



使用 **Trident** 操作员安装 Trident

NetApp
October 29, 2026

目录

使用 Trident 操作员安装	1
手动部署 Trident 操作员（标准模式）	1
有关 Trident 26.06 的关键信息	1
手动部署 Trident 操作员并安装 Trident	1
验证安装	4
手动部署 Trident 操作员（离线模式）	5
有关 Trident 的重要信息	6
手动部署 Trident 操作员并安装 Trident	6
验证安装	9
使用 Helm 部署 Trident 操作员（标准模式）	11
有关 Trident 26.06 的关键信息	11
部署 Trident 操作员并使用 Helm 安装 Trident	11
在安装过程中传递配置数据	12
配置选项	12
使用 Helm 部署 Trident 操作员（离线模式）	18
有关 Trident 26.06 的关键信息	18
部署 Trident 操作员并使用 Helm 安装 Trident	18
在安装过程中传递配置数据	19
配置选项	20
定制 Trident 操作员安装	25
了解控制器 pod 和节点 pod	25
配置选项	25
示例配置	29

使用 Trident 操作员安装

手动部署 Trident 操作员（标准模式）

您可以手动部署 Trident 操作员来安装 Trident。此过程适用于 Trident 所需的容器映像未存储在私有注册表中的安装。如果确实有专用映像注册表，请使用 ["脱机部署流程"](#)。

有关 Trident 26.06 的关键信息

您必须阅读以下有关 Trident 的重要信息。

▼ 有关 Trident 的重要信息

- Trident 现在支持 Kubernetes 1.37。在升级 Kubernetes 之前，请先升级 Trident。
- Trident 严格执行在 SAN 环境中使用多路径配置，在 multipath.conf 文件中的建议值为 `find_multipaths: no`。

使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes`或`find_multipaths: smart`值将导致挂载失败。自 21.07 版本发布以来，Trident 建议使用`find_multipaths: no`。

手动部署 Trident 操作员并安装 Trident

请检查 ["安装概述"](#) 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

在开始安装之前，请登录到 Linux 主机并验证其是否正在管理工作，["支持的 Kubernetes 集群"](#)以及您是否具有必要的权限。



使用 OpenShift 时，在后面的所有示例中使用 `oc`而不是`kubectl`，并首先通过运行 ``oc login -u system:admin`或`oc login -u kube-admin`以 system:admin 身份登录。`

▼ Details

1. 验证您的 Kubernetes 版本：

```
kubectl version
```

2. 验证集群管理员权限：

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. 验证是否可以启动使用 Docker Hub 镜像的 Pod，并通过 Pod 网络到达您的存储系统：

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
ping <management IP>
```

步骤 1: 下载 Trident 安装程序包

Trident 安装程序包包含部署 Trident 操作员和安装 Trident 所需的一切。从 ["GitHub 上的 Assets 部分"](#) 下载并提取最新版本的 Trident 安装程序。

```
wget https://github.com/NetApp/trident/releases/download/v26.06.0/trident-
installer-26.06.0.tar.gz
tar -xf trident-installer-26.06.0.tar.gz
cd trident-installer
```

步骤 2: 创建 TridentOrchestrator CRD

创建 TridentOrchestrator Custom Resource Definition (CRD)。您稍后将创建 TridentOrchestrator Custom Resource。在 `deploy/crds` 中使用适当的 CRD YAML 版本来创建 `TridentOrchestrator` CRD。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

步骤 3: 部署 Trident 操作员

Trident 安装程序提供了一个捆绑文件，可用于安装操作员和创建相关对象。捆绑文件是使用默认配置部署操作员和安装 Trident 的简便方法。

- 对于运行 Kubernetes 1.24 的集群，请使用 `bundle_pre_1_25.yaml`。
- 对于运行 Kubernetes 1.25 或更高版本的集群，请使用 `bundle_post_1_25.yaml`。

开始之前

- 默认情况下，Trident 安装程序在 `trident` 命名空间中部署运算符。如果 `trident` 命名空间不存在，请使用以下方法创建它：

```
kubectl apply -f deploy/namespace.yaml
```

- 要在 `trident` 命名空间以外的命名空间中部署运算符，请更新 `serviceaccount.yaml`、`clusterrolebinding.yaml` 和 `operator.yaml`，并使用 `kustomization.yaml` 生成捆绑包文件。
 - a. 使用以下命令创建 `kustomization.yaml`，其中 `<bundle.yaml>` 是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml`，基于您的 Kubernetes 版本。

```
cp deploy/kustomization_<bundle.yaml> deploy/kustomization.yaml
```

- b. 使用以下命令编译捆绑包，其中 `<bundle.yaml>` 是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml`，具体取决于您的 Kubernetes 版本。

```
kubectl kustomize deploy/ > deploy/<bundle.yaml>
```

步骤

1. 创建资源并部署 operator:

```
kubectl create -f deploy/<bundle.yaml>
```

2. 验证是否已创建 operator、deployment 和 replicaset。

```
kubectl get all -n <operator-namespace>
```



Kubernetes 集群中应该只有*一个实例*的运算符。请勿创建 Trident 运算符的多个部署。

步骤 4: 创建 `TridentOrchestrator` 并安装 Trident

现在，您可以创建 `TridentOrchestrator` 并安装 Trident。或者，您可以["定制您的 Trident 安装"](#)使用 `TridentOrchestrator` 规范中的属性。

```
kubectl create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

kubectl describe torc trident

Name:          trident
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Spec:
  Debug:      true
  Namespace:  trident
  nodePrep:
    - iscsi
Status:
```

```

Current Installation Params:
  IPv6:                                false
  Autosupport Hostname:
  Autosupport Image:                   netapp/trident-autosupport:26.06
  Autosupport Proxy:
  Autosupport Serial Number:
  Debug:                               true
  Image Pull Secrets:
  Image Registry:
  k8sTimeout:                          30
  Kubelet Dir:                         /var/lib/kubelet
  Log Format:                           text
  Silence Autosupport:                 false
  Trident Image:                       netapp/trident:26.06.0
Message:                               Trident installed Namespace:
trident
  Status:                              Installed
  Version:                             v26.06.0
Events:
  Type Reason Age From Message ---- -Normal
  Installing 74s trident-operator.netapp.io Installing Trident Normal
  Installed 67s trident-operator.netapp.io Trident installed

```

验证安装

有几种方法可以验证您的安装。

使用 **TridentOrchestrator** 状态

`TridentOrchestrator` 的状态指示安装是否成功，并显示已安装的 Trident 版本。在安装过程中，`TridentOrchestrator` 的状态从 `Installing` 更改为 `Installed`。如果您观察到 `Failed` 状态，并且操作员无法自行恢复，[link:../troubleshooting.html\["检查日志"\]](#)。

状态	说明
安装	操作员正在使用此 TridentOrchestrator CR 安装 Trident。
已安装	Trident 已成功安装。
卸载	操作员正在卸载 Trident，因为 <code>spec.uninstall=true</code> 。
已卸载	Trident 已卸载。

状态	说明
失败	操作员无法安装、修补、更新或卸载 Trident；操作员将自动尝试从此状态恢复。如果此状态持续存在，则需要故障排除。
正在更新	操作员正在更新现有安装。
错误	TridentOrchestrator 未使用。另一个已存在。

使用 pod 创建状态

您可以通过查看已创建的 Pod 的状态来确认 Trident 安装是否已完成：

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
AGE			
trident-controller-7d466bf5c7-v4cpw	6/6	Running	0
1m			
trident-node-linux-mr6zc	2/2	Running	0
1m			
trident-node-linux-xrp7w	2/2	Running	0
1m			
trident-node-linux-zh2jt	2/2	Running	0
1m			
trident-operator-766f7b8658-ldzsv	1/1	Running	0
3m			

使用 tridentctl

您可以使用 `tridentctl` 检查已安装的 Trident 版本。

```
./tridentctl -n trident version
```

```
+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+
| 26.06.0       | 26.06.0       |
+-----+
```

手动部署 Trident 操作员（离线模式）

您可以手动部署 Trident 操作员来安装 Trident。此过程适用于 Trident 所需的容器映像存储在私有注册表中的安装。如果您没有私有映像注册表，请使用 ["标准部署流程"](#)。

有关 Trident 的重要信息

您必须阅读以下有关 **Trident** 的重要信息。

▼ 有关 Trident 的重要信息

- Trident 现在支持 Kubernetes 1.37。在升级 Kubernetes 之前，请先升级 Trident。
- Trident 严格执行在 SAN 环境中使用多路径配置，在 multipath.conf 文件中的建议值为 `find_multipaths: no`。

使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes`或`find_multipaths: smart`值将导致挂载失败。自 21.07 版本发布以来，Trident 建议使用`find_multipaths: no。`

手动部署 Trident 操作员并安装 Trident

请检查 ["安装概述"](#) 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

登录到 Linux 主机并验证其是否正在管理工作和["支持的 Kubernetes 集群"](#)并且您具有必要的权限。



使用 OpenShift 时，在后面的所有示例中使用 `oc`而不是`kubectl，并首先通过运行 oc login -u system:admin`或`oc login -u kube-admin`以 system:admin 身份登录。`

▼ Details

1. 验证您的 Kubernetes 版本：

```
kubectl version
```

2. 验证集群管理员权限：

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. 验证是否可以启动使用 Docker Hub 镜像的 Pod，并通过 Pod 网络到达您的存储系统：

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
ping <management IP>
```

步骤 1：下载 Trident 安装程序包

Trident 安装程序包包含部署 Trident 操作员和安装 Trident 所需的一切。从 ["GitHub 上的 Assets 部分"](#) 下载并提取最新版本的 Trident 安装程序。

```
wget https://github.com/NetApp/trident/releases/download/v6.0/trident-
installer-26.06.0.tar.gz
tar -xf trident-installer-26.06.0.tar.gz
cd trident-installer
```

步骤 2: 创建 **TridentOrchestrator CRD**

创建 **TridentOrchestrator Custom Resource Definition (CRD)**。您稍后将创建 **TridentOrchestrator Custom Resources**。在 `deploy/crds` 中使用适当的 CRD YAML 版本来创建 `TridentOrchestrator CRD`：

```
kubectl create -f deploy/crds/<VERSION>.yaml
```

步骤 3: 更新 **operator** 中的注册表位置

在 `/deploy/operator.yaml` 中，更新 `image: docker.io/netapp/trident-operator:26.06.0` 以反映图像注册表的位置。您的 **"Trident 和 CSI 镜像"** 可以位于一个注册表或不同的注册表中，但所有 CSI 映像必须位于同一注册表中。例如：

- `image: <your-registry>/trident-operator:26.06.0` 如果您的镜像都位于一个注册表中。
- `image: <your-registry>/netapp/trident-operator:26.06.0` 如果您的 Trident 镜像位于与 CSI 镜像不同的注册表中。

步骤 4: 部署 **Trident** 操作员

Trident 安装程序提供了一个捆绑文件，可用于安装操作员和创建相关对象。捆绑文件是使用默认配置部署操作员和安装 Trident 的简便方法。

- 对于运行 Kubernetes 1.24 的集群，请使用 `bundle_pre_1_25.yaml`。
- 对于运行 Kubernetes 1.25 或更高版本的集群，请使用 `bundle_post_1_25.yaml`。

开始之前

- 默认情况下，Trident 安装程序在 `trident` 命名空间中部署运算符。如果 `trident` 命名空间不存在，请使用以下方法创建它：

```
kubectl apply -f deploy/namespace.yaml
```

- 要在 `trident` 命名空间以外的命名空间中部署运算符，请更新 `serviceaccount.yaml`、`clusterrolebinding.yaml` 和 `operator.yaml`，并使用 `kustomization.yaml` 生成捆绑包文件。
 - a. 使用以下命令创建 `kustomization.yaml`，其中 `<bundle.yaml>` 是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml`，基于您的 Kubernetes 版本。

```
cp deploy/kustomization_<bundle.yaml> deploy/kustomization.yaml
```

- b. 使用以下命令编译捆绑包，其中 `<bundle.yaml>` 是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml`，具体取决于您的 Kubernetes 版本。

```
kubectl kustomize deploy/ > deploy/<bundle.yaml>
```

步骤

1. 创建资源并部署 operator：

```
kubectl create -f deploy/<bundle.yaml>
```

2. 验证是否已创建 operator、deployment 和 replicaset。

```
kubectl get all -n <operator-namespace>
```



Kubernetes 集群中应该只有*一个实例*的运算符。请勿创建 Trident 运算符的多个部署。

第 5 步：更新 **TridentOrchestrator** 中的映像注册表位置

您的 **"Trident 和 CSI 镜像"** 可以位于一个注册表或不同的注册表中，但所有 CSI 映像必须位于同一注册表中。更新 `deploy/crds/tridentorchestrator_cr.yaml` 以根据注册表配置添加其他位置规格。

一个注册表中的图像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.06"
tridentImage: "<your-registry>/trident:26.06.0"
```

不同注册表中的图像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.06"
tridentImage: "<your-registry>/trident:26.06.0"
```

步骤 6：创建 **TridentOrchestrator** 并安装 Trident

现在，您可以创建 `TridentOrchestrator` 并安装 Trident。也可以进一步[定制您的 Trident 安装](#)使用 `TridentOrchestrator` 规范中的属性。以下示例显示了 Trident 和 CSI 映像位于不同注册表中的安装。

```
kubectl create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created
```

```

kubectl describe torc trident

Name:          trident
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/trident-autosupport:26.06
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:26.06.0
Status:
  Current Installation Params:
    IPv6:              false
    Autosupport Hostname:
    Autosupport Image:  <your-registry>/trident-autosupport:26.06
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:              true
    Http Request Timeout: 90s
    Image Pull Secrets:
    Image Registry:     <your-registry>
    k8sTimeout:         30
    Kubelet Dir:        /var/lib/kubelet
    Log Format:          text
    Probe Port:         17546
    Silence Autosupport: false
    Trident Image:      <your-registry>/trident:26.06.0
  Message:             Trident installed
  Namespace:           trident
  Status:              Installed
  Version:             v26.06.0
Events:
  Type Reason Age From Message ----
  Installing 74s trident-operator.netapp.io Installing Trident Normal
  Installed 67s trident-operator.netapp.io Trident installed

```

验证安装

有几种方法可以验证您的安装。

使用 **TridentOrchestrator** 状态

``TridentOrchestrator`` 的状态指示安装是否成功，并显示已安装的 `Trident` 版本。在安装过程中，``TridentOrchestrator`` 的状态从 ``Installing`` 更改为 ``Installed``。如果您观察到 ``Failed`` 状态，并且操作员无法自行恢复，`link:../troubleshooting.html["检查日志"]`。

状态	说明
安装	操作员正在使用此 <code>TridentOrchestrator</code> CR 安装 <code>Trident</code> 。
已安装	<code>Trident</code> 已成功安装。
卸载	操作员正在卸载 <code>Trident</code> ，因为 <code>spec.uninstall=true</code> 。
已卸载	<code>Trident</code> 已卸载。
失败	操作员无法安装、修补、更新或卸载 <code>Trident</code> ；操作员将自动尝试从此状态恢复。如果此状态持续存在，则需要故障排除。
正在更新	操作员正在更新现有安装。
错误	<code>TridentOrchestrator</code> 未使用。另一个已存在。

使用 **pod** 创建状态

您可以通过查看已创建的 Pod 的状态来确认 `Trident` 安装是否已完成：

```
kubectl get pods -n trident
```

NAME	READY	STATUS	RESTARTS
AGE			
trident-controller-7d466bf5c7-v4cpw1m	6/6	Running	0
trident-node-linux-mr6zc1m	2/2	Running	0
trident-node-linux-xrp7w1m	2/2	Running	0
trident-node-linux-zh2jt1m	2/2	Running	0
trident-operator-766f7b8658-ldzsv3m	1/1	Running	0

使用 **tridentctl**

您可以使用 ``tridentctl`` 检查已安装的 `Trident` 版本。

```
./tridentctl -n trident version
```

```
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 26.06.0       | 26.06.0       |
+-----+-----+
```

使用 Helm 部署 Trident 操作员（标准模式）

您可以使用 Helm 部署 Trident 操作员并安装 Trident。此过程适用于 Trident 所需的容器映像未存储在私有注册表中的安装。如果确实有专用映像注册表，请使用["脱机部署流程"](#)。

有关 Trident 26.06 的关键信息

您必须阅读以下有关 **Trident** 的重要信息。

▼ 有关 Trident 的重要信息

- Trident 现在支持 Kubernetes 1.37。在升级 Kubernetes 之前，请先升级 Trident。
- Trident 严格执行在 SAN 环境中使用多路径配置，在 multipath.conf 文件中的建议值为 `find_multipaths: no`。

使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes` 或 `find_multipaths: smart` 值将导致挂载失败。自 21.07 版本发布以来，Trident 建议使用 `find_multipaths: no`。

部署 Trident 操作员并使用 Helm 安装 Trident

使用 Trident ["Helm Chart"](#) 您可以部署 Trident 操作员并一步安装 Trident。

请检查 ["安装概述"](#) 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

除了 ["部署先决条件"](#) 之外，您还需要 ["Helm 版本 3"](#)。

步骤

1. 添加 Trident Helm 存储库：

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

2. 使用 `helm install` 并指定部署的名称，如下例所示，其中 `100.2606.0` 是您要安装的 Trident 版本。

```
helm install <name> netapp-trident/trident-operator --version 100.2606.0
--create-namespace --namespace <trident-namespace>
```



如果您已经为 Trident 创建了命名空间，则 `--create-namespace` 参数不会创建其他命名空间。

您可以使用 `helm list` 查看安装详细信息，例如名称、命名空间、图表、状态、应用版本和修订版本号。

在安装过程中传递配置数据

在安装过程中，有两种传递配置数据的方法：

选项	说明
<code>--values</code> (或 <code>-f</code>)	指定具有重写的 YAML 文件。这可以指定多次，最右边的文件将优先。
<code>--set</code>	在命令行上指定重写。

例如，要更改 `debug` 的默认值，请运行以下命令，其中 `100.2606.0` 是您要安装的 Trident 版本：

```
helm install <name> netapp-trident/trident-operator --version 100.2606.0
--create-namespace --namespace trident --set tridentDebug=true
```

配置选项

此表和作为 Helm 图表一部分的 `values.yaml` 文件提供了键及其默认值的列表。

选项	说明	默认
<code>nodeSelector</code>	Pod 分配的节点标签	
<code>podAnnotations</code>	Pod 注释	
<code>deploymentAnnotations</code>	部署注释	
<code>tolerations</code>	Pod 分配的容忍度	

选项	说明	默认
affinity	Pod 分配的亲和力	<pre> affinity: nodeAffinity: requiredDuringSchedulingIgnoredDur ingExecution: nodeSelectorTerms: - matchExpressions: - key: kubernetes.io/arch operator: In values: - arm64 - amd64 - key: kubernetes.io/os operator: In values: - linux </pre> <div>  <p>请勿从 values.yaml 文件中删除默认关联。如果要提供自定义关联，请扩展默认关联。</p> </div>
tridentContr ollerPluginN odeSelector	Pod 的其他节点选择器。有关详细 信息，请参阅 了解控制器 pod 和节 点 pod 。	
tridentContr ollerPluginT olerations	覆盖 Kubernetes 对 pod 的容忍。有 关详细信息，请参阅 了解控制器 pod 和节点 pod 。	
tridentNodeP luginNodeSel ector	Pod 的其他节点选择器。有关详细 信息，请参阅 了解控制器 pod 和节 点 pod 。	
tridentNodeP luginTolerat ions	覆盖 Kubernetes 对 pod 的容忍。有 关详细信息，请参阅 了解控制器 pod 和节点 pod 。	
imageRegistr y	标识 trident-operator、 trident 和其他图像的注册表。留 空以接受默认值。重要提示：在专 用存储库中安装 Trident 时，如果您 使用 imageRegistry 开关来指定 存储库位置，请不要在存储库路径 中使用 /netapp/。	""
imagePullPol icy	为 trident-operator 设置镜像 拉取策略。	IfNotPresent

选项	说明	默认
imagePullSecrets	为 trident-operator、trident 和其他镜像设置镜像拉取密钥。	
kubeletDir	允许覆盖 kubelet 内部状态的主机位置。	"/var/lib/kubelet"
operatorLogLevel	允许将 Trident 操作员的日志级别设置为：trace、debug、info、warn、error、或 fatal。	"info"
operatorDebug	允许将 Trident 操作员的日志级别设置为调试。	true
operatorImage	允许完全覆盖 trident-operator 的图像。	""
operatorImageTag	允许覆盖 trident-operator 图像的标记。	""
tridentIPv6	允许启用 Trident 在 IPv6 集群中工作。	false
tridentK8sTimeout	覆盖大多数 Kubernetes API 操作的默认 30 秒超时（如果非零，以秒为单位）。	0
tridentHttpRequestTimeout	覆盖 HTTP 请求的默认 90 秒超时，0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutoSupport	允许禁用 Trident 定期 AutoSupport 报告。	false
tridentAutoSupportImageTag	允许覆盖 Trident AutoSupport 容器的图像标记。	<version>
tridentAutoSupportProxy	使 Trident AutoSupport 容器能够通过 HTTP 代理拨打回家电话。	""
tridentLogFormat	设置 Trident 日志记录格式 (text 或 json)。	"text"
tridentDisableAuditLog	禁用 Trident 审计记录器。	true
tridentLogLevel	允许将 Trident 的日志级别设置为：trace、debug、info、warn、error 或 fatal。	"info"
tridentDebug	允许将 Trident 的日志级别设置为 debug。您可以通过与节点健康检查 (NHC) 运算符集成来自动执行强制分离过程。有关信息，请参见 "使用 Trident 自动化有状态应用程序的故障转移" 。	false

选项	说明	默认
tridentLogWorkflows	允许为跟踪日志记录或日志抑制启用特定 Trident 工作流。	""
tridentLogLayers	允许启用特定 Trident 图层以进行跟踪日志记录或日志抑制。	""
tridentImage	允许完全覆盖 Trident 的镜像。	""
tridentImageTag	允许覆盖 Trident 镜像的标签。	""
tridentProbePort	允许覆盖用于 Kubernetes 活动/就绪探测的默认端口。	""
windows	允许在 Windows worker 节点上安装 Trident。	false
enableForceDetach	允许启用强制分离功能。	false
excludePodSecurityPolicy	从创建中排除操作员 Pod 安全策略。	false
cloudProvider	在 AKS 集群上使用托管标识或云标识时设置为 "Azure"。在 EKS 集群上使用云标识时设置为 "AWS"。	""
cloudIdentity	在 AKS 集群上使用云标识时，设置为工作负载标识 ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx")。在 EKS 集群上使用云标识时，设置为 AWS IAM 角色 ("eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role")。	""
iscsiSelfHealingInterval	调用 iSCSI 自我修复的时间间隔。	5m0s
iscsiSelfHealingWaitTime	iSCSI 自我修复尝试通过执行注销和后续登录来解决过时会话的持续时间。	7m0s
nodePrep	使 Trident 能够准备 Kubernetes 集群的节点，以使用指定的数据存储协议管理卷。*目前，iscsi 是唯一受支持的值。*注意：从 OpenShift 4.19 开始，此功能支持的最低 Trident 版本为 25.06.1。	

选项	说明	默认
enableConcurrency	启用并发 Trident 控制器操作，以提高吞吐量。有关支持的驱动程序的更多信息，请参见链接： https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability [Controller Scalability documentation]。	false
k8sAPIQPS	控制器在与 Kubernetes API 服务器通信时使用的每秒查询数 (QPS) 限制。Burst 值根据 QPS 值自动设置。	100；可选

选项	说明	默认
resources	<p>为 Trident 控制器、节点和操作员 pod 设置 Kubernetes 资源限制和请求。您可以为每个容器和 sidecar 配置 CPU 和内存，以管理 Kubernetes 中的资源分配。</p> <p>有关配置资源请求和限制的详细信息，请参阅 "Pod 和容器的资源管理"。</p> <div>  <ul style="list-style-type: none"> 请勿更改任何容器或字段的名 称。 请勿更改缩进 - YAML 缩进对于正确解析至关重要。 </div> <div>  <ul style="list-style-type: none"> 默认情况下不会应用任何限制 - 只有请求具有默认值，如果未指定，则会自动应用。 容器名称按 Pod 规格中显示的方式列出。 Sidecar 列在每个主容器下。 检查 TORC 的 `status.CurrentInstallationParams` 字段以查看当前应用的值。 </div>	<pre>resources: controller: trident-main: requests: cpu: 10m memory: 80Mi limits: cpu: memory: csi-provisioner: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-attacher: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-resizer: requests: cpu: 3m memory: 20Mi limits: cpu: memory: csi-snapshotter: requests: cpu: 2m memory: 20Mi limits: cpu: memory: trident-autosupport: requests: cpu: 1m memory: 30Mi limits: cpu: memory: node:</pre>

选项	说明	默认
httpsMetrics	为 Prometheus 指标端点启用 HTTPS。	false
hostNetwork	为 Trident 控制器启用主机网络。如果要在多主机网络中分离前端和后端流量，则此功能非常有用。	false

了解控制器 pod 和节点 pod

Trident 作为单个控制器 pod 运行，并在集群中的每个工作节点上加上一个节点 pod。节点 pod 必须运行在任何可能要装载 Trident 卷的主机上。

Kubernetes ["节点选择器"](#) 和 ["容忍和污点"](#) 用于限制 pod 在特定或首选节点上运行。使用 ControllerPlugin 和 NodePlugin，您可以指定约束和覆盖。

- 控制器插件处理卷配置和管理，例如快照和调整大小。
- 节点插件处理将存储附加到节点。

使用 Helm 部署 Trident 操作员（离线模式）

您可以使用 Helm 部署 Trident 操作员并安装 Trident。此过程适用于 Trident 所需的容器映像存储在私有注册表中的安装。如果您没有私有映像注册表，请使用 ["标准部署流程"](#)。

有关 Trident 26.06 的关键信息

您必须阅读以下有关 **Trident** 的重要信息。

▼ **有关 Trident 的重要信息**

- Trident 现在支持 Kubernetes 1.37。在升级 Kubernetes 之前，请先升级 Trident。
- Trident 严格执行在 SAN 环境中使用多路径配置，在 multipath.conf 文件中的建议值为 `find_multipaths: no`。

使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes` 或 `find_multipaths: smart` 值将导致挂载失败。自 21.07 版本发布以来，Trident 建议使用 `find_multipaths: no`。

部署 Trident 操作员并使用 Helm 安装 Trident

使用 Trident ["Helm Chart"](#) 您可以部署 Trident 操作员并一步安装 Trident。

请检查 ["安装概述"](#) 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

除了 ["部署先决条件"](#) 之外，您还需要 ["Helm 版本 3"](#)。



在专用存储库中安装 Trident 时，如果您使用 `imageRegistry`` 开关指定存储库位置，请不要在存储库路径中使用 ``/netapp/``。

步骤

1. 添加 Trident Helm 存储库：

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

2. 使用 ``helm install`` 并指定部署和映像注册表位置的名称。您的 "Trident 和 CSI 镜像" 可以位于一个注册表或不同的注册表中，但所有 CSI 映像必须位于同一注册表中。在示例中，``100.2606.0`` 是您要安装的 Trident 版本。

一个注册表中的图像

```
helm install <name> netapp-trident/trident-operator --version
100.2606.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

不同注册表中的图像

```
helm install <name> netapp-trident/trident-operator --version
100.2606.0 --set imageRegistry=<your-registry> --set operatorImage
=<your-registry>/trident-operator:26.06.0 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:26.06
--set tridentImage=<your-registry>/trident:26.06.0 --create
--namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```



如果您已经为 Trident 创建了命名空间，则 `--create-namespace` 参数不会创建其他命名空间。

您可以使用 ``helm list`` 查看安装详细信息，例如名称、命名空间、图表、状态、应用版本和修订版号。

在安装过程中传递配置数据

在安装过程中，有两种传递配置数据的方法：

选项	说明
<code>--values</code> (或 <code>-f</code>)	指定具有重写的 YAML 文件。这可以指定多次，最右边的文件将优先。
<code>--set</code>	在命令行上指定重写。

例如，要更改 ``debug`` 的默认值，请运行以下命令，其中 ``100.2606.0`` 是您要安装的 Trident 版本：

```
helm install <name> netapp-trident/trident-operator --version 100.2606.0
--create-namespace --namespace trident --set tridentDebug=true
```

要添加 nodePrep 值，请运行以下命令：

```
helm install <name> netapp-trident/trident-operator --version 100.2606.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

配置选项

此表和作为 Helm 图表一部分的 `values.yaml` 文件提供了键及其默认值的列表。



请勿从 values.yaml 文件中删除默认关联。如果要提供自定义关联，请扩展默认关联。

选项	说明	默认
nodeSelector	Pod 分配的节点标签	
podAnnotations	Pod 注释	
deploymentAnnotations	部署注释	
tolerations	Pod 分配的容忍度	

选项	说明	默认
affinity	Pod 分配的亲和力	<pre> affinity: nodeAffinity: requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms: - matchExpressions: - key: kubernetes.io/arch operator: In values: - arm64 - amd64 - key: kubernetes.io/os operator: In values: - linux </pre> <div>  <p>请勿从 values.yaml 文件中删除默认关联。如果要提供自定义关联，请扩展默认关联。</p> </div>
tridentControllerPluginNodeSelector	Pod 的其他节点选择器。有关详细信息，请参阅 "了解控制器 pod 和节点 pod" 。	
tridentControllerPluginTolerations	覆盖 Kubernetes 对 pod 的容忍。有关详细信息，请参阅 "了解控制器 pod 和节点 pod" 。	
tridentNodePluginNodeSelector	Pod 的其他节点选择器。有关详细信息，请参阅 "了解控制器 pod 和节点 pod" 。	
tridentNodePluginTolerations	覆盖 Kubernetes 对 pod 的容忍。有关详细信息，请参阅 "了解控制器 pod 和节点 pod" 。	

选项	说明	默认
imageRegistry	标识 trident-operator、trident 和其他图像的注册表。留空以接受默认值。重要提示：在专用存储库中安装 Trident 时，如果您使用 imageRegistry 开关来指定存储库位置，请不要在存储库路径中使用 /netapp/。	""
imagePullPolicy	为 trident-operator 设置镜像拉取策略。	IfNotPresent
imagePullSecrets	为 trident-operator、trident 和其他镜像设置镜像拉取密钥。	
kubeletDir	允许覆盖 kubelet 内部状态的主机位置。	"/var/lib/kubelet"
operatorLogLevel	允许将 Trident 操作员的日志级别设置为：trace、debug、info、warn、error、或 fatal。	"info"
operatorDebug	允许将 Trident 操作员的日志级别设置为调试。	true
operatorImage	允许完全覆盖 trident-operator 的图像。	""
operatorImageTag	允许覆盖 trident-operator 图像的标记。	""
tridentIPv6	允许启用 Trident 在 IPv6 集群中工作。	false
tridentK8sTimeout	<p>覆盖大多数 Kubernetes API 操作的默认 180 秒超时（如果非零，以秒为单位）。</p> <div>  <p>该 tridentK8sTimeout 参数仅适用于 Trident 安装。</p> </div>	180
tridentHttpRequestTimeout	覆盖 HTTP 请求的默认 90 秒超时，0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutosupport	允许禁用 Trident 定期 AutoSupport 报告。	false
tridentAutosupportImageTag	允许覆盖 Trident AutoSupport 容器的图像标记。	<version>
tridentAutosupportProxy	使 Trident AutoSupport 容器能够通过 HTTP 代理拨打回家电话。	""

选项	说明	默认
tridentLogFormat	设置 Trident 日志记录格式 (text 或 json)。	"text"
tridentDisableAuditLog	禁用 Trident 审计记录器。	true
tridentLogLevel	允许将 Trident 的日志级别设置为：trace、debug、info、warn、error 或 fatal。	"info"
tridentDebug	允许将 Trident 的日志级别设置为 debug。	false
tridentLogWorkflows	允许为跟踪日志记录或日志抑制启用特定 Trident 工作流。	""
tridentLogLayers	允许启用特定 Trident 图层以进行跟踪日志记录或日志抑制。	""
tridentImage	允许完全覆盖 Trident 的镜像。	""
tridentImageTag	允许覆盖 Trident 镜像的标签。	""
tridentProbePort	允许覆盖用于 Kubernetes 活动/就绪探测的默认端口。	""
windows	允许在 Windows worker 节点上安装 Trident。	false
enableForceDetach	允许启用强制分离功能。您可以通过与节点健康检查 (NHC) 运算符集成来自动执行强制分离过程。有关信息，请参见 "使用 Trident 自动化有状态应用程序的故障转移" 。	false
excludePodSecurityPolicy	从创建中排除操作员 Pod 安全策略。	false
nodePrep	<p>使 Trident 能够准备 Kubernetes 集群的节点，以使用指定的数据存储协议管理卷。目前，iscsi 是唯一受支持的值。</p> <div>  <p>从 OpenShift 4.19 开始，此功能支持的最低 Trident 版本为 25.06.1。</p> </div>	

选项	说明	默认
resources	<p>为 Trident 控制器、节点和操作员 pod 设置 Kubernetes 资源限制和请求。您可以为每个容器和 sidecar 配置 CPU 和内存，以管理 Kubernetes 中的资源分配。</p> <p>有关配置资源请求和限制的详细信息，请参阅 "Pod 和容器的资源管理"。</p> <div>  <ul style="list-style-type: none"> 请勿更改任何容器或字段的名 称。 请勿更改缩进 - YAML 缩进对于正确解析至关重要。 </div> <div>  <ul style="list-style-type: none"> 默认情况下不会应用任何限制 - 只有请求具有默认值。 容器名称按 Pod 规格中显示的方式列出。 Sidecar 列在每个主容器下。 检查 TORC 的 `status.CurrentInstallationParams` 字段以查看当前应用的值。 </div>	<pre>resources: controller: trident-main: requests: cpu: 10m memory: 80Mi limits: cpu: memory: csi-provisioner: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-attacher: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-resizer: requests: cpu: 3m memory: 20Mi limits: cpu: memory: csi-snapshotter: requests: cpu: 2m memory: 20Mi limits: cpu: memory: trident-autosupport: requests: cpu: 1m memory: 30Mi limits: cpu: memory:</pre>

定制 Trident 操作员安装

Trident 操作员允许您使用 `TridentOrchestrator` 规范中的属性自定义 Trident 安装。如果要自定义超出 `TridentOrchestrator` 参数允许范围的安装，请考虑使用 `tridentctl` 生成自定义 YAML 清单以根据需要进行修改。

了解控制器 pod 和节点 pod

Trident 作为单个控制器 pod 和集群中每个工作节点上的节点 pod 运行。节点 pod 必须运行在任何可能要装载 Trident 卷的主机上。

Kubernetes ["节点选择器"](#) 和 ["容忍和污点"](#) 用于限制 pod 在特定或首选节点上运行。使用 ControllerPlugin 和 NodePlugin，您可以指定约束和覆盖。

- 控制器插件处理卷配置和管理，例如快照和调整大小。
- 节点插件处理将存储附加到节点。

配置选项



`spec.namespace` 在 `TridentOrchestrator` 中指定，以表示安装 Trident 的命名空间。此参数*在安装 Trident 后无法更新*。尝试执行此操作会导致 `TridentOrchestrator` 状态更改为 `Failed`。Trident 不能跨命名空间迁移。

此表详细说明了 `TridentOrchestrator` 属性。

参数	说明	默认
<code>namespace</code>	安装 Trident 的命名空间	"default"
<code>debug</code>	为 Trident 启用调试	false
<code>enableForceDetach</code>	<code>ontap-san</code> 、 <code>ontap-san-economy</code> 、 <code>ontap-nas</code> 和 <code>ontap-nas-economy</code> 仅适用。与 Kubernetes 非优雅节点关闭 (NGNS) 一起使用，使集群管理员能够在节点变得不健康时将带有装载卷的工作负载安全地迁移到新节点。有关信息，请参见 "使用 Trident 自动化有状态应用程序的故障转移" 。	false
<code>windows</code>	设置为 true 可在 Windows 工作节点上启用安装。	false
<code>cloudProvider</code>	在 AKS 集群上使用托管标识或云标识时设置为 "Azure"。在 EKS 集群上使用云标识时设置为 "AWS"。在 GKE 集群上使用云标识时设置为 "GCP"。	" "

参数	说明	默认
cloudIdentity	在 AKS 集群上使用云标识时，设置为工作负载标识 ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")。在 EKS 集群上使用云标识时，设置为 AWS IAM 角色 ("eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role")。在 GKE 集群上使用云标识时，设置为云标识 ("iam.gke.io/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com")。	" "
IPv6	通过 IPv6 安装 Trident	false
k8sTimeout	<div>  <div>该 k8sTimeout 参数仅适用于 Trident 安装。</div> </div> Kubernetes 操作超时。	180sec
silenceAutosupport	不要将自动支持捆绑包自动发送到 NetApp	false
autosupportImage	Autosupport Telemetry 的容器映像	"netapp/trident-autosupport10"
autosupportProxy	用于发送 Autosupport 遥测的代理的地址/端口	"http://proxy.example.com:8888"
uninstall	用于卸载 Trident 的标志	false
logFormat	要使用的 Trident 日志记录格式 [text,json]	"text"
tridentImage	要安装的 Trident 镜像	"netapp/trident:26.06"
imageRegistry	内部注册表的路径，格式为 <registry FQDN>[:port][/subpath]	"registry.k8s.io"
kubeletDir	主机上的 kubelet 目录路径	"/var/lib/kubelet"
wipeout	要删除的资源列表，以执行 Trident 的完全删除	
imagePullSecrets	从内部注册表中提取镜像的密钥	
imagePullPolicy	设置 Trident 操作员的镜像拉取策略。有效值为： Always 始终拉取镜像。 IfNotPresent 仅在节点上不存在镜像时拉取镜像。 Never 从不拉取镜像。	IfNotPresent
controllerPluginNodeSelector	Pod 的其他节点选择器。遵循与 pod.spec.nodeSelector 相同的格式。	无默认值；可选
controllerPluginTolerations	覆盖 Kubernetes 对 pod 的容忍。遵循与 pod.spec.Tolerations 相同的格式。	无默认值；可选
nodePluginNodeSelector	Pod 的其他节点选择器。遵循与 pod.spec.nodeSelector 相同的格式。	无默认值；可选
nodePluginTolerations	覆盖 Kubernetes 对 pod 的容忍。遵循与 pod.spec.Tolerations 相同的格式。	无默认值；可选

参数	说明	默认
nodePrep	<p>使 Trident 能够准备 Kubernetes 集群的节点，以使用指定的数据存储协议管理卷。目前， iscsi 是唯一受支持的值。</p> <div>  <p>从 OpenShift 4.19 开始，此功能支持的最低 Trident 版本为 25.06.1。</p> </div>	
k8sAPIQPS	控制器在与 Kubernetes API 服务器通信时使用的每秒查询数 (QPS) 限制。Burst 值根据 QPS 值自动设置。	100；可选
enableConcurrency	<p>启用并发 Trident 控制器操作，以提高吞吐量。有关支持的驱动程序的更多信息，请参见链接： https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability[Controller Scalability documentation]。</p>	resources

参数	说明	默认
<p>为 Trident 控制器和节点 pod 设置 Kubernetes 资源限制和请求。您可以为每个容器和 sidecar 配置 CPU 和内存，以管理 Kubernetes 中的资源分配。</p> <p>有关配置资源请求和限制的详细信息，请参阅 "Pod 和容器的资源管理"。</p>	<pre>resources: controller: trident-main: requests: cpu: 10m memory: 80Mi limits: cpu: memory: csi-provisioner: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-attacher: requests: cpu: 2m memory: 20Mi limits: cpu: memory: csi-resizer: requests: cpu: 3m memory: 20Mi limits: cpu: memory: csi-snapshotter: requests: cpu: 2m memory: 20Mi limits: cpu: memory: trident-autosupport: requests: cpu: 1m memory: 30Mi limits: cpu: memory: node:</pre>	httpsMetrics



- 请勿更改任何容器或字段的名称。
- 请勿更改缩进 - YAML 缩进对于正确解析至关重要。



- 默认情况下不会应用任何限制 - 只有请求具有默认值，如果未指定，则会自动应用。
- 容器名称按 Pod 规格中显示的方式列出。
- Sidecar 列在每个主容器下。

参数	说明	默认
为 Prometheus 指标端点启用 HTTPS。	false	hostNetwork



有关格式化 pod 参数的更多信息，请参阅 ["将 Pod 分配给节点"](#)。

示例配置

您可以在定义[\[配置选项\]](#)时使用 `TridentOrchestrator` 中的属性来自定义您的安装。

▼ 基本自定义配置

此示例在运行 `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml` 命令后创建，表示一个基本的自定义安装：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
```

▼ 节点选择器

此示例使用节点选择器安装 Trident。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  controllerPluginNodeSelector:
    nodetype: master
  nodePluginNodeSelector:
    storage: netapp
```

▼ Windows 工作节点

此示例在运行 `cat deploy/crds/tridentorchestrator_cr.yaml` 命令后创建，用于在 Windows 辅助节点上安装 Trident。

```

apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true

```

▼ AKS 集群上的托管身份

此示例安装 Trident 以在 AKS 集群上启用托管身份。

```

apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  cloudProvider: "Azure"

```

▼ AKS 集群上的云身份

此示例安装 Trident 以在 AKS 集群上与云标识一起使用。

```

apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  cloudProvider: "Azure"
  cloudIdentity: 'azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx'

```

▼ EKS 集群上的云身份

此示例安装 Trident 以在 AKS 集群上与云标识一起使用。

```

apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident

```

```
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/trident-role'"
```

▼ 适用于 GKE 的云标识

此示例安装 Trident 以便与 GKE 集群上的云身份一起使用。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcp-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '012345678901'
  network: gcnv-network
  location: us-west2
  serviceLevel: Premium
  storagePool: pool-premium1
```

▼ Trident 控制器和 Trident Linux 节点 pod 的 Kubernetes 资源请求和限制配置

此示例为 Trident 控制器和 Trident Linux 节点 pod 配置 Kubernetes 资源请求和限制。



免责声明：本示例中提供的请求和限制值仅用于演示目的。根据您的环境和工作负载要求调整这些值。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
```

```
    limits:
      cpu: 200m
      memory: 256Mi
# sidecars
csi-provisioner:
  requests:
    cpu: 2m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
csi-attacher:
  requests:
    cpu: 2m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
csi-resizer:
  requests:
    cpu: 3m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
csi-snapshotter:
  requests:
    cpu: 2m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
trident-autosupport:
  requests:
    cpu: 1m
    memory: 30Mi
  limits:
    cpu: 50m
    memory: 128Mi
node:
  linux:
    trident-main:
      requests:
        cpu: 10m
        memory: 60Mi
      limits:
```

```

        cpu: 200m
        memory: 256Mi
# sidecars
node-driver-registrar:
  requests:
    cpu: 1m
    memory: 10Mi
  limits:
    cpu: 50m
    memory: 32Mi

```

▼ Trident 控制器和 Trident Windows 和 Linux 节点 pod 的 Kubernetes 资源请求和限制配置

此示例为 Trident 控制器和 Trident Windows 和 Linux 节点 pod 配置 Kubernetes 资源请求和限制。



免责声明：本示例中提供的请求和限制值仅用于演示目的。根据您的环境和工作负载要求调整这些值。

```

apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  windows: true
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
        limits:
          cpu: 200m
          memory: 256Mi
      # sidecars
    csi-provisioner:
      requests:
        cpu: 2m
        memory: 20Mi
      limits:
        cpu: 100m
        memory: 64Mi
    csi-attacher:

```

```

    requests:
      cpu: 2m
      memory: 20Mi
    limits:
      cpu: 100m
      memory: 64Mi
csi-resizer:
  requests:
    cpu: 3m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
csi-snapshotter:
  requests:
    cpu: 2m
    memory: 20Mi
  limits:
    cpu: 100m
    memory: 64Mi
trident-autosupport:
  requests:
    cpu: 1m
    memory: 30Mi
  limits:
    cpu: 50m
    memory: 128Mi
node:
  linux:
    trident-main:
      requests:
        cpu: 10m
        memory: 60Mi
      limits:
        cpu: 200m
        memory: 256Mi
    # sidecars
  node-driver-registrar:
    requests:
      cpu: 1m
      memory: 10Mi
    limits:
      cpu: 50m
      memory: 32Mi
windows:
  trident-main:

```

```
    requests:
      cpu: 6m
      memory: 40Mi
    limits:
      cpu: 200m
      memory: 128Mi
# sidecars
node-driver-registrar:
  requests:
    cpu: 6m
    memory: 40Mi
  limits:
    cpu: 100m
    memory: 128Mi
liveness-probe:
  requests:
    cpu: 2m
    memory: 40Mi
  limits:
    cpu: 50m
    memory: 64Mi
```

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 <http://www.netapp.com/TM> 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。