 ClusterRole 绑定到工程服务帐户：

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

2. 将 ClusterRole 绑定到营销服务帐户：

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

第 **5** 步：测试权限

测试权限是否正确。

步骤

1. 确认工程用户可以访问工程资源：

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

2. 确认工程用户无法访问营销资源：

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

第 **6** 步：授予 **AppVault** 对象访问权限

要执行备份和快照等数据管理任务，集群管理员需要将 AppVault 对象的访问权限授予单个用户。

步骤

1. 创建并应用 AppVault 和密码组合 YAML 文件，该文件授予用户对 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 Role CR，使集群管理员能够授予对命名空间中特定资源的访问权限。例如：

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

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

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

4. 验证权限是否正确。

   a. 尝试检索所有命名空间的 AppVault 对象信息：

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

   此时将显示与以下内容类似的输出：

```
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
```

此时将显示与以下内容类似的输出：

```
yes
```

结果

您授予 AppVault 权限的用户应能够使用授权的 AppVault 对象进行应用程序数据管理操作，并且不应能够访问分配的命名空间之外的任何资源或创建他们无权访问的新资源。

监控 Trident Protect 资源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 开源工具来监控受 Trident Protect 保护的资源的运行状况。

kube-state-metrics 服务从 Kubernetes API 通信中生成指标。将它与 Trident Protect 一起使用可提供有关环境中资源状态的有用信息。

Prometheus 是一个工具包，可以提取 kube-state-metrics 生成的数据，并将其呈现为有关这些对象的易读信息。kube-state-metrics 和 Prometheus 共同为您提供了一种方法，可以监控您使用 Trident Protect 管理的资源的运行状况和状态。

Alertmanager 是一项服务，可接收 Prometheus 等工具发送的警报，并将其路由到您配置的目标。

> 这些步骤中包含的配置和指导只是示例；您需要自定义它们以匹配您的环境。有关具体说明和支持，请参阅以下官方文档：
>
> - "kube-state-metrics 文档"
> - "Prometheus 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，需要安装和配置 kube-state-metrics、Prometheus 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 应用于集群：

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

3. 为 Helm 图表创建配置文件（例如， `metrics-config.yaml`）。您可以自定义以下示例配置以匹配您的环境：

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 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 "kube-state-metrics 自定义资源文档"中的说明配置 kube-state-metrics，为 Trident Protect 使用的自定义资源生成指标。

安装 **Prometheus**

您可以按照 "Prometheus 文档"中的说明安装 Prometheus。

安装 **Alertmanager**

您可以按照 "Alertmanager 文档"中的说明安装 Alertmanager。

步骤 **2**：配置监控工具以协同工作

安装监控工具后，需要将其配置为协同工作。

步骤

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

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

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

2. 配置 Prometheus 以将警报路由到 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，并向 Alertmanager 发送警报。您现在可以配置触发警报的条件以及应将警报发送到何处。

步骤 3：配置警报和警报目标

将工具配置为协同工作后，需要配置触发警报的信息类型以及应将警报发送到何处。

警报示例：备份失败

以下示例定义了在备份自定义资源的状态设置为 `Error` 5 秒或更长时间时触发的关键警报。您可以自定义此示例以匹配您的环境，并在 `prometheus.yaml`配置文件中包含此 YAML 代码段：

rules.yaml：为失败的备份定义 **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** 以向其他渠道发送警报

您可以通过在 `alertmanager.yaml` 文件中指定相应的配置，将 Alertmanager 配置为向其他渠道（如电子邮件、PagerDuty、Microsoft Teams 或其他通知服务）发送通知。

以下示例将 Alertmanager 配置为向 Slack 通道发送通知。要根据您的环境自定义此示例，请将 `api_url` 键的值替换为环境中使用的 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 使管理员能够生成捆绑包，其中包括对 NetApp Support 有用的信息，包括有关所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) 文件将支持捆绑包上传到 NetApp 支持站点 (NSS)。

使用 **CR** 创建支持包

步骤

1. 创建自定义资源 (CR) 文件并将其命名（例如， `trident-protect-support-bundle.yaml`）。
2. 配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.triggerType**: (*Required*) 确定支持包是立即生成还是计划生成。计划包生成发生在 UTC 时间凌晨 12 点。可能的值：
     - 已计划
     - 手动
   - **spec.uploadEnabled**: (*Optional*) 控制是否应在生成支持捆绑包后将其上传到 NetApp 支持站点。如果未指定，则默认为 `false`。可能的值：
     - true
     - false（默认）
   - **spec.dataWindowStart**: (*Optional*) RFC 3339 格式的日期字符串，指定支持包中包含数据的窗口应开始的日期和时间。如果未指定，则默认为 24 小时前。您可以指定的最早窗口日期为 7 天前。

     示例 YAML：

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

3. 使用正确的值填充 `trident-protect-support-bundle.yaml`文件后，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，用环境中的信息替换括号中的值。 `trigger-type`确定是否立即创建捆绑包，或者创建时间是否由计划决定，并且可以是 `Manual`或 `Scheduled`。默认设置为 `Manual`。

   例如：

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

监控和检索支持包

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. 将支持包检索到本地系统。从已完成的 AutoSupport 包中获取 copy 命令：

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

从输出中查找 `kubectl cp`命令并运行它，将 destination 参数替换为首选的本地目录。

升级 Trident Protect

您可以将 Trident Protect 升级到最新版本，以利用新功能或错误修复。

- 从版本 24.10 升级时，升级期间运行的快照可能会失败。此故障不会阻止创建未来的快照，无论是手动还是计划的快照。如果快照在升级过程中失败，您可以手动创建新的快照，以确保您的应用程序受到保护。

  为了避免潜在的故障，您可以在升级之前禁用所有快照计划，然后在升级后重新启用它们。但是，这会导致在升级期间丢失任何计划的快照。

- 对于私有注册表安装，请确保您的私有注册表中提供了目标版本所需的 Helm 图表和图像，并验证您的自定义 Helm 值与新图表版本兼容。有关详细信息，请参阅 "从私有注册表安装 Trident Protect"。

步骤 1：选择一个版本

Trident Protect 版本遵循基于日期的 `YY.MM`命名约定，其中"YY"是年份的最后两位数字，"MM"是月份。点版本遵循 `YY.MM.X`约定，其中"X"是补丁级别。您将根据要升级的版本选择要升级到的版本。

- 您可以直接升级到安装版本的四个版本窗口内的任何目标版本。例如，您可以直接从 24.10（或任何 24.10 dot 版本）升级到 25.10。

- 如果您要从四版本窗口之外的版本进行升级，请执行多步骤升级。使用您要从"早期版本"升级的升级说明升级到适合四发布窗口的最新版本。例如，如果您运行的是 24.10 并希望升级到 26.02：
  a. 首次从 24.10 升级到 25.02。
  b. 然后从 25.02 升级到 26.02。

步骤 2：升级 Trident Protect

要升级 Trident Protect，请执行以下步骤。

步骤

1. 更新 Trident Helm 存储库：

```
helm repo update
```

2. 升级 Trident Protect CRD：

> 如果您要从 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. 运行此命令可将 CRD 的管理从 `trident-protect-crds`切换到 `trident-protect`：

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

   b. 运行此命令删除 `trident-protect-crds` 图表的 Helm 密钥：

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

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

3. 升级 Trident Protect：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2602.0 --namespace trident-protect
```

> 您可以通过在 upgrade 命令中添加 `--set logLevel=debug`来配置升级过程中的日志级别。默认日志记录级别为 `warn`。建议将调试日志记录用于故障排除，因为它有助于 NetApp 支持诊断问题，而无需更改日志级别或重现问题。