Instala usando el operador Trident

Trident

NetApp
July 01, 2026

Tabla de contenidos

Instala usando el operador Trident

Implementa manualmente el operador Trident (modo estándar)

Puedes desplegar manualmente el operador Trident para instalar Trident. Este proceso aplica a instalaciones donde las imágenes de contenedor requeridas por Trident no están almacenadas en un registro privado. Si tienes un registro de imágenes privado, usa el "proceso para el despliegue fuera de línea".

Información crítica sobre Trident 26.02

Debes leer la siguiente información crítica sobre Trident.

Información crítica sobre Trident

- Kubernetes 1.35 ya es compatible con Trident. Actualiza Trident antes de actualizar Kubernetes.
- Trident impone estrictamente el uso de la configuración de multivía en entornos SAN, con un valor recomendado de `find_multipaths: no` en el archivo multipath.conf.

  El uso de una configuración sin multivía o el uso de `find_multipaths: yes` o `find_multipaths: smart` en el archivo multipath.conf provocará fallos de montaje. Trident ha recomendado el uso de `find_multipaths: no` desde la versión 21.07.

Despliega manualmente el operador Trident e instala Trident

Revisa "la visión general de la instalación" para asegurarte de que cumpliste los requisitos previos de instalación y seleccionaste la opción de instalación correcta para tu entorno.

Antes de empezar

Antes de comenzar la instalación, inicia sesión en el host Linux y verifica que está gestionando un servidor en funcionamiento, "clúster de Kubernetes compatible" y que tienes los privilegios necesarios.

Con OpenShift, usa `oc` en vez de `kubectl` en todos los ejemplos que siguen, y primero inicia sesión como **system:admin** ejecutando `oc login -u system:admin` o `oc login -u kube-admin`.

1. Verifica tu versión de Kubernetes:

```
kubectl version
```

2. Verifica los privilegios del administrador del clúster:

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

3. Verifica que puedes lanzar un pod que use una imagen de Docker Hub y acceder a tu sistema de almacenamiento a través de la red de pods:

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

Paso 1: descarga el paquete de instalación de Trident

El paquete instalador de Trident contiene todo lo que necesitas para desplegar el operador Trident e instalar Trident. Descarga y extrae la última versión del instalador de Trident desde "la sección *Assets* en GitHub".

```
wget https://github.com/NetApp/trident/releases/download/v26.02.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

Paso 2: crea el `TridentOrchestrator`CRD

Crea la `TridentOrchestrator` Custom Resource Definition (CRD). Vas a crear un `TridentOrchestrator` Custom Resource más adelante. Usa la versión CRD YAML apropiada en `deploy/crds` para crear el `TridentOrchestrator` CRD.

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

Paso 3: despliega el operador Trident

El instalador de Trident proporciona un archivo de paquete que puede usarse para instalar el operador y crear objetos asociados. El archivo de paquete es una forma sencilla de desplegar el operador e instalar Trident usando una configuración predeterminada.

- Para los clústeres que ejecutan Kubernetes 1.24, usa `bundle_pre_1_25.yaml`.
- Para clústeres que ejecutan Kubernetes 1.25 o posterior, usa `bundle_post_1_25.yaml`.

Antes de empezar

- Por defecto, el instalador de Trident despliega el operador en el `trident` namespace. Si el `trident` namespace no existe, créalo usando:

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

- Para desplegar el operador en un espacio de nombres distinto del `trident` namespace, actualiza `serviceaccount.yaml`, `clusterrolebinding.yaml` y `operator.yaml` y genera tu archivo bundle usando el `kustomization.yaml`.

  a. Crea el `kustomization.yaml` usando el siguiente comando donde *<bundle.yaml>* es `bundle_pre_1_25.yaml` o `bundle_post_1_25.yaml` según tu versión de Kubernetes.

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

  b. Compila el paquete usando el siguiente comando donde *<bundle.yaml>* es `bundle_pre_1_25.yaml` o `bundle_post_1_25.yaml` según tu versión de Kubernetes.

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

Pasos

1. Crea los recursos y despliega el operador:

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

2. Verifica que se hayan creado el operador, el deployment y los replicasets.

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

> Solo debe haber **una instancia** del operador en un clúster de Kubernetes. No crees varias implementaciones del operador Trident.

Paso 4: crea el `TridentOrchestrator` e instala Trident

Ahora puedes crear el `TridentOrchestrator` e instalar Trident. Opcionalmente, puedes "personaliza tu instalación Trident" usando los atributos en la `TridentOrchestrator` spec.

```
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:26.02
    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:26.02.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v26.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

Verifica la instalación

Hay varias formas de verificar tu instalación.

Usando `TridentOrchestrator` status

El estado de `TridentOrchestrator` indica si la instalación fue exitosa y muestra la versión de Trident instalada. Durante la instalación, el estado de `TridentOrchestrator` cambia de `Installing` a `Installed`. Si observas el estado de `Failed` y el operador no puede recuperarse por sí solo, "revisa los registros".

Estado	Descripción
Instalando	El operador está instalando Trident usando este `TridentOrchestrator` CR.
Instalado	Trident se ha instalado correctamente.
Desinstalar	El operador está desinstalando Trident porque `spec.uninstall=true`.
Desinstalado	Trident está desinstalado.
Con errores	El operador no pudo instalar, parchear, actualizar o desinstalar Trident; el operador intentará automáticamente recuperarse de este estado. Si este estado persiste necesitarás hacer una solución de problemas.
Actualizando	El operador está actualizando una instalación existente.
Error	El `TridentOrchestrator` no se usa. Ya existe otro.

Uso del estado de creación de pods

Puedes confirmar si la instalación de Trident se completó revisando el estado de los pods creados:

```
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
```

Usando `tridentctl`

Puedes usar `tridentctl` para comprobar la versión de Trident instalada.

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.02.0        | 26.02.0        |
+----------------+----------------+
```

Despliega manualmente el operador Trident (modo offline)

Puedes desplegar manualmente el operador Trident para instalar Trident. Este proceso aplica a instalaciones donde las imágenes de contenedor requeridas por Trident se almacenan en un registro privado. Si no tienes un registro de imágenes privado, usa el "proceso para la implementación estándar".

Información crítica sobre Trident

Debes leer la siguiente información crítica sobre Trident.

Información crítica sobre Trident

- Kubernetes 1.35 ya es compatible con Trident. Actualiza Trident antes de actualizar Kubernetes.
- Trident impone estrictamente el uso de la configuración de multivía en entornos SAN, con un valor recomendado de `find_multipaths: no` en el archivo multipath.conf.

  El uso de una configuración sin multivía o el uso de `find_multipaths: yes` o `find_multipaths: smart` en el archivo multipath.conf provocará fallos de montaje. Trident ha recomendado el uso de `find_multipaths: no` desde la versión 21.07.

Despliega manualmente el operador Trident e instala Trident

Revisa "la visión general de la instalación" para asegurarte de que cumpliste los requisitos previos de instalación y seleccionaste la opción de instalación correcta para tu entorno.

Antes de empezar

Inicia sesión en el host Linux y verifica que está gestionando un servidor que funciona y "clúster de Kubernetes compatible" y que tienes los privilegios necesarios.

Con OpenShift, usa `oc` en vez de `kubectl` en todos los ejemplos que siguen, y primero inicia sesión como **system:admin** ejecutando `oc login -u system:admin` o `oc login -u kube-admin`.

1. Verifica tu versión de Kubernetes:

```
kubectl version
```

2. Verifica los privilegios del administrador del clúster:

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

3. Verifica que puedes lanzar un pod que use una imagen de Docker Hub y acceder a tu sistema de almacenamiento a través de la red de pods:

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

Paso 1: descarga el paquete de instalación de Trident

El paquete instalador de Trident contiene todo lo que necesitas para desplegar el operador Trident e instalar Trident. Descarga y extrae la última versión del instalador de Trident desde "la sección *Assets* en GitHub".

```
wget https://github.com/NetApp/trident/releases/download/v6.0/trident-
installer-26.02.0.tar.gz
tar -xf trident-installer-26.02.0.tar.gz
cd trident-installer
```

Paso 2: crea el `TridentOrchestrator`CRD

Crea la `TridentOrchestrator` Custom Resource Definition (CRD). Vas a crear una `TridentOrchestrator` Custom Resources más adelante. Usa la versión adecuada de CRD YAML en `deploy/crds` para crear la `TridentOrchestrator` CRD:

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

Paso 3: actualiza la ubicación del registro en el operador

En `/deploy/operator.yaml`, actualiza `image: docker.io/netapp/trident-operator:26.02.0` para reflejar la ubicación de tu registro de imágenes. Tu "Imágenes de Trident y CSI" puede estar en un registro o en registros diferentes, pero todas las imágenes CSI deben estar en el mismo registro. Por ejemplo:

- `image: <your-registry>/trident-operator:26.02.0` si tus imágenes están todas ubicadas en un registro.
- `image: <your-registry>/netapp/trident-operator:26.02.0` si tu imagen de Trident está en

un registro diferente al de tus imágenes CSI.

Paso 4: despliega el operador Trident

El instalador de Trident proporciona un archivo de paquete que puede usarse para instalar el operador y crear objetos asociados. El archivo de paquete es una forma sencilla de desplegar el operador e instalar Trident usando una configuración predeterminada.

- Para los clústeres que ejecutan Kubernetes 1.24, usa `bundle_pre_1_25.yaml`.
- Para clústeres que ejecutan Kubernetes 1.25 o posterior, usa `bundle_post_1_25.yaml`.

Antes de empezar

- Por defecto, el instalador de Trident despliega el operador en el `trident` namespace. Si el `trident` namespace no existe, créalo usando:

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

- Para desplegar el operador en un espacio de nombres distinto del `trident` namespace, actualiza `serviceaccount.yaml`, `clusterrolebinding.yaml` y `operator.yaml` y genera tu archivo bundle usando el `kustomization.yaml`.

  a. Crea el `kustomization.yaml` usando el siguiente comando donde *<bundle.yaml>* es `bundle_pre_1_25.yaml` o `bundle_post_1_25.yaml` según tu versión de Kubernetes.

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

  b. Compila el paquete usando el siguiente comando donde *<bundle.yaml>* es `bundle_pre_1_25.yaml` o `bundle_post_1_25.yaml` según tu versión de Kubernetes.

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

Pasos

1. Crea los recursos y despliega el operador:

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

2. Verifica que se hayan creado el operador, el deployment y los replicasets.

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

Solo debe haber **una instancia** del operador en un clúster de Kubernetes. No crees varias implementaciones del operador Trident.

Paso 5: actualiza la ubicación del registro de la imagen en la `TridentOrchestrator`

Tu "Imágenes de Trident y CSI" puede estar en un registro o en registros diferentes, pero todas las imágenes CSI deben estar en el mismo registro. Actualiza `deploy/crds/tridentorchestrator_cr.yaml` para agregar las especificaciones de ubicación adicionales según la configuración de tu registro.

Imágenes en un registro

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.02"
tridentImage: "<your-registry>/trident:26.02.0"
```

Imágenes en diferentes registros

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.02"
tridentImage: "<your-registry>/trident:26.02.0"
```

Paso 6: crea el `TridentOrchestrator` e instala Trident

Ahora puedes crear el `TridentOrchestrator` e instalar Trident. Opcionalmente, puedes ampliar aún más "personaliza tu instalación Trident" usando los atributos en la `TridentOrchestrator` spec. El siguiente ejemplo muestra una instalación donde las imágenes de Trident y CSI están ubicadas en diferentes registros.

```
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:26.02
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:26.02.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/trident-autosupport:26.02
    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:26.02.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v26.02.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

Verifica la instalación

Hay varias formas de verificar tu instalación.

Usando `TridentOrchestrator` status

El estado de `TridentOrchestrator` indica si la instalación fue exitosa y muestra la versión de Trident instalada. Durante la instalación, el estado de `TridentOrchestrator` cambia de `Installing` a `Installed`. Si observas el estado de `Failed` y el operador no puede recuperarse por sí solo, "revisa los registros".

Estado	Descripción
Instalando	El operador está instalando Trident usando este `TridentOrchestrator` CR.
Instalado	Trident se ha instalado correctamente.
Desinstalar	El operador está desinstalando Trident porque `spec.uninstall=true`.
Desinstalado	Trident está desinstalado.
Con errores	El operador no pudo instalar, parchear, actualizar o desinstalar Trident; el operador intentará automáticamente recuperarse de este estado. Si este estado persiste necesitarás hacer una solución de problemas.
Actualizando	El operador está actualizando una instalación existente.
Error	El `TridentOrchestrator` no se usa. Ya existe otro.

Uso del estado de creación de pods

Puedes confirmar si la instalación de Trident se completó revisando el estado de los pods creados:

```
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
```

Usando `tridentctl`

Puedes usar `tridentctl` para comprobar la versión de Trident instalada.

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.02.0        | 26.02.0        |
+----------------+----------------+
```

Implementa el operador de Trident usando Helm (modo estándar)

Puedes desplegar el operador Trident e instalar Trident usando Helm. Este proceso aplica a instalaciones donde las imágenes de contenedor requeridas por Trident no están almacenadas en un registro privado. Si tienes un registro de imágenes privado, usa el "proceso para el despliegue fuera de línea".

Información crítica sobre Trident 25.10

Debes leer la siguiente información crítica sobre Trident.

Información crítica sobre Trident

- Kubernetes 1.35 ya es compatible con Trident. Actualiza Trident antes de actualizar Kubernetes.
- Trident impone estrictamente el uso de la configuración de multivía en entornos SAN, con un valor recomendado de `find_multipaths: no` en el archivo multipath.conf.

  El uso de una configuración sin multivía o el uso de `find_multipaths: yes` o `find_multipaths: smart` en el archivo multipath.conf provocará fallos de montaje. Trident ha recomendado el uso de `find_multipaths: no` desde la versión 21.07.

Implementa el operador Trident e instala Trident usando Helm

Usando el "Helm Chart" de Trident, puedes desplegar el operador Trident e instalar Trident en un solo paso.

Revisa "la visión general de la instalación" para asegurarte de que cumpliste los requisitos previos de instalación y seleccionaste la opción de instalación correcta para tu entorno.

Antes de empezar

Además de "requisitos previos de despliegue" tú necesitas "Helm versión 3".

Pasos

1. Agrega el repositorio Trident Helm:

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

2. Usa `helm install` y especifica un nombre para tu despliegue como en el siguiente ejemplo donde `100..0` es la versión de Trident que estás instalando.

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace <trident-namespace>
```

> Si ya creaste un espacio de nombres para Trident, el parámetro `--create-namespace` no creará un espacio de nombres adicional.

Puedes usar `helm list` para revisar detalles de la instalación como nombre, espacio de nombres, chart, estado, versión de la app y número de revisión.

Pasa los datos de configuración durante la instalación

Hay dos formas de pasar datos de configuración durante la instalación:

Opción	Descripción
`--values` (o `-f`)	Especifica un archivo YAML con anulaciones. Esto se puede especificar varias veces y el archivo más a la derecha tendrá prioridad.
`--set`	Especifica las modificaciones en la línea de comandos.

Por ejemplo, para cambiar el valor por defecto de `debug`, ejecuta el siguiente comando donde `100.2602.0` es la versión de Trident que estás instalando:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set tridentDebug=true
```

Opciones de configuración

Esta tabla y el archivo `values.yaml`, que es parte del Helm chart, proporcionan la lista de claves y sus valores predeterminados.

Opción	Descripción	Predeterminado
`nodeSelector`	Etiquetas de nodo para la asignación de pod	

Opción	Descripción	Predeterminado
`podAnnotations`	Anotaciones del pod	
`deploymentAnnotations`	Anotaciones de deployment	
`tolerations`	Tolerancias para la asignación de pods	

`affinity`	Afinidad para la asignación de pods	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedulingIgnoredDuringExecution:`
		`  nodeSelectorTerms:`
		`  - matchExpressions:`
		`  - key: kubernetes.io/arch`
		`  operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key: kubernetes.io/os`
		`  operator: In`
		`  values:`
		`  - linux`
		
		No elimines la afinidad predeterminada del archivo values.yaml. Cuando quieras proporcionar una afinidad personalizada, amplía la afinidad predeterminada.

`tridentControllerPluginNodeSelector`	Selectores de nodos adicionales para pods. Consulta Entender los pods de controlador y los pods de nodo para más detalles.	
`tridentControllerPluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Consulta Entender los pods de controlador y los pods de nodo para más detalles.	

Opción	Descripción	Predeterminado
`tridentNodePluginNodeSelector`	Selectores de nodos adicionales para pods. Consulta Entender los pods de controlador y los pods de nodo para más detalles.	
`tridentNodePluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Consulta Entender los pods de controlador y los pods de nodo para más detalles.	
`imageRegistry`	Identifica el registro para el `trident-operator`, `trident` y otras imágenes. Déjalo vacío para aceptar el valor predeterminado. IMPORTANTE: cuando instales Trident en un repositorio privado, si usas el switch `imageRegistry` para especificar la ubicación del repositorio, no uses `/netapp/` en la ruta del repositorio.	`""`
`imagePullPolicy`	Establece la política de extracción de imágenes para el `trident-operator`.	`IfNotPresent`
`imagePullSecrets`	Establece los secretos de extracción de imágenes para las `trident-operator`, `trident` y otras imágenes.	
`kubeletDir`	Permite anular la ubicación del host del estado interno de kubelet.	`"/var/lib/kubelet"`
`operatorLogLevel`	Permite establecer el nivel de registro del operador Trident en: `trace`, `debug`, `info`, `warn`, `error` o `fatal`.	`"info"`
`operatorDebug`	Permite establecer el nivel de registro del operador Trident en modo debug.	`true`
`operatorImage`	Permite la anulación completa de la imagen para `trident-operator`.	`""`
`operatorImageTag`	Permite anular la etiqueta de la `trident-operator` imagen.	`""`
`tridentIPv6`	Permite habilitar Trident para trabajar en clústeres IPv6.	`false`

Opción	Descripción	Predeterminado
`tridentK8sTimeout`	Anula el tiempo de espera predeterminado de 30 segundos para la mayoría de las operaciones de la API de Kubernetes (si no es cero, en segundos).	`0`
`tridentHttpRequestTimeout`	Anula el tiempo de espera predeterminado de 90 segundos para las solicitudes HTTP, con `0s` siendo una duración infinita para el tiempo de espera. No se permiten valores negativos.	`"90s"`
`tridentSilenceAutosupport`	Permite deshabilitar los informes periódicos de AutoSupport de Trident.	`false`
`tridentAutosupportImageTag`	Permite sobrescribir la etiqueta de la imagen para el contenedor Trident AutoSupport.	`<version>`
`tridentAutosupportProxy`	Permite que el contenedor Trident AutoSupport se comunique con la central a través de un proxy HTTP.	`""`
`tridentLogFormat`	Establece el formato de registro de Trident (`text` o `json`).	`"text"`
`tridentDisableAuditLog`	Desactiva el auditor de registros de Trident.	`true`
`tridentLogLevel`	Permite establecer el nivel de registro de Trident en: `trace`, `debug`, `info`, `warn`, `error` o `fatal`.	`"info"`
`tridentDebug`	Permite establecer el nivel de registro de Trident en `debug`. Puedes automatizar el proceso de desinstalación forzosa mediante la integración con el operador node health check (NHC). Para más información, consulta "Automatizando la conmutación por error de aplicaciones con estado con Trident".	`false`
`tridentLogWorkflows`	Permite habilitar flujos de trabajo específicos de Trident para el registro de trazas o la supresión de logs.	`""`
`tridentLogLayers`	Permite habilitar capas específicas de Trident para el registro de trazas o la supresión de logs.	`""`

Opción	Descripción	Predeterminado
`tridentImage`	Permite la anulación completa de la imagen para Trident.	`""`
`tridentImageTag`	Permite anular la etiqueta de la imagen para Trident.	`""`
`tridentProbePort`	Permite anular el puerto predeterminado usado para las sondas de liveness/readiness de Kubernetes.	`""`
`windows`	Permite que Trident se instale en el nodo trabajador de Windows.	`false`
`enableForceDetach`	Permite activar la función de desconexión forzosa.	`false`
`excludePodSecurityPolicy`	Excluye la política de seguridad del pod de operador de la creación.	`false`
`cloudProvider`	Establécelo en `"Azure"` cuando uses identidades gestionadas o una identidad en la nube en un clúster AKS. Establécelo en "AWS" cuando uses una identidad en la nube en un clúster EKS.	`""`
`cloudIdentity`	Configura la identidad de carga de trabajo ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx") cuando uses cloud identity en un clúster AKS. Configura el rol de AWS IAM ("'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'") cuando uses cloud identity en un clúster EKS.	`""`
`iscsiSelfHealingInterval`	El intervalo en el que se invoca la reparación automática de iSCSI.	`5m0s`
`iscsiSelfHealingWaitTime`	La duración después de la cual la reparación automática iSCSI inicia un intento de resolver una sesión obsoleta realizando un cierre de sesión y un inicio de sesión posterior.	`7m0s`

Opción	Descripción	Predeterminado
`nodePrep`	Permite que Trident prepare los nodos del clúster de Kubernetes para gestionar volúmenes usando el protocolo de almacenamiento de datos especificado. **Actualmente, `iscsi` es el único valor admitido.** Nota: a partir de OpenShift 4.19, la versión mínima de Trident admitida para esta función es 25.06.1.	
`enableConcurrency`	Permite operaciones simultáneas del controlador Trident para mejorar el rendimiento. **Tech Preview**: Esta función es experimental y actualmente admite flujos de trabajo paralelos limitados con los controladores ONTAP-NAS (NFS solo) y ONTAP-SAN (NVMe para unified ONTAP 9), además de la tech preview existente para el controlador ONTAP-SAN (protocolos iSCSI y FCP en unified ONTAP 9).	false
`k8sAPIQPS`	El límite de consultas por segundo (QPS) que usa el controlador mientras se comunica con el servidor de la API de Kubernetes. El valor de ráfaga se establece automáticamente según el valor de QPS.	`100`; opcional

Opción	Descripción	Predeterminado

`resources`	Establece los límites y las solicitudes de recursos de Kubernetes para los pods del controlador, nodo y operador de Trident. Puedes configurar la CPU y la memoria de cada contenedor y sidecar para gestionar la asignación de recursos en Kubernetes.
	
	Para obtener más información sobre cómo configurar las solicitudes y los límites de recursos, consulta "Gestión de recursos para pods y contenedores".
	
	• NO cambies los nombres de ningún contenedor ni de ningún campo.
	• NO cambies la sangría: la sangría de YAML es fundamental para que se analice correctamente.
		<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:
		    linux:
		      trident-main:</pre>

Opción	Descripción	Predeterminado
`httpsMetrics`	Habilita HTTPS para el endpoint de métricas de Prometheus.	false
`hostNetwork`	Habilita la conexión en red del host para el controlador Trident. Esto es útil cuando quieres separar el tráfico de frontend y backend en una red multi-home.	false

e si no se especifican.

- Los nombres de los figuran tal como aparecen en las especificaciones del pod.
- Los sidecars aparecen debajo de cada contenedor principal.
- Revisa el campo `status.currentInstallationParams` del TORC para ver los valores que están aplicados ahora.

```
    requests:
      cpu: 1m
      memory: 10Mi
  linux:
    cpu:
    memory:
windows:
  trident-main:
    requests:
      cpu: 6m
      memory: 40Mi
    limits:
      cpu:
      memory:
  node-driver-registrar:
    requests:
      cpu: 6m
      memory: 40Mi
    limits:
      cpu:
      memory:
  liveness-probe:
    requests:
      cpu: 2m
      memory: 40Mi
    limits:
      cpu:
```

Entender los pods de controlador y los pods de nodo

Trident se ejecuta como un pod controlador único, más un pod de nodo en cada nodo trabajador del clúster. El pod de nodo debe estar ejecutándose en cualquier host donde quieras montar potencialmente un volumen Trident.

Kubernetes "selectores de nodos" y "tolerancias y taints" se usan para restringir que un pod se ejecute en un nodo específico o preferido. Usando `ControllerPlugin` y `NodePlugin`, puedes especificar restricciones y anulaciones.

- El complemento del controlador se encarga del aprovisionamiento y la gestión de volúmenes, como las instantáneas y el cambio de tamaño.
- El plugin de nodo se encarga de conectar el almacenamiento al nodo.

Implementa el operador de Trident usando Helm (modo sin conexión)

Puedes desplegar el operador Trident e instalar Trident usando Helm. Este proceso aplica a instalaciones donde las imágenes de contenedor requeridas por Trident se almacenan en un registro privado. Si no tienes un registro de imágenes privado, usa el "proceso para la implementación estándar".

Información crítica sobre Trident 26.02

Debes leer la siguiente información crítica sobre Trident.

Información crítica sobre Trident

- Kubernetes 1.35 ya es compatible con Trident. Actualiza Trident antes de actualizar Kubernetes.
- Trident impone estrictamente el uso de la configuración de multivía en entornos SAN, con un valor recomendado de `find_multipaths: no` en el archivo multipath.conf.

  El uso de una configuración sin multivía o el uso de `find_multipaths: yes` o `find_multipaths: smart` en el archivo multipath.conf provocará fallos de montaje. Trident ha recomendado el uso de `find_multipaths: no` desde la versión 21.07.

Implementa el operador Trident e instala Trident usando Helm

Usando el "Helm Chart" de Trident, puedes desplegar el operador Trident e instalar Trident en un solo paso.

Revisa "la visión general de la instalación" para asegurarte de que cumpliste los requisitos previos de instalación y seleccionaste la opción de instalación correcta para tu entorno.

Antes de empezar

Además de "requisitos previos de despliegue" tú necesitas "Helm versión 3".

> Cuando instales Trident en un repositorio privado, si estás usando el modificador `imageRegistry` para especificar la ubicación del repositorio, no uses `/netapp/` en la ruta del repositorio.

Pasos

1. Agrega el repositorio Trident Helm:

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

2. Usa `helm install` y especifica un nombre para tu deployment y la ubicación del image registry. Tu "Imágenes de Trident y CSI" puede estar en un registro o en registros diferentes, pero todas las imágenes CSI deben estar en el mismo registro. En los ejemplos, `100.2602.0` es la versión de Trident que estás instalando.

Imágenes en un registro

```
helm install <name> netapp-trident/trident-operator --version
100.2602.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

Imágenes en diferentes registros

```
helm install <name> netapp-trident/trident-operator --version
100.2602.0 --set imageRegistry=<your-registry> --set operatorImage
=<your-registry>/trident-operator:26.02.0 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:26.02
--set tridentImage=<your-registry>/trident:26.02.0 --create
-namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```

> Si ya creaste un espacio de nombres para Trident, el parámetro `--create-namespace` no creará un espacio de nombres adicional.

Puedes usar `helm list` para revisar detalles de la instalación como nombre, espacio de nombres, chart, estado, versión de la app y número de revisión.

Pasa los datos de configuración durante la instalación

Hay dos formas de pasar datos de configuración durante la instalación:

Opción	Descripción
`--values` (o `-f`)	Especifica un archivo YAML con anulaciones. Esto se puede especificar varias veces y el archivo más a la derecha tendrá prioridad.
`--set`	Especifica las modificaciones en la línea de comandos.

Por ejemplo, para cambiar el valor por defecto de `debug`, ejecuta el siguiente comando donde `100.2602.0` es la versión de Trident que estás instalando:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set tridentDebug=true
```

Para añadir el valor nodePrep, ejecuta el siguiente comando:

```
helm install <name> netapp-trident/trident-operator --version 100.2602.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

Opciones de configuración

Esta tabla y el archivo `values.yaml`, que es parte del Helm chart, proporcionan la lista de claves y sus valores predeterminados.

> No elimines la afinidad predeterminada del archivo values.yaml. Cuando quieras proporcionar una afinidad personalizada, amplía la afinidad predeterminada.

Opción	Descripción	Predeterminado
`nodeSelector`	Etiquetas de nodo para la asignación de pod	
`podAnnotations`	Anotaciones del pod	
`deploymentAnnotations`	Anotaciones de deployment	
`tolerations`	Tolerancias para la asignación de pods	

Opción	Descripción	Predeterminado

`affinity`	Afinidad para la asignación de pods	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedul`
		`ingIgnoredDuringExecu`
		`tion:`
		
		`nodeSelectorTerms:`
		`  -`
		`matchExpressions:`
		`  - key:`
		`kubernetes.io/arch`
		
		`operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key:`
		`kubernetes.io/os`
		
		`operator: In`
		`  values:`
		`  - linux`
		
		No elimines la afinidad predeterminada del archivo values.yaml. Cuando quieras proporcionar una afinidad personalizada, amplía la afinidad predeterminada.

`tridentControllerPluginNodeSelector`	Selectores de nodos adicionales para pods. Consulta "Entender los pods de controlador y los pods de nodo" para más detalles.	
`tridentControllerPluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Consulta "Entender los pods de controlador y los pods de nodo" para más detalles.	

Opción	Descripción	Predeterminado
`tridentNodePluginNodeSelector`	Selectores de nodos adicionales para pods. Consulta "Entender los pods de controlador y los pods de nodo" para más detalles.	
`tridentNodePluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Consulta "Entender los pods de controlador y los pods de nodo" para más detalles.	
`imageRegistry`	Identifica el registro para el `trident-operator`, `trident` y otras imágenes. Déjalo vacío para aceptar el valor predeterminado. IMPORTANTE: cuando instales Trident en un repositorio privado, si usas el switch `imageRegistry` para especificar la ubicación del repositorio, no uses `/netapp/` en la ruta del repositorio.	""
`imagePullPolicy`	Establece la política de extracción de imágenes para el `trident-operator`.	`IfNotPresent`
`imagePullSecrets`	Establece los secretos de extracción de imágenes para las `trident-operator`, `trident` y otras imágenes.	
`kubeletDir`	Permite anular la ubicación del host del estado interno de kubelet.	`"/var/lib/kubelet"`
`operatorLogLevel`	Permite establecer el nivel de registro del operador Trident en: `trace`, `debug`, `info`, `warn`, `error` o `fatal`.	`"info"`
`operatorDebug`	Permite establecer el nivel de registro del operador Trident en modo debug.	`true`
`operatorImage`	Permite la anulación completa de la imagen para `trident-operator`.	""
`operatorImageTag`	Permite anular la etiqueta de la `trident-operator` imagen.	""
`tridentIPv6`	Permite habilitar Trident para trabajar en clústeres IPv6.	`false`

Opción	Descripción	Predeterminado
`tridentK8sTimeout`	Anula el tiempo de espera predeterminado de 180 segundos para la mayoría de las operaciones de la API de Kubernetes (si no es cero, en segundos). El `tridentK8sTimeout` parámetro solo es aplicable para la instalación de Trident.	`180`
`tridentHttpRequestTimeout`	Anula el tiempo de espera predeterminado de 90 segundos para las solicitudes HTTP, con `0s` siendo una duración infinita para el tiempo de espera. No se permiten valores negativos.	`"90s"`
`tridentSilenceAutosupport`	Permite deshabilitar los informes periódicos de AutoSupport de Trident.	`false`
`tridentAutosupportImageTag`	Permite sobrescribir la etiqueta de la imagen para el contenedor Trident AutoSupport.	`<version>`
`tridentAutosupportProxy`	Permite que el contenedor Trident AutoSupport se comunique con la central a través de un proxy HTTP.	""
`tridentLogFormat`	Establece el formato de registro de Trident (`text` o `json`).	`"text"`
`tridentDisableAuditLog`	Desactiva el auditor de registros de Trident.	`true`
`tridentLogLevel`	Permite establecer el nivel de registro de Trident en: `trace`, `debug`, `info`, `warn`, `error` o `fatal`.	`"info"`
`tridentDebug`	Permite establecer el nivel de registro de Trident en `debug`.	`false`
`tridentLogWorkflows`	Permite habilitar flujos de trabajo específicos de Trident para el registro de trazas o la supresión de logs.	""
`tridentLogLayers`	Permite habilitar capas específicas de Trident para el registro de trazas o la supresión de logs.	""

Opción	Descripción	Predeterminado
`tridentImage`	Permite la anulación completa de la imagen para Trident.	""
`tridentImageTag`	Permite anular la etiqueta de la imagen para Trident.	""
`tridentProbePort`	Permite anular el puerto predeterminado usado para las sondas de liveness/readiness de Kubernetes.	""
`windows`	Permite que Trident se instale en el nodo trabajador de Windows.	`false`
`enableForceDetach`	Permite activar la función de desconexión forzosa. Puedes automatizar el proceso de desinstalación forzosa mediante la integración con el operador node health check (NHC). Para más información, consulta "Automatizando la conmutación por error de aplicaciones con estado con Trident".	`false`
`excludePodSecurityPolicy`	Excluye la política de seguridad del pod de operador de la creación.	`false`
`nodePrep`	Permite que Trident prepare los nodos del clúster de Kubernetes para gestionar volúmenes usando el protocolo de almacenamiento de datos especificado. **Actualmente, `iscsi` es el único valor admitido.** A partir de OpenShift 4.19, la versión mínima de Trident compatible con esta función es 25.06.1.	

Opción	Descripción	Predeterminado

`resources`	Establece los límites y las solicitudes de recursos de Kubernetes para los pods del controlador, nodo y operador de Trident. Puedes configurar la CPU y la memoria de cada contenedor y sidecar para gestionar la asignación de recursos en Kubernetes.
	
	Para obtener más información sobre cómo configurar las solicitudes y los límites de recursos, consulta "Gestión de recursos para pods y contenedores".
	
	• NO cambies los nombres de ningún contenedor ni de ningún campo.
	• NO cambies la sangría: la sangría de YAML es fundamental para que se analice correctamente.
		<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:
		    linux:</pre>

Personaliza la instalación del operador Trident

El operador Trident te permite personalizar la instalación de Trident usando los atributos en la `TridentOrchestrator`spec. Si quieres personalizar la instalación más allá de lo que permiten los `TridentOrchestrator`arguments, considera usar `tridentctl`para generar manifiestos YAML personalizados y modificarlos como necesites.

Entender los pods de controlador y los pods de nodo

Trident se ejecuta como un único pod de controlador y un pod de nodo en cada nodo de trabajo del clúster. El pod de nodo debe estar ejecutándose en cualquier host donde quieras montar potencialmente un volumen Trident.

Kubernetes "selectores de nodos" y "tolerancias y taints" se usan para restringir que un pod se ejecute en un nodo específico o preferido. Usando `ControllerPlugin` y `NodePlugin`, puedes especificar restricciones y anulaciones.

- El complemento del controlador se encarga del aprovisionamiento y la gestión de volúmenes, como las instantáneas y el cambio de tamaño.
- El plugin de nodo se encarga de conectar el almacenamiento al nodo.

Opciones de configuración

`spec.namespace` se especifica en `TridentOrchestrator` para indicar el namespace donde está instalado Trident. Este parámetro **no se puede actualizar después de instalar Trident**. Si intentas hacerlo, el estado de `TridentOrchestrator` cambia a `Failed`. Trident no está pensado para migrarse entre namespaces.

Esta tabla detalla `TridentOrchestrator` los atributos.

Parámetro	Descripción	Predeterminado
`namespace`	Espacio de nombres para instalar Trident	`"default"`
`debug`	Habilita la depuración para Trident	`false`
`enableForceDetach`	`ontap-san`, `ontap-san-economy`, `ontap-nas` y `ontap-nas-economy` solamente. Funciona con Kubernetes Non-Graceful Node Shutdown (NGNS) para que los administradores de clúster tengan la capacidad de migrar de forma segura cargas de trabajo con volúmenes montados a nuevos nodos si un nodo deja de estar saludable. Para más información, consulta "Automatizando la conmutación por error de aplicaciones con estado con Trident".	`false`
`windows`	La configuración de `true` permite la instalación en nodos worker de Windows.	`false`

- No se aplican límites por defecto: sólo las solicitudes tienen valores por defecto.
- Los contenedores de los contenedores figuran tal como aparecen en las especificaciones del pod.
- Los sidecars aparecen debajo de cada contenedor principal.
- Revisa el campo `status.CurrentInstallationParams` del TORC para ver los valores que están aplicados ahora.

```
      trident-main:
        requests:
          cpu: 10m
          memory:
60Mi
        limits:
          cpu:
          memory:
      node-driver-
registrar:
        requests:
          cpu: 1m
          memory:
10Mi
        limits:
          cpu:
          memory:
    windows:
      trident-main:
        requests:
          cpu: 6m
          memory:
40Mi
        limits:
          cpu:
          memory:
      node-driver-
registrar:
        requests:
          cpu:
          memory:
  operator:
    requests:
      cpu: 10m
      memory: 40Mi
    limits:
```

Parámetro	Descripción	Predeterminado
`cloudProvider`	Establécelo en `"Azure"` cuando uses identidades gestionadas o una identidad en la nube en un clúster AKS. Configúralo en `"AWS"` cuando uses una identidad en la nube en un clúster de EKS. Configúralo en `"GCP"` cuando uses una identidad en la nube en un clúster de GKE.	`""`
`cloudIdentity`	Configura la identidad de carga de trabajo ("azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx") cuando uses cloud identity en un clúster AKS. Configura el rol de AWS IAM ("'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'") cuando uses cloud identity en un clúster EKS. Configura la cloud identity ("'iam.gke.io/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com'") cuando uses cloud identity en un clúster de GKE.	`""`
`IPv6`	Instala Trident sobre IPv6	false
`k8sTimeout`	Tiempo de espera para las operaciones de Kubernetes. El `k8sTimeout` parámetro solo es aplicable para la instalación de Trident.	`180sec`
`silenceAutosupport`	No envíes paquetes de autosupport a NetApp automáticamente	`false`
`autosupportImage`	La imagen del contenedor para Autosupport Telemetry	`"netapp/trident-autosupport10"`
`autosupportProxy`	La dirección/puerto de un proxy para enviar Autosupport Telemetry	`"http://proxy.example.com:8888"`
`uninstall`	Una bandera usada para desinstalar Trident	`false`
`logFormat`	Formato de registro de Trident que se usará [text,json]	`"text"`
`tridentImage`	Imagen de Trident para instalar	`"netapp/trident:26.02"`
`imageRegistry`	Ruta al registro interno, de formato `<registry FQDN>[:port][/subpath]`	`"registry.k8s.io"`
`kubeletDir`	Ruta al directorio de kubelet en el host	`"/var/lib/kubelet"`
`wipeout`	Una lista de recursos para eliminar y así realizar una eliminación completa de Trident	
`imagePullSecrets`	Secretos para extraer imágenes de un registro interno	

Parámetro	Descripción	Predeterminado
`imagePullPolicy`	Establece la política de extracción de imágenes para el operador Trident. Los valores válidos son: `Always` para extraer siempre la imagen. `IfNotPresent` para extraer la imagen solo si no existe en el nodo. `Never` para nunca extraer la imagen.	`IfNotPresent`
`controllerPluginNodeSelector`	Selectores de nodos adicionales para pods. Sigue el mismo formato que `pod.spec.nodeSelector`.	Sin valor predeterminado; opcional
`controllerPluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Sigue el mismo formato que `pod.spec.Tolerations`.	Sin valor predeterminado; opcional
`nodePluginNodeSelector`	Selectores de nodos adicionales para pods. Sigue el mismo formato que `pod.spec.nodeSelector`.	Sin valor predeterminado; opcional
`nodePluginTolerations`	Anula las tolerancias de Kubernetes para los pods. Sigue el mismo formato que `pod.spec.Tolerations`.	Sin valor predeterminado; opcional
`nodePrep`	Permite que Trident prepare los nodos del clúster de Kubernetes para gestionar volúmenes usando el protocolo de almacenamiento de datos especificado. **Actualmente, `iscsi` es el único valor admitido.** A partir de OpenShift 4.19, la versión mínima de Trident compatible con esta función es 25.06.1.	
`k8sAPIQPS`	El límite de consultas por segundo (QPS) que usa el controlador mientras se comunica con el servidor de la API de Kubernetes. El valor de ráfaga se establece automáticamente según el valor de QPS.	`100`; opcional
`enableConcurrency`	Permite operaciones simultáneas del controlador Trident para mejorar el rendimiento. **Tech Preview**: Esta función es experimental y actualmente admite flujos de trabajo paralelos limitados con los controladores ONTAP-NAS (NFS solo) y ONTAP-SAN (NVMe para unified ONTAP 9), además de la tech preview existente para el controlador ONTAP-SAN (protocolos iSCSI y FCP en unified ONTAP 9).	false

Parámetro	Descripción	Predeterminado

`resources`	Establece los límites y las solicitudes de recursos de Kubernetes para el controlador Trident y los pods de nodo. Puedes configurar la CPU y la memoria de cada contenedor y sidecar para gestionar la asignación de recursos en Kubernetes.
	
	Para obtener más información sobre cómo configurar las solicitudes y los límites de recursos, consulta "Gestión de recursos para pods y contenedores".
	
	• NO cambies los nombres de ningún contenedor ni de ningún campo.
	• NO cambies la sangría: la sangría de YAML es fundamental para que se analice correctamente.
	
	• No se aplican límites por defecto: solo las solicitudes tienen valores por defecto y se aplican automáticamente si no se especifican.
	• Los nombres de los contenedores figuran tal como aparecen en las especificaciones del pod.
	• Los sidecars aparecen debajo de cada contenedor principal.
	• Revisa el campo `status.CurrentInstallationParams` del TORC para ver los valores que están aplicados ahora.
		<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:</pre>

Parámetro	Descripción	Predeterminado
`httpsMetrics`	Habilita HTTPS para el endpoint de métricas de Prometheus.	false
`hostNetwork`	Habilita la conexión en red del host para el controlador Trident. Esto es útil cuando quieres separar el tráfico de frontend y backend en una red multi-home.	false

Para más información sobre cómo formatear los parámetros del pod, consulta "Asignar pods a nodos".

Configuraciones de ejemplo

Puedes usar los atributos en Opciones de configuración cuando defines `TridentOrchestrator` para personalizar tu instalación.

Configuración básica personalizada

Este ejemplo, creado después de ejecutar el `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml` comando, representa una instalación personalizada básica:

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

Selectores de nodos

Este ejemplo instala Trident con selectores de nodos.

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

Nodos trabajadores de Windows

Este ejemplo, creado después de ejecutar el `cat deploy/crds/tridentorchestrator_cr.yaml` comando, instala Trident en un nodo trabajador Windows.

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

Identidades gestionadas en un clúster de AKS

Este ejemplo instala Trident para habilitar identidades gestionadas en un clúster de AKS.

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

Cloud identity en un clúster AKS

Este ejemplo instala Trident para usarlo con una identidad en la nube en un clúster AKS.

```
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-xxxxxxxxxxx'
```

Cloud identity en un clúster EKS

Este ejemplo instala Trident para usarlo con una identidad en la nube en un clúster AKS.

```
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'"
```

Identidad en la nube para GKE

Este ejemplo instala Trident para usarlo con una identidad en la nube en un clúster GKE.

```
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
```

Configuración de solicitudes y límites de recursos de Kubernetes para el controlador Trident y los pods de nodos Linux Trident

Este ejemplo configura las solicitudes y los límites de recursos de Kubernetes para el controlador Trident y los pods de nodo Linux Trident.

> **Descargo de responsabilidad**: Los valores de solicitud y límite proporcionados en este ejemplo son solo para fines de demostración. Ajusta estos valores según tu entorno y los requisitos de carga de trabajo.

```
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
```

Configuración de solicitudes y límites de recursos de Kubernetes para el controlador Trident y los pods de nodo Trident Windows y Linux

Este ejemplo configura las solicitudes y los límites de recursos de Kubernetes para el controlador Trident y los pods de nodo Trident Windows y Linux.

Descargo de responsabilidad: Los valores de solicitud y límite proporcionados en este ejemplo son solo para fines de demostración. Ajusta estos valores según tu entorno y los requisitos de carga de trabajo.

```
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
```