Installation mit dem Trident operator

Trident

NetApp
July 01, 2026

Inhalt

Installation mit dem Trident operator

Manuelle Bereitstellung des Trident Operators (Standardmodus)

Sie können den Trident-Operator manuell bereitstellen, um Trident zu installieren. Dieser Prozess gilt für Installationen, bei denen die von Trident benötigten Container-Images nicht in einer privaten Registry gespeichert sind. Wenn Sie eine private Image-Registry haben, verwenden Sie die "Prozess für die Offline-Bereitstellung".

Wichtige Informationen zu Trident 26.02

Sie müssen die folgenden wichtigen Informationen über Trident lesen.

Wichtige Informationen zu Trident

- Kubernetes 1.35 wird nun in Trident unterstützt. Trident sollte vor Kubernetes aktualisiert werden.
- Trident setzt die Verwendung der Multipathing-Konfiguration in SAN-Umgebungen strikt durch, mit einem empfohlenen Wert von `find_multipaths: no` in der multipath.conf-Datei.

  Die Verwendung einer Konfiguration ohne Multipathing oder die Verwendung von `find_multipaths: yes` oder `find_multipaths: smart` Wert in der Datei multipath.conf führt zu Mount-Fehlern. Trident hat die Verwendung von `find_multipaths: no` seit der Version 21.07 empfohlen.

Den Trident-Operator manuell bereitstellen und Trident installieren

Überprüfen Sie "die Installationsübersicht", um sicherzustellen, dass Sie die Installationsvoraussetzungen erfüllt und die richtige Installationsoption für Ihre Umgebung ausgewählt haben.

Bevor Sie beginnen

Bevor Sie mit der Installation beginnen, melden Sie sich am Linux-Host an und vergewissern Sie sich, dass dieser ein funktionierendes "unterstützter Kubernetes-Cluster" verwaltet und dass Sie über die erforderlichen Berechtigungen verfügen.

Mit OpenShift verwenden Sie `oc` statt `kubectl` in allen folgenden Beispielen und melden Sie sich zuerst als **system:admin** an, indem Sie `oc login -u system:admin` oder `oc login -u kube-admin` ausführen.

1. Überprüfen Sie Ihre Kubernetes-Version:

```
kubectl version
```

2. Überprüfen Sie die Cluster-Administratorrechte:

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

3. Überprüfen Sie, ob Sie einen Pod starten können, der ein Image von Docker Hub verwendet, und Ihr Speichersystem über das Pod-Netzwerk erreichen können.

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

Schritt 1: Laden Sie das Trident-Installationspaket herunter

Das Trident-Installationspaket enthält alles, was Sie zum Bereitstellen des Trident-Operators und zum Installieren von Trident benötigen. Laden Sie die neueste Version des Trident-Installationsprogramms von "der Abschnitt *Assets* auf GitHub" herunter und extrahieren Sie sie.

```
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
```

Schritt 2: Erstellen Sie die `TridentOrchestrator` CRD

Erstellen Sie die `TridentOrchestrator` benutzerdefinierte Ressourcendefinition (CRD). Sie werden später eine `TridentOrchestrator` benutzerdefinierte Ressource erstellen. Verwenden Sie die entsprechende CRD-YAML-Version in `deploy/crds`, um die `TridentOrchestrator` CRD zu erstellen.

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

Schritt 3: Den Trident Operator einsetzen

Das Trident-Installationsprogramm stellt eine Bundle-Datei bereit, mit der der Operator installiert und zugehörige Objekte erstellt werden können. Die Bundle-Datei ist eine einfache Möglichkeit, den Operator bereitzustellen und Trident mit einer Standardkonfiguration zu installieren.

- Für Cluster, auf denen Kubernetes 1.24 läuft, verwenden Sie `bundle_pre_1_25.yaml`.

- Für Cluster, auf denen Kubernetes 1.25 oder höher läuft, verwenden Sie `bundle_post_1_25.yaml`.

Bevor Sie beginnen

- Standardmäßig installiert das Trident-Installationsprogramm den Operator im `trident` Namespace. Falls der `trident` Namespace nicht existiert, erstellen Sie ihn mit:

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

- Um den Operator in einem anderen Namespace als dem `trident` Namespace bereitzustellen, aktualisieren Sie `serviceaccount.yaml`, `clusterrolebinding.yaml` und `operator.yaml` und generieren Sie Ihre Bundle-Datei mit dem `kustomization.yaml`.

  a. Erstellen Sie das `kustomization.yaml` mit dem folgenden Befehl, wobei *<bundle.yaml>* `bundle_pre_1_25.yaml` oder `bundle_post_1_25.yaml` abhängig von Ihrer Kubernetes-Version ist.

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

  b. Kompilieren Sie das Bundle mit dem folgenden Befehl, wobei *<bundle.yaml>* `bundle_pre_1_25.yaml` oder `bundle_post_1_25.yaml` entsprechend Ihrer Kubernetes-Version ist.

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

Schritte

1. Erstellen Sie die Ressourcen und stellen Sie den Operator bereit:

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

2. Überprüfen Sie, ob der Operator, die Bereitstellung und die ReplicaSets erstellt wurden.

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

Es sollte nur **eine Instanz** des Operators in einem Kubernetes-Cluster geben. Erstellen Sie nicht mehrere Deployments des Trident Operators.

Schritt 4: Erstellen Sie das `TridentOrchestrator` und installieren Sie Trident

Sie können jetzt die `TridentOrchestrator` erstellen und Trident installieren. Optional können Sie "Passen Sie Ihre Trident Installation individuell an" mit den Attributen in der `TridentOrchestrator`-Spezifikation arbeiten.

```
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
```

Überprüfen Sie die Installation

Es gibt mehrere Möglichkeiten, Ihre Installation zu überprüfen.

Status mit `TridentOrchestrator` verwenden

Der Status von `TridentOrchestrator` zeigt an, ob die Installation erfolgreich war und zeigt die installierte Version von Trident an. Während der Installation ändert sich der Status von `TridentOrchestrator` `Installing` zu `Installed`. Wenn Sie den `Failed` Status beobachten und der Operator das Problem nicht selbst beheben kann, "Überprüfen Sie die Protokolle".

Status	Beschreibung
Installation	Der Operator installiert Trident mit diesem `TridentOrchestrator` CR.
Installiert	Trident wurde erfolgreich installiert.
Deinstallation	Der Betreiber deinstalliert Trident, weil `spec.uninstall=true`.
Deinstalliert	Trident ist deinstalliert.
Fehlgeschlagen	Der Operator konnte Trident nicht installieren, patchen, aktualisieren oder deinstallieren; der Operator wird automatisch versuchen, sich von diesem Zustand zu erholen. Wenn dieser Zustand weiterhin besteht, ist eine Fehlerbehebung erforderlich.
Aktualisierung	Der Betreiber aktualisiert eine bestehende Installation.
Fehler	Das `TridentOrchestrator` wird nicht verwendet. Ein anderer existiert bereits.

Verwendung des Pod-Erstellungsstatus

Sie können bestätigen, ob die Trident Installation abgeschlossen wurde, indem Sie den Status der erstellten Pods überprüfen:

```
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
```

Verwendung `tridentctl`

Sie können `tridentctl` verwenden, um die installierte Version von Trident zu überprüfen.

```
./tridentctl -n trident version

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

Manuelle Bereitstellung des Trident Operators (Offline-Modus)

Sie können den Trident-Operator manuell bereitstellen, um Trident zu installieren. Dieser Vorgang gilt für Installationen, bei denen die von Trident benötigten Container-Images in einer privaten Registry gespeichert sind. Wenn Sie keine private Image-Registry besitzen, verwenden Sie die "Prozess für die Standardbereitstellung".

Wichtige Informationen zu Trident

Sie müssen die folgenden wichtigen Informationen über Trident lesen.

Wichtige Informationen zu Trident

- Kubernetes 1.35 wird nun in Trident unterstützt. Trident sollte vor Kubernetes aktualisiert werden.
- Trident setzt die Verwendung der Multipathing-Konfiguration in SAN-Umgebungen strikt durch, mit einem empfohlenen Wert von `find_multipaths: no` in der multipath.conf-Datei.

  Die Verwendung einer Konfiguration ohne Multipathing oder die Verwendung von `find_multipaths: yes` oder `find_multipaths: smart` Wert in der Datei multipath.conf führt zu Mount-Fehlern. Trident hat die Verwendung von `find_multipaths: no` seit der Version 21.07 empfohlen.

Den Trident-Operator manuell bereitstellen und Trident installieren

Überprüfen Sie "die Installationsübersicht", um sicherzustellen, dass Sie die Installationsvoraussetzungen erfüllt und die richtige Installationsoption für Ihre Umgebung ausgewählt haben.

Bevor Sie beginnen

Melden Sie sich am Linux-Host an und vergewissern Sie sich, dass er ein funktionierendes und "unterstützter Kubernetes-Cluster" verwaltet und dass Sie über die erforderlichen Berechtigungen verfügen.

Mit OpenShift verwenden Sie `oc` statt `kubectl` in allen folgenden Beispielen und melden Sie sich zuerst als **system:admin** an, indem Sie `oc login -u system:admin` oder `oc login -u kube-admin` ausführen.

1. Überprüfen Sie Ihre Kubernetes-Version:

```
kubectl version
```

2. Überprüfen Sie die Cluster-Administratorrechte:

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

3. Überprüfen Sie, ob Sie einen Pod starten können, der ein Image von Docker Hub verwendet, und Ihr Speichersystem über das Pod-Netzwerk erreichen können.

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

Schritt 1: Laden Sie das Trident-Installationspaket herunter

Das Trident-Installationspaket enthält alles, was Sie zum Bereitstellen des Trident-Operators und zum Installieren von Trident benötigen. Laden Sie die neueste Version des Trident-Installationsprogramms von "der Abschnitt *Assets* auf GitHub" herunter und extrahieren Sie sie.

```
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
```

Schritt 2: Erstellen Sie die `TridentOrchestrator` CRD

Erstellen Sie die `TridentOrchestrator` benutzerdefinierte Ressourcendefinition (CRD). Sie werden später eine `TridentOrchestrator` benutzerdefinierte Ressource erstellen. Verwenden Sie die entsprechende CRD-YAML-Version in `deploy/crds`, um die `TridentOrchestrator` CRD zu erstellen:

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

Schritt 3: Aktualisieren Sie den Registrierungsspeicherort im Operator

In `/deploy/operator.yaml` `image: docker.io/netapp/trident-operator:26.02.0`aktualisieren Sie den Pfad, um den Speicherort Ihrer Image-Registry widerzuspiegeln. Ihre "Trident und CSI images" können sich in einer

Registry oder in verschiedenen Registrys befinden, aber alle CSI-Images müssen sich in derselben Registry befinden. Beispiel:

- `image: <your-registry>/trident-operator:26.02.0` wenn sich alle Ihre Images in einer einzigen Registry befinden.
- `image: <your-registry>/netapp/trident-operator:26.02.0`wenn sich Ihr Trident-Image in einem anderen Registry als Ihre CSI-Images befindet.

Schritt 4: Den Trident Operator einsetzen

Das Trident-Installationsprogramm stellt eine Bundle-Datei bereit, mit der der Operator installiert und zugehörige Objekte erstellt werden können. Die Bundle-Datei ist eine einfache Möglichkeit, den Operator bereitzustellen und Trident mit einer Standardkonfiguration zu installieren.

- Für Cluster, auf denen Kubernetes 1.24 läuft, verwenden Sie `bundle_pre_1_25.yaml`.
- Für Cluster, auf denen Kubernetes 1.25 oder höher läuft, verwenden Sie `bundle_post_1_25.yaml`.

Bevor Sie beginnen

- Standardmäßig installiert das Trident-Installationsprogramm den Operator im `trident` Namespace. Falls der `trident` Namespace nicht existiert, erstellen Sie ihn mit:

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

- Um den Operator in einem anderen Namespace als dem `trident` Namespace bereitzustellen, aktualisieren Sie `serviceaccount.yaml`, `clusterrolebinding.yaml` und `operator.yaml` und generieren Sie Ihre Bundle-Datei mit dem `kustomization.yaml`.

  a. Erstellen Sie das `kustomization.yaml` mit dem folgenden Befehl, wobei *<bundle.yaml>* `bundle_pre_1_25.yaml` oder `bundle_post_1_25.yaml` abhängig von Ihrer Kubernetes-Version ist.

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

  b. Kompilieren Sie das Bundle mit dem folgenden Befehl, wobei *<bundle.yaml>* `bundle_pre_1_25.yaml` oder `bundle_post_1_25.yaml` entsprechend Ihrer Kubernetes-Version ist.

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

Schritte

1. Erstellen Sie die Ressourcen und stellen Sie den Operator bereit:

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

2. Überprüfen Sie, ob der Operator, die Bereitstellung und die ReplicaSets erstellt wurden.

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

Es sollte nur **eine Instanz** des Operators in einem Kubernetes-Cluster geben. Erstellen Sie nicht mehrere Deployments des Trident Operators.

Schritt 5: Aktualisieren Sie den Speicherort der Bildregistrierung in der `TridentOrchestrator`

Ihre "Trident und CSI images" können sich in einer Registry oder in verschiedenen Registrys befinden, aber alle CSI-Images müssen sich in derselben Registry befinden. Aktualisieren Sie `deploy/crds/tridentorchestrator_cr.yaml` , um die zusätzlichen Speicherortangaben entsprechend Ihrer Registry-Konfiguration hinzuzufügen.

Images in einem Registry

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

Images in verschiedenen Registern

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

Schritt 6: Erstellen Sie das `TridentOrchestrator` und installieren Sie Trident

Sie können jetzt die `TridentOrchestrator` erstellen und Trident installieren. Optional können Sie "Passen Sie Ihre Trident Installation individuell an" mithilfe der Attribute in der `TridentOrchestrator`-Spezifikation weiter anpassen. Das folgende Beispiel zeigt eine Installation, bei der sich Trident- und CSI-Images in unterschiedlichen Registries befinden.

```
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
```

Überprüfen Sie die Installation

Es gibt mehrere Möglichkeiten, Ihre Installation zu überprüfen.

Status mit `TridentOrchestrator` verwenden

Der Status von `TridentOrchestrator` zeigt an, ob die Installation erfolgreich war und zeigt die installierte Version von Trident an. Während der Installation ändert sich der Status von `TridentOrchestrator` `Installing` zu `Installed`. Wenn Sie den `Failed` Status beobachten und der Operator das Problem nicht selbst beheben kann, "Überprüfen Sie die Protokolle".

Status	Beschreibung
Installation	Der Operator installiert Trident mit diesem `TridentOrchestrator` CR.
Installiert	Trident wurde erfolgreich installiert.
Deinstallation	Der Betreiber deinstalliert Trident, weil `spec.uninstall=true`.
Deinstalliert	Trident ist deinstalliert.
Fehlgeschlagen	Der Operator konnte Trident nicht installieren, patchen, aktualisieren oder deinstallieren; der Operator wird automatisch versuchen, sich von diesem Zustand zu erholen. Wenn dieser Zustand weiterhin besteht, ist eine Fehlerbehebung erforderlich.
Aktualisierung	Der Betreiber aktualisiert eine bestehende Installation.
Fehler	Das `TridentOrchestrator` wird nicht verwendet. Ein anderer existiert bereits.

Verwendung des Pod-Erstellungsstatus

Sie können bestätigen, ob die Trident Installation abgeschlossen wurde, indem Sie den Status der erstellten Pods überprüfen:

```
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
```

Verwendung `tridentctl`

Sie können `tridentctl` verwenden, um die installierte Version von Trident zu überprüfen.

```
./tridentctl -n trident version

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

Trident Operator mit Helm bereitstellen (Standardmodus)

Sie können den Trident-Operator bereitstellen und Trident mithilfe von Helm installieren. Dieser Prozess gilt für Installationen, bei denen die von Trident benötigten Container-Images nicht in einer privaten Registry gespeichert sind. Wenn Sie eine private Image-Registry haben, verwenden Sie die "Prozess für die Offline-Bereitstellung".

Wichtige Informationen zu Trident 25.10

Sie müssen die folgenden wichtigen Informationen über Trident lesen.

Wichtige Informationen zu Trident

- Kubernetes 1.35 wird nun in Trident unterstützt. Trident sollte vor Kubernetes aktualisiert werden.
- Trident setzt die Verwendung der Multipathing-Konfiguration in SAN-Umgebungen strikt durch, mit einem empfohlenen Wert von `find_multipaths: no` in der multipath.conf-Datei.

  Die Verwendung einer Konfiguration ohne Multipathing oder die Verwendung von `find_multipaths: yes` oder `find_multipaths: smart` Wert in der Datei multipath.conf führt zu Mount-Fehlern. Trident hat die Verwendung von `find_multipaths: no` seit der Version 21.07 empfohlen.

Den Trident-Operator bereitstellen und Trident mit Helm installieren

Mit dem Trident "Helm Chart" können Sie den Trident Operator bereitstellen und Trident in einem Schritt installieren.

Überprüfen Sie "die Installationsübersicht", um sicherzustellen, dass Sie die Installationsvoraussetzungen erfüllt und die richtige Installationsoption für Ihre Umgebung ausgewählt haben.

Bevor Sie beginnen

Zusätzlich zu dem "Bereitstellungsvoraussetzungen" benötigen Sie "Helm Version 3".

Schritte

1. Fügen Sie das Trident Helm repository hinzu:

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

2. Verwenden Sie `helm install` und geben Sie einen Namen für Ihre Bereitstellung an, wie im folgenden Beispiel, wobei `100..0` die Version von Trident ist, die Sie installieren.

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

> Wenn Sie bereits einen Namespace für Trident erstellt haben, wird durch den `--create -namespace` Parameter kein zusätzlicher Namespace erstellt.

Sie können `helm list` verwenden, um Installationsdetails wie Name, Namespace, Chart, Status, App-Version und Revisionsnummer zu überprüfen.

Konfigurationsdaten während der Installation übergeben

Es gibt zwei Möglichkeiten, Konfigurationsdaten während der Installation zu übergeben:

Option	Beschreibung
`--values` (oder `-f`)	Geben Sie eine YAML-Datei mit Überschreibungen an. Dies kann mehrfach angegeben werden, und die ganz rechts stehende Datei hat Vorrang.
`--set`	Geben Sie Überschreibungen in der Befehlszeile an.

Um beispielsweise den Standardwert von `debug` zu ändern, führen Sie den folgenden Befehl aus, wobei `100.2602.0` die Version von Trident ist, die Sie installieren:

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

Konfigurationsoptionen

Diese Tabelle und die `values.yaml` Datei, die Teil des Helm-Charts ist, enthalten die Liste der Schlüssel und ihrer Standardwerte.

Option	Beschreibung	Standard
`nodeSelector`	Knotenbezeichnungen für die Pod-Zuweisung	
`podAnnotations`	Pod-Anmerkungen	
`deploymentAnnotations`	Bereitstellungsanmerkungen	
`tolerations`	Toleranzen für die Pod-Zuordnung	

Option	Beschreibung	Standard

`affinity`	Affinität für die Pod-Zuweisung	<pre>affinity:
		  nodeAffinity:
		
		requiredDuringSchedulingIgnoredDuringExecution:
		      nodeSelectorTerms:
		        - matchExpressions:
		            - key: kubernetes.io/arch
		              operator: In
		              values:
		              - arm64
		              - amd64
		            - key: kubernetes.io/os
		              operator: In
		              values:
		              - linux</pre> Entfernen Sie die Standardaffinität nicht aus der Datei values.yaml. Wenn Sie eine benutzerdefinierte Affinität angeben möchten, erweitern Sie die Standardaffinität.

`tridentControllerPluginNodeSelector`	Zusätzliche Knotenselektoren für Pods. Siehe Controller-Pods und Node-Pods verstehen für Details.	
`tridentControllerPluginTolerations`	Überschreibt die Kubernetes-Toleranzen für Pods. Siehe Controller-Pods und Node-Pods verstehen für Details.	
`tridentNodePluginNodeSelector`	Zusätzliche Knotenselektoren für Pods. Siehe Controller-Pods und Node-Pods verstehen für Details.	
`tridentNodePluginTolerations`	Überschreibt die Kubernetes-Toleranzen für Pods. Siehe Controller-Pods und Node-Pods verstehen für Details.	

Option	Beschreibung	Standard
`imageRegistry`	Identifiziert die Registry für die `trident-operator`, `trident` und andere Images. Leer lassen, um den Standardwert zu akzeptieren. WICHTIG: Wenn Sie Trident in einem privaten Repository installieren und den `imageRegistry`-Schalter zur Angabe des Repository-Speicherorts verwenden, verwenden Sie `/netapp/` nicht im Repository-Pfad.	`""`
`imagePullPolicy`	Legt die Image-Pull-Richtlinie für das `trident-operator` fest.	`IfNotPresent`
`imagePullSecrets`	Legt die Image-Pull-Secrets für die `trident-operator`, `trident` und andere Images fest.	
`kubeletDir`	Ermöglicht das Überschreiben des Host-Speicherorts des internen Zustands von kubelet.	`"/var/lib/kubelet"`
`operatorLogLevel`	Ermöglicht das Protokollierungslevel des Trident-Operators auf: `trace`, `debug`, `info`, `warn`, `error` oder `fatal` festzulegen.	`"info"`
`operatorDebug`	Ermöglicht das Festlegen des Protokollierungsgrads des Trident-Operators auf Debug.	`true`
`operatorImage`	Ermöglicht das vollständige Überschreiben des Bildes für `trident-operator`.	`""`
`operatorImageTag`	Ermöglicht das Überschreiben des Tags des `trident-operator` -Images.	`""`
`tridentIPv6`	Ermöglicht die Aktivierung von Trident für den Einsatz in IPv6-Clustern.	`false`
`tridentK8sTimeout`	Überschreibt die standardmäßige 30-Sekunden-Zeitüberschreitung für die meisten Kubernetes-API-Operationen (falls ungleich Null, in Sekunden).	`0`

Option	Beschreibung	Standard
`tridentHttpRequestTimeout`	Überschreibt die standardmäßige 90-Sekunden-Zeitüberschreitung für HTTP-Anfragen, wobei `0s` eine unbegrenzte Dauer für die Zeitüberschreitung bedeutet. Negative Werte sind nicht zulässig.	`"90s"`
`tridentSilenceAutosupport`	Ermöglicht das Deaktivieren der periodischen AutoSupport-Berichterstattung von Trident.	`false`
`tridentAutosupportImageTag`	Ermöglicht das Überschreiben des Tags des Images für den Trident AutoSupport-Container.	`<version>`
`tridentAutosupportProxy`	Ermöglicht dem Trident AutoSupport Container, über einen HTTP-Proxy nach Hause zu telefonieren.	`""`
`tridentLogFormat`	Legt das Trident Protokollierungsformat (`text` oder `json` fest.	`"text"`
`tridentDisableAuditLog`	Deaktiviert den Trident Audit-Logger.	`true`
`tridentLogLevel`	Ermöglicht das Protokollierungslevel von Trident auf `trace`, `debug`, `info`, `warn`, `error` oder `fatal` festzulegen.	`"info"`
`tridentDebug`	Ermöglicht das Festlegen des Protokollierungslevels von Trident auf `debug`. Sie können den erzwungenen Trennvorgang durch Integration mit dem node health check (NHC) Operator automatisieren. Weitere Informationen finden Sie unter "Automatisierung des Failovers von zustandsbehafteten Anwendungen mit Trident".	`false`
`tridentLogWorkflows`	Ermöglicht die Aktivierung bestimmter Trident-Workflows für die Protokollierung von Ablaufverfolgungen oder die Unterdrückung von Protokollen.	`""`
`tridentLogLayers`	Ermöglicht die Aktivierung bestimmter Trident-Ebenen für die Protokollierung von Ablaufverfolgungen oder die Unterdrückung von Protokollen.	`""`

Option	Beschreibung	Standard
`tridentImage`	Ermöglicht das vollständige Überschreiben des Images für Trident.	`""`
`tridentImageTag`	Ermöglicht das Überschreiben des Tags des Images für Trident.	`""`
`tridentProbePort`	Ermöglicht das Überschreiben des Standardports, der für Kubernetes Liveness-/Readiness-Probes verwendet wird.	`""`
`windows`	Ermöglicht die Installation von Trident auf einem Windows-Worker-Knoten.	`false`
`enableForceDetach`	Ermöglicht das Aktivieren der Funktion „Force Detach“.	`false`
`excludePodSecurityPolicy`	Schließt die Sicherheitsrichtlinie für den Operator-Pod von der Erstellung aus.	`false`
`cloudProvider`	Auf `"Azure"` setzen, wenn verwaltete Identitäten oder eine Cloud-Identität auf einem AKS-Cluster verwendet werden. Auf "AWS" setzen, wenn eine Cloud-Identität auf einem EKS-Cluster verwendet wird.	`""`
`cloudIdentity`	Legen Sie auf die Workload-Identität („azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx“) fest, wenn Sie Cloud Identity auf einem AKS-Cluster verwenden. Legen Sie auf die AWS IAM-Rolle („'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'“) fest, wenn Sie Cloud Identity auf einem EKS-Cluster verwenden.	`""`
`iscsiSelfHealingInterval`	Das Intervall, in dem die iSCSI Selbstreparatur aufgerufen wird.	`5m0s`
`iscsiSelfHealingWaitTime`	Die Zeitspanne, nach der die iSCSI Selbstreparatur einen Versuch unternimmt, eine veraltete Sitzung durch Durchführung einer Abmeldung und anschließenden Anmeldung zu beheben.	`7m0s`

Option	Beschreibung	Standard
`nodePrep`	Ermöglicht Trident, die Knoten des Kubernetes-Clusters für die Verwaltung von Volumes mithilfe des angegebenen Datenspeicherprotokolls vorzubereiten. **Derzeit ist `iscsi` der einzige unterstützte Wert.** HINWEIS: Ab OpenShift 4.19 ist die Mindestversion von Trident für diese Funktion 25.06.1.	
`enableConcurrency`	Ermöglicht gleichzeitige Trident-Controller-Operationen für einen verbesserten Durchsatz. **Tech Preview**: Diese Funktion ist experimentell und unterstützt derzeit eingeschränkte parallele Arbeitsabläufe mit den ONTAP-NAS (nur NFS) und ONTAP-SAN (NVMe für unified ONTAP 9) Treibern, zusätzlich zur bestehenden Tech Preview für den ONTAP-SAN Treiber (iSCSI- und FCP-Protokolle in unified ONTAP 9).	false
`k8sAPIQPS`	Die vom Controller bei der Kommunikation mit dem Kubernetes-API-Server verwendete Abfrageanzahl pro Sekunde (QPS). Der Burst-Wert wird automatisch anhand des QPS-Werts festgelegt.	`100`; optional

Option	Beschreibung	Standard

`resources`	Legt Kubernetes-Ressourcenlimits und -anforderungen für die Trident-Controller-, Node- und Operator-Pods fest. Sie können CPU und Arbeitsspeicher für jeden Container und Sidecar konfigurieren, um die Ressourcenzuweisung in Kubernetes zu verwalten.
	
	Weitere Informationen zur Konfiguration von Ressourcenanforderungen und -limits finden Sie unter "Ressourcenverwaltung für Pods und Container".
	
	• Ändern Sie NICHT die Namen von Containern oder Feldern.
	• Ändern Sie die Einrückung NICHT – die YAML-Einrückung ist für das korrekte Parsen entscheidend.
		<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>

Option	Beschreibung	Standard
`httpsMetrics`	Aktivieren Sie HTTPS für den Prometheus metrics endpoint.	false
`hostNetwork`	Aktiviert die Host-Netzwerkverbindung für den Trident Controller. Dies ist nützlich, wenn Sie den Frontend- und Backend-Datenverkehr in einem Multi-Home-Netzwerk trennen möchten.	false

wenn sie nicht angegeben sind.
- Containernamen werden so aufgelistet, wie sie in den Pod-Spezifikationen erscheinen.
- Sidecars sind unter jedem Hauptcontainer aufgeführt.
- Überprüfen Sie das Feld „TORC" `status.CurrentTridentConfig`, um die aktuell angewendeten Werte anzuzeigen.

```
            memory: 10Mi
          limits:
            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:
              memory:
      operator:
        requests:
          cpu: 10m
          memory: 40Mi
        limits:
          cpu:
          memory:
```

Controller-Pods und Node-Pods verstehen

Trident läuft als einzelner Controller-Pod sowie als Node-Pod auf jedem Worker-Knoten im Cluster. Der Node-Pod muss auf jedem Host ausgeführt werden, auf dem Sie möglicherweise ein Trident Volume einbinden möchten.

Kubernetes "Knotenselektoren" und "Toleranzen und Taints" werden verwendet, um einen Pod auf einen bestimmten oder bevorzugten Knoten zu beschränken. Mit dem `ControllerPlugin` und `NodePlugin` können Sie Einschränkungen und Überschreibungen festlegen.

- Das Controller-Plugin übernimmt die Bereitstellung und Verwaltung von Volumes, wie Snapshots und Größenänderungen.
- Das Node-Plugin übernimmt das Anbinden des Speichers an den Knoten.

Trident operator mit Helm bereitstellen (Offline mode)

Sie können den Trident-Operator bereitstellen und Trident mithilfe von Helm installieren. Dieser Vorgang gilt für Installationen, bei denen die von Trident benötigten Container-Images in einer privaten Registry gespeichert sind. Wenn Sie keine private Image-Registry besitzen, verwenden Sie die "Prozess für die Standardbereitstellung".

Wichtige Informationen zu Trident 26.02

Sie müssen die folgenden wichtigen Informationen über Trident lesen.

Wichtige Informationen zu Trident

- Kubernetes 1.35 wird nun in Trident unterstützt. Trident sollte vor Kubernetes aktualisiert werden.
- Trident setzt die Verwendung der Multipathing-Konfiguration in SAN-Umgebungen strikt durch, mit einem empfohlenen Wert von `find_multipaths: no` in der multipath.conf-Datei.

  Die Verwendung einer Konfiguration ohne Multipathing oder die Verwendung von `find_multipaths: yes` oder `find_multipaths: smart` Wert in der Datei multipath.conf führt zu Mount-Fehlern. Trident hat die Verwendung von `find_multipaths: no` seit der Version 21.07 empfohlen.

Den Trident-Operator bereitstellen und Trident mit Helm installieren

Mit dem Trident "Helm Chart" können Sie den Trident Operator bereitstellen und Trident in einem Schritt installieren.

Überprüfen Sie "die Installationsübersicht", um sicherzustellen, dass Sie die Installationsvoraussetzungen erfüllt und die richtige Installationsoption für Ihre Umgebung ausgewählt haben.

Bevor Sie beginnen

Zusätzlich zu dem "Bereitstellungsvoraussetzungen" benötigen Sie "Helm Version 3".

> Wenn Sie Trident in einem privaten Repository installieren und den `imageRegistry` Schalter verwenden, um den Speicherort des Repositorys anzugeben, verwenden Sie `/netapp/` nicht im Repository-Pfad.

Schritte

1. Fügen Sie das Trident Helm repository hinzu:

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

2. Verwenden Sie `helm install` und geben Sie einen Namen für Ihre Bereitstellung sowie den Speicherort der Image-Registry an. Ihre "Trident und CSI images" können sich in einer Registry oder in verschiedenen Registrys befinden, aber alle CSI-Images müssen sich in derselben Registry befinden. In den Beispielen ist `100.2602.0` die Version von Trident, die Sie installieren.

Images in einem Registry

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

Images in verschiedenen Registern

```
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}
```

Wenn Sie bereits einen Namespace für Trident erstellt haben, wird durch den `--create -namespace` Parameter kein zusätzlicher Namespace erstellt.

Sie können `helm list` verwenden, um Installationsdetails wie Name, Namespace, Chart, Status, App-Version und Revisionsnummer zu überprüfen.

Konfigurationsdaten während der Installation übergeben

Es gibt zwei Möglichkeiten, Konfigurationsdaten während der Installation zu übergeben:

Option	Beschreibung
`--values` (oder `-f`)	Geben Sie eine YAML-Datei mit Überschreibungen an. Dies kann mehrfach angegeben werden, und die ganz rechts stehende Datei hat Vorrang.
`--set`	Geben Sie Überschreibungen in der Befehlszeile an.

Um beispielsweise den Standardwert von `debug` zu ändern, führen Sie den folgenden Befehl aus, wobei `100.2602.0` die Version von Trident ist, die Sie installieren:

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

Um den nodePrep Wert hinzuzufügen, führen Sie den folgenden Befehl aus:

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

Konfigurationsoptionen

Diese Tabelle und die `values.yaml` Datei, die Teil des Helm-Charts ist, enthalten die Liste der Schlüssel und ihrer Standardwerte.

> Entfernen Sie die Standardaffinität nicht aus der Datei values.yaml. Wenn Sie eine benutzerdefinierte Affinität angeben möchten, erweitern Sie die Standardaffinität.

Option	Beschreibung	Standard
`nodeSelector`	Knotenbezeichnungen für die Pod-Zuweisung	
`podAnnotations`	Pod-Anmerkungen	
`deploymentAnnotations`	Bereitstellungsanmerkungen	
`tolerations`	Toleranzen für die Pod-Zuordnung	

Option	Beschreibung	Standard

`affinity`	Affinität für die Pod-Zuweisung	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedul`
		`ingIgnoredDuringExecu`
		`tion:`
		
		`nodeSelectorTerms:`
		`  -`
		`matchExpressions:`
		`  - key:`
		`kubernetes.io/arch`
		
		`operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key:`
		`kubernetes.io/os`
		
		`operator: In`
		`  values:`
		`  - linux`
		
		Entfernen Sie die Standardaffinität nicht aus der Datei values.yaml. Wenn Sie eine benutzerdefinierte Affinität angeben möchten, erweitern Sie die Standardaffinität.

`tridentControllerPluginNod` `eSelector`	Zusätzliche Knotenselektoren für Pods. Siehe "Controller-Pods und Node-Pods verstehen" für Details.	
`tridentControllerPluginTol` `erations`	Überschreibt die Kubernetes-Toleranzen für Pods. Siehe "Controller-Pods und Node-Pods verstehen" für Details.	

Option	Beschreibung	Standard
`tridentNodePluginNodeSelector`	Zusätzliche Knotenselektoren für Pods. Siehe "Controller-Pods und Node-Pods verstehen" für Details.	
`tridentNodePluginTolerations`	Überschreibt die Kubernetes-Toleranzen für Pods. Siehe "Controller-Pods und Node-Pods verstehen" für Details.	
`imageRegistry`	Identifiziert die Registry für die `trident-operator`, `trident` und andere Images. Leer lassen, um den Standardwert zu akzeptieren. WICHTIG: Wenn Sie Trident in einem privaten Repository installieren und den `imageRegistry`-Schalter zur Angabe des Repository-Speicherorts verwenden, verwenden Sie `/netapp/` nicht im Repository-Pfad.	""
`imagePullPolicy`	Legt die Image-Pull-Richtlinie für das `trident-operator` fest.	`IfNotPresent`
`imagePullSecrets`	Legt die Image-Pull-Secrets für die `trident-operator`, `trident` und andere Images fest.	
`kubeletDir`	Ermöglicht das Überschreiben des Host-Speicherorts des internen Zustands von kubelet.	`"/var/lib/kubelet"`
`operatorLogLevel`	Ermöglicht das Protokollierungslevel des Trident-Operators auf: `trace`, `debug`, `info`, `warn`, `error` oder `fatal` festzulegen.	`"info"`
`operatorDebug`	Ermöglicht das Festlegen des Protokollierungsgrads des Trident-Operators auf Debug.	`true`
`operatorImage`	Ermöglicht das vollständige Überschreiben des Bildes für `trident-operator`.	""
`operatorImageTag`	Ermöglicht das Überschreiben des Tags des `trident-operator` -Images.	""
`tridentIPv6`	Ermöglicht die Aktivierung von Trident für den Einsatz in IPv6-Clustern.	`false`

Option	Beschreibung	Standard
`tridentK8sTimeout`	Überschreibt die standardmäßige 180-Sekunden-Zeitüberschreitung für die meisten Kubernetes-API-Operationen (falls ungleich Null, in Sekunden). Der `tridentK8sTimeout` Parameter ist nur für Trident-Installationen anwendbar.	`180`
`tridentHttpRequestTimeout`	Überschreibt die standardmäßige 90-Sekunden-Zeitüberschreitung für HTTP-Anfragen, wobei `0s` eine unbegrenzte Dauer für die Zeitüberschreitung bedeutet. Negative Werte sind nicht zulässig.	`"90s"`
`tridentSilenceAutosupport`	Ermöglicht das Deaktivieren der periodischen AutoSupport-Berichterstattung von Trident.	`false`
`tridentAutosupportImageTag`	Ermöglicht das Überschreiben des Tags des Images für den Trident AutoSupport-Container.	`<version>`
`tridentAutosupportProxy`	Ermöglicht dem Trident AutoSupport Container, über einen HTTP-Proxy nach Hause zu telefonieren.	`""`
`tridentLogFormat`	Legt das Trident Protokollierungsformat (`text` oder `json` fest.	`"text"`
`tridentDisableAuditLog`	Deaktiviert den Trident Audit-Logger.	`true`
`tridentLogLevel`	Ermöglicht das Protokollierungslevel von Trident auf `trace`, `debug`, `info`, `warn`, `error` oder `fatal` festzulegen.	`"info"`
`tridentDebug`	Ermöglicht das Festlegen des Protokollierungslevels von Trident auf `debug`.	`false`
`tridentLogWorkflows`	Ermöglicht die Aktivierung bestimmter Trident-Workflows für die Protokollierung von Ablaufverfolgungen oder die Unterdrückung von Protokollen.	`""`

Option	Beschreibung	Standard
`tridentLogLayers`	Ermöglicht die Aktivierung bestimmter Trident-Ebenen für die Protokollierung von Ablaufverfolgungen oder die Unterdrückung von Protokollen.	""
`tridentImage`	Ermöglicht das vollständige Überschreiben des Images für Trident.	""
`tridentImageTag`	Ermöglicht das Überschreiben des Tags des Images für Trident.	""
`tridentProbePort`	Ermöglicht das Überschreiben des Standardports, der für Kubernetes Liveness-/Readiness-Probes verwendet wird.	""
`windows`	Ermöglicht die Installation von Trident auf einem Windows-Worker-Knoten.	`false`
`enableForceDetach`	Ermöglicht das Aktivieren der Funktion „Force Detach“. Sie können den erzwungenen Trennvorgang durch Integration mit dem node health check (NHC) Operator automatisieren. Weitere Informationen finden Sie unter "Automatisierung des Failovers von zustandsbehafteten Anwendungen mit Trident".	`false`
`excludePodSecurityPolicy`	Schließt die Sicherheitsrichtlinie für den Operator-Pod von der Erstellung aus.	`false`
`nodePrep`	Ermöglicht Trident, die Knoten des Kubernetes-Clusters für die Verwaltung von Volumes mithilfe des angegebenen Datenspeicherprotokolls vorzubereiten. **Derzeit ist `iscsi` der einzige unterstützte Wert.** Ab OpenShift 4.19 ist die minimale unterstützte Trident-Version für diese Funktion 25.06.1.	

Option	Beschreibung	Standard

`resources`	Legt Kubernetes-Ressourcenlimits und -anforderungen für die Trident-Controller-, Node- und Operator-Pods fest. Sie können CPU und Arbeitsspeicher für jeden Container und Sidecar konfigurieren, um die Ressourcenzuweisung in Kubernetes zu verwalten.
	
	Weitere Informationen zur Konfiguration von Ressourcenanforderungen und -limits finden Sie unter "Ressourcenverwaltung für Pods und Container".
	
	• Ändern Sie NICHT die Namen von Containern oder Feldern.
	• Ändern Sie die Einrückung NICHT – die YAML-Einrückung ist für das korrekte Parsen entscheidend.
		<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>

Trident-Operator-Installation anpassen

Der Trident-Operator ermöglicht es Ihnen, die Trident-Installation mithilfe der Attribute in der `TridentOrchestrator` spec anzupassen. Wenn Sie die Installation über das hinaus anpassen möchten, was `TridentOrchestrator` Argumente erlauben, sollten Sie `tridentctl` verwenden, um benutzerdefinierte YAML-Manifeste zu generieren und diese nach Bedarf zu modifizieren.

Controller-Pods und Node-Pods verstehen

Trident läuft als einzelner Controller-Pod und als Node-Pod auf jedem Worker-Knoten im Cluster. Der Node-Pod muss auf jedem Host ausgeführt werden, auf dem Sie möglicherweise ein Trident Volume einbinden möchten.

Kubernetes "Knotenselektoren" und "Tolerations und Taints" werden verwendet, um einen Pod auf einen bestimmten oder bevorzugten Knoten zu beschränken. Mit dem `ControllerPlugin` und `NodePlugin` können Sie Einschränkungen und Überschreibungen festlegen.

- Das Controller-Plugin übernimmt die Bereitstellung und Verwaltung von Volumes, wie Snapshots und Größenänderungen.
- Das Node-Plugin übernimmt das Anbinden des Speichers an den Knoten.

Konfigurationsoptionen

`spec.namespace` wird in `TridentOrchestrator` angegeben, um den Namespace zu kennzeichnen, in dem Trident installiert ist. Dieser Parameter **kann nach der Installation von Trident nicht mehr aktualisiert werden**. Ein entsprechender Versuch führt dazu, dass sich der `TridentOrchestrator` Status in `Failed` ändert. Trident ist nicht für die Migration zwischen Namespaces vorgesehen.

Diese Tabelle enthält `TridentOrchestrator`Attribute im Detail.

Parameter	Beschreibung	Standard
`namespace`	Namespace, in dem Trident installiert werden soll	`"default"`
`debug`	Debugging für Trident aktivieren	`false`
`enableForceDetach`	`ontap-san`, `ontap-san-economy`, `ontap-nas`, und `ontap-nas-economy` nur. Funktioniert mit Kubernetes Non-Graceful Node Shutdown (NGNS), um Clusteradministratoren die Möglichkeit zu geben, Workloads mit eingebundenen Volumes sicher auf neue Knoten zu migrieren, falls ein Knoten fehlerhaft wird. Weitere Informationen finden Sie unter "Automatisierung des Failovers von zustandsbehafteten Anwendungen mit Trident".	`false`
`windows`	Das Setzen auf `true` ermöglicht die Installation auf Windows-Worker-Knoten.	`false`

- Es gelten standardmäßig keine Beschränkungen für Requests haben Standardwerte.
- Containernamen werden so aufgelistet, wie sie in den Pod-Spezifikationen erscheinen.
- Sidecars sind unter jedem Hauptcontainer aufgeführt.
- Überprüfen Sie das Feld „TORC“ `status.CurrentInstallationParams`, um die aktuell angewendeten Werte anzuzeigen.

```
    trident-main:
      requests:
        cpu: 10m
        memory: 40Mi
      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:
  operator:
    requests:
      cpu: 10m
      memory: 40Mi
    limits:
```

Parameter	Beschreibung	Standard
`cloudProvider`	Auf `"Azure"` setzen, wenn verwaltete Identitäten oder eine Cloud-Identität auf einem AKS-Cluster verwendet werden. Setzen Sie auf `"AWS"`, wenn eine Cloud-Identität auf einem EKS-Cluster verwendet wird. Setzen Sie auf `"GCP"`, wenn eine Cloud-Identität auf einem GKE-Cluster verwendet wird.	`""`
`cloudIdentity`	Legen Sie auf die Workload-Identität („azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx“) fest, wenn Sie Cloud Identity auf einem AKS-Cluster verwenden. Legen Sie auf die AWS IAM-Rolle („'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'“) fest, wenn Sie Cloud Identity auf einem EKS-Cluster verwenden. Auf Cloud-Identität („'iam.gke.io/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com'“) setzen, wenn Cloud Identity auf einem GKE-Cluster verwendet wird.	`""`
`IPv6`	Trident über IPv6 installieren	false
`k8sTimeout`	Zeitüberschreitung für Kubernetes-Operationen. Der `k8sTimeout` Parameter ist nur für Trident-Installationen anwendbar.	`180sec`
`silenceAutosupport`	Senden Sie Autosupport-Pakete nicht automatisch an NetApp	`false`
`autosupportImage`	Das Container-Image für Autosupport Telemetry	`"netapp/trident-autosupport10"`
`autosupportProxy`	Die Adresse/der Port eines Proxys zum Senden von Autosupport-Telemetrie	`"http://proxy.example.com:8888"`
`uninstall`	Eine Kennzeichnung, die zur Deinstallation von Trident verwendet wird	`false`
`logFormat`	Trident Protokollierungsformat, das verwendet werden soll [text,json]	`"text"`
`tridentImage`	Trident-Image zur Installation	`"netapp/trident:26.02"`
`imageRegistry`	Pfad zur internen Registry im Format `<registry FQDN>[:port][/subpath]`	`"registry.k8s.io"`
`kubeletDir`	Pfad zum kubelet-Verzeichnis auf dem Host	`"/var/lib/kubelet"`
`wipeout`	Eine Liste der Ressourcen, die gelöscht werden müssen, um eine vollständige Entfernung von Trident durchzuführen	

Parameter	Beschreibung	Standard
`imagePullSecrets`	Geheimnisse zum Abrufen von Images aus einer internen Registry	
`imagePullPolicy`	Legt die Richtlinie zum Abrufen von Images für den Trident Operator fest. Gültige Werte sind: `Always` um das Image immer abzurufen. `IfNotPresent` um das Image nur abzurufen, wenn es noch nicht auf dem Knoten vorhanden ist. `Never` um das Image niemals abzurufen.	`IfNotPresent`
`controllerPluginNodeSelector`	Zusätzliche Knotenselektoren für Pods. Folgt dem gleichen Format wie `pod.spec.nodeSelector`.	Kein Standardwert; optional
`controllerPluginTolerations`	Überschreibt die Kubernetes-Toleranzen für Pods. Folgt dem gleichen Format wie `pod.spec.Tolerations`.	Kein Standardwert; optional
`nodePluginNodeSelector`	Zusätzliche Knotenselektoren für Pods. Folgt dem gleichen Format wie `pod.spec.nodeSelector`.	Kein Standardwert; optional
`nodePluginTolerations`	Überschreibt die Kubernetes-Toleranzen für Pods. Folgt dem gleichen Format wie `pod.spec.Tolerations`.	Kein Standardwert; optional
`nodePrep`	Ermöglicht Trident, die Knoten des Kubernetes-Clusters für die Verwaltung von Volumes mithilfe des angegebenen Datenspeicherprotokolls vorzubereiten. **Derzeit ist `iscsi` der einzige unterstützte Wert.** Ab OpenShift 4.19 ist die minimale unterstützte Trident-Version für diese Funktion 25.06.1.	
`k8sAPIQPS`	Die vom Controller bei der Kommunikation mit dem Kubernetes-API-Server verwendete Abfrageanzahl pro Sekunde (QPS). Der Burst-Wert wird automatisch anhand des QPS-Werts festgelegt.	`100`; optional
`enableConcurrency`	Ermöglicht gleichzeitige Trident-Controller-Operationen für einen verbesserten Durchsatz. **Tech Preview**: Diese Funktion ist experimentell und unterstützt derzeit eingeschränkte parallele Arbeitsabläufe mit den ONTAP-NAS (nur NFS) und ONTAP-SAN (NVMe für unified ONTAP 9) Treibern, zusätzlich zur bestehenden Tech Preview für den ONTAP-SAN Treiber (iSCSI- und FCP-Protokolle in unified ONTAP 9).	false

Parameter	Beschreibung	Standard

`resources`	Legt Ressourcenlimits und -anforderungen für den Trident-Controller und die Node-Pods in Kubernetes fest. Sie können CPU und Arbeitsspeicher für jeden Container und Sidecar konfigurieren, um die Ressourcenzuweisung in Kubernetes zu verwalten.
	
	Weitere Informationen zur Konfiguration von Ressourcenanforderungen und -limits finden Sie unter "Ressourcenverwaltung für Pods und Container".
	
	• Ändern Sie NICHT die Namen von Containern oder Feldern.
	• Ändern Sie die Einrückung NICHT – die YAML-Einrückung ist für das korrekte Parsen entscheidend.
	
	• Standardmäßig werden keine Limits angewendet – nur Anfragen haben Standardwerte und werden automatisch angewendet, wenn sie nicht angegeben sind.
	• Containernamen werden so aufgelistet, wie sie in den Pod-Spezifikationen erscheinen.
	• Sidecars sind unter jedem Hauptcontainer aufgeführt.
	• Überprüfen Sie das Feld „TORC“ `status.CurrentInstallationParams`, um die aktuell angewendeten Werte anzuzeigen.
		`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:`

Parameter	Beschreibung	Standard
`httpsMetrics`	Aktivieren Sie HTTPS für den Prometheus metrics endpoint.	false
`hostNetwork`	Aktiviert die Host-Netzwerkverbindung für den Trident Controller. Dies ist nützlich, wenn Sie den Frontend- und Backend-Datenverkehr in einem Multi-Home-Netzwerk trennen möchten.	false

Weitere Informationen zur Formatierung von Pod-Parametern finden Sie unter "Pods.Knoten zuweisen".

Beispielkonfigurationen

Sie können die Attribute in Konfigurationsoptionen verwenden, wenn Sie `TridentOrchestrator` definieren, um Ihre Installation anzupassen.

Grundlegende benutzerdefinierte Konfiguration

Dieses Beispiel, das nach Ausführung des `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml` Befehls erstellt wurde, stellt eine einfache benutzerdefinierte Installation dar:

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

Knotenselektoren

Dieses Beispiel installiert Trident mit Knotenselektoren.

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

Windows-Worker-Knoten

Dieses Beispiel, das nach Ausführung des `cat deploy/crds/tridentorchestrator_cr.yaml` Befehls erstellt wurde, installiert Trident auf einem Windows-Worker-Knoten.

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

Verwaltete Identitäten auf einem AKS-Cluster

Dieses Beispiel installiert Trident, um verwaltete Identitäten auf einem AKS-Cluster zu aktivieren.

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

Cloud-Identität auf einem AKS-Cluster

Dieses Beispiel installiert Trident zur Verwendung mit einer Cloud-Identität auf einem AKS-Cluster.

```
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-Identität auf einem EKS-Cluster

Dieses Beispiel installiert Trident zur Verwendung mit einer Cloud-Identität auf einem AKS-Cluster.

```
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'"
```

Cloud-Identität für GKE

Dieses Beispiel installiert Trident zur Verwendung mit einer Cloud-Identität auf einem GKE-Cluster.

```
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
```

Konfiguration der Kubernetes-Ressourcenanforderungen und -limits für Trident controller und Trident Linux node pods

Dieses Beispiel konfiguriert Kubernetes-Ressourcenanforderungen und -limits für Trident controller und Trident Linux node pods.

Hinweis: Die in diesem Beispiel angegebenen Anforderungs- und Grenzwerte dienen lediglich Demonstrationszwecken. Passen Sie diese Werte an Ihre Umgebung und Arbeitslastanforderungen an.

```
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
```

Konfiguration von Kubernetes-Ressourcenanforderungen und -limits für Trident Controller sowie für Trident Windows- und Linux-Node-Pods

Dieses Beispiel konfiguriert Kubernetes-Ressourcenanforderungen und -Limits für Trident Controller sowie Trident Windows- und Linux Node Pods.

Hinweis: Die in diesem Beispiel angegebenen Anforderungs- und Grenzwerte dienen lediglich Demonstrationszwecken. Passen Sie diese Werte an Ihre Umgebung und Arbeitslastanforderungen an.

```
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
```