Trident オペレータを使用したインストール

Trident

NetApp
July 01, 2026

目次

Trident オペレータを使用したインストール

Tridentオペレータを手動で導入する（標準モード）

Tridentオペレーターを手動で導入してTridentをインストールできます。このプロセスは、Tridentで必要なコンテナイメージがプライベートレジストリに保存されていないインストールに適用されます。プライベートイメージレジストリをお持ちの場合は、"オフライン展開のプロセス"を使用してください。

Trident 26.02に関する重要な情報

Trident に関する以下の重要な情報を必ずお読みください。

Tridentに関する重要な情報

- Kubernetes 1.35 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制し、multipath.conf ファイル内の推奨値は `find_multipaths: no`です。

  非マルチパス構成の使用、または multipath.conf ファイルでの `find_multipaths: yes`または `find_multipaths: smart`値の使用は、マウントの失敗を引き起こします。Trident は、 21.07 リリース以降 `find_multipaths: no`の使用を推奨しています。

Tridentオペレーターを手動で展開してTridentをインストールする

"インストールの概要"を確認して、インストールの前提条件を満たしていること、および環境に適したインストールオプションが選択されていることを確認します。

開始する前に

インストールを始める前に、Linuxホストにログインし、正常に動作している"サポートされている Kubernetes クラスター"を管理していること、および必要な権限を持っていることを確認します。

> OpenShift では、以下のすべての例で `oc` を `kubectl` の代わりに使用し、最初に `oc login -u system:admin` または `oc login -u kube-admin` を実行して **system:admin** としてログインしてください。

1. Kubernetes のバージョンを確認します：

```
kubectl version
```

2. クラスタ管理者権限を確認します：

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

3. Docker Hub からイメージを使用するポッドを起動し、ポッド ネットワーク経由でストレージ システムにアクセスできることを確認します：

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

ステップ**1**：**Trident**インストーラーパッケージをダウンロードする

Trident インストーラー パッケージには、Trident オペレーターを導入して Trident をインストールするために必要なものがすべて含まれています。最新バージョンのTridentインストーラーを"GitHub の *Assets* セクション"からダウンロードして解凍します。

```
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
```

ステップ**2**： `TridentOrchestrator` **CRD**を作成する

`TridentOrchestrator`カスタムリソース定義（CRD）を作成します。後で`TridentOrchestrator`カスタムリソースを作成します。 `deploy/crds`で適切なCRD YAMLバージョンを使用して `TridentOrchestrator`CRDを作成します。

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

ステップ**3**：**Trident**オペレーターを導入する

Tridentインストーラーは、オペレーターをインストールし、関連オブジェクトを作成するために使用できるバンドル ファイルを提供します。バンドル ファイルは、オペレーターを導入し、デフォルト設定を使用してTridentをインストールする簡単な方法です。

- Kubernetes 1.24を実行しているクラスターの場合は、 `bundle_pre_1_25.yaml`を使用します。
- Kubernetes 1.25以降を実行しているクラスターの場合は、 `bundle_post_1_25.yaml`を使用します。

開始する前に

- デフォルトでは、Tridentインストーラは `trident`ネームスペースにオペレータを導入します。 `trident`ネームスペースが存在しない場合は、次のコマンドを使用して作成します：

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

- `trident`名前空間以外の名前空間にオペレーターを導入するには、 `serviceaccount.yaml`、 `clusterrolebinding.yaml`、 `operator.yaml`を更新し、 `kustomization.yaml`を使用してバンドルファイルを生成します。

  a. `kustomization.yaml`を次のコマンドで作成します。ここで、_<bundle.yaml>_は、Kubernetes のバージョンに応じて `bundle_pre_1_25.yaml`または `bundle_post_1_25.yaml`になります。

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

  b. 次のコマンドを使用してバンドルをコンパイルします。_<bundle.yaml>_は `bundle_pre_1_25.yaml`または `bundle_post_1_25.yaml`Kubernetesのバージョンに基づきます。

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

手順

1. リソースを作成し、オペレーターをデプロイします：

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

2. オペレーター、デプロイメント、レプリカセットが作成されたことを確認します。

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

Kubernetes クラスターにはオペレーターのインスタンスが **1** つ のみ存在する必要があります。Trident オペレーターの複数のデプロイメントを作成しないでください。

ステップ**4**： `**TridentOrchestrator**`を作成して**Trident**をインストールする

これで `TridentOrchestrator`を作成し、Tridentをインストールできます。オプションで、"Tridentインストールをカスタマイズ"を `TridentOrchestrator`仕様の属性を使って行うこともできます。

```
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
```

インストールの確認

インストールを確認する方法はいくつかあります。

`TridentOrchestrator`ステータスの使用

`TridentOrchestrator`のステータスは、インストールが成功したかどうかを示し、インストールされた `Trident` のバージョンを表示します。インストール中、`TridentOrchestrator`のステータスは `Installing`から `Installed`に変わります。`Failed`ステータスが表示され、オペレータが自己修復できない場合は、`link:../troubleshooting.html["ログを確認する"]`。

ステータス	概要
インストールしています	オペレーターは、この `TridentOrchestrator` CRを使用してTridentをインストールしています。
インストール済み	Tridentが正常にインストールされました。
アンインストール	オペレーターはTridentをアンインストール中です `spec.uninstall=true`。
アンインストール済み	Tridentがアンインストールされました。
失敗	オペレーターはTridentのインストール、パッチ適用、更新、またはアンインストールができませんでした。オペレーターは自動的にこの状態からの回復を試みます。この状態が続く場合は、トラブルシューティングが必要になります。
更新中	オペレーターは既存のインストールを更新しています。
エラー	その `TridentOrchestrator`は使用されません。別のものがすでに存在します。

ポッド作成ステータスの使用

作成されたポッドのステータスを確認することで、Tridentのインストールが完了したかどうかを確認できます：

```
kubectl get pods -n trident

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

使用 `tridentctl`

`tridentctl`を使用して、インストールされているTridentのバージョンを確認できます。

```
./tridentctl -n trident version

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

Tridentオペレーターを手動で導入する（オフラインモード）

Tridentオペレーターを手動で導入してTridentをインストールできます。このプロセスは、 Trident に必要なコンテナイメージがプライベートレジストリに保存されているインストールに適用されます。プライベートイメージレジストリがない場合は、"標準導入のプロセス"を使用します。

Tridentに関する重要な情報

Trident に関する以下の重要な情報を必ずお読みください。

Tridentに関する重要な情報

- Kubernetes 1.35 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制し、multipath.conf ファイル内の推奨値は `find_multipaths: no`です。

  非マルチパス構成の使用、または multipath.conf ファイルでの `find_multipaths: yes`または `find_multipaths: smart`値の使用は、マウントの失敗を引き起こします。Trident は、 21.07 リリース以降 `find_multipaths: no`の使用を推奨しています。

Tridentオペレーターを手動で展開してTridentをインストールする

"インストールの概要"を確認して、インストールの前提条件を満たしていること、および環境に適したインストールオプションが選択されていることを確認します。

開始する前に

Linux ホストにログインし、正常に動作している"サポートされている Kubernetes クラスター"を管理していること、および必要な権限があることを確認します。

> OpenShift では、以下のすべての例で `oc` を `kubectl` の代わりに使用し、最初に `oc login -u system:admin` または `oc login -u kube-admin` を実行して **system:admin** としてログインしてください。

1. Kubernetes のバージョンを確認します：

```
kubectl version
```

2. クラスタ管理者権限を確認します：

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

3. Docker Hub からイメージを使用するポッドを起動し、ポッド ネットワーク経由でストレージ システムにアクセスできることを確認します：

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

ステップ**1**：**Trident**インストーラーパッケージをダウンロードする

Trident インストーラー パッケージには、Trident オペレーターを導入して Trident をインストールするために必要なものがすべて含まれています。最新バージョンのTridentインストーラーを"GitHub の *Assets* セクション"からダウンロードして解凍します。

```
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
```

ステップ**2**： `TridentOrchestrator` **CRD**を作成する

`TridentOrchestrator`カスタムリソース定義（CRD）を作成します。`TridentOrchestrator`カスタム リソースは後で作成します。 `deploy/crds`で適切なCRD YAML バージョンを使用して、 `TridentOrchestrator`CRD を作成します：

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

ステップ**3**：オペレータのレジストリの場所を更新する

`/deploy/operator.yaml`で、 `image: docker.io/netapp/trident-operator:26.02.0`を更新して、イメージレジストリの場所を反映します。link:../trident-get-started/requirements.html#container-images-and-corresponding-kubernetes-versions["TridentとCSIのイメージ"]は1つのレジストリまたは異なるレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例：

- `image: <your-registry>/trident-operator:26.02.0`すべてのイメージが 1 つのレジストリにある場合。
- `image: <your-registry>/netapp/trident-operator:26.02.0` Trident イメージが CSI イメージとは異なるレジストリにある場合。

ステップ**4**：**Trident**オペレーターを導入する

Tridentインストーラーは、オペレーターをインストールし、関連オブジェクトを作成するために使用できるバンドル ファイルを提供します。バンドル ファイルは、オペレーターを導入し、デフォルト設定を使用してTridentをインストールする簡単な方法です。

- Kubernetes 1.24を実行しているクラスターの場合は、 `bundle_pre_1_25.yaml`を使用します。
- Kubernetes 1.25以降を実行しているクラスターの場合は、 `bundle_post_1_25.yaml`を使用します。

開始する前に

- デフォルトでは、Tridentインストーラは `trident`ネームスペースにオペレータを導入します。 `trident`ネ

ームスペースが存在しない場合は、次のコマンドを使用して作成します：

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

- `trident`名前空間以外の名前空間にオペレーターを導入するには、 `serviceaccount.yaml`、`clusterrolebinding.yaml`、 `operator.yaml`を更新し、 `kustomization.yaml`を使用してバンドルファイルを生成します。

  a. `kustomization.yaml`を次のコマンドで作成します。ここで、_<bundle.yaml>_は、Kubernetes のバージョンに応じて `bundle_pre_1_25.yaml`または `bundle_post_1_25.yaml`になります。

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

  b. 次のコマンドを使用してバンドルをコンパイルします。_<bundle.yaml>_は `bundle_pre_1_25.yaml`または `bundle_post_1_25.yaml`Kubernetesのバージョンに基づきます。

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

手順

1. リソースを作成し、オペレーターをデプロイします：

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

2. オペレーター、デプロイメント、レプリカセットが作成されたことを確認します。

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

Kubernetes クラスターにはオペレーターのインスタンスが **1** つ のみ存在する必要があります。Trident オペレーターの複数のデプロイメントを作成しないでください。

ステップ**5**： **`TridentOrchestrator`**でイメージレジストリの場所を更新します

"TridentとCSIのイメージ"は1つのレジストリまたは異なるレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。 `deploy/crds/tridentorchestrator_cr.yaml`を更新して、レジストリ構成に基づいて追加の場所の仕様を追加します。

1つのレジストリ内の画像

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

異なるレジストリ内のイメージ

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

ステップ**6**： `**TridentOrchestrator**`を作成して**Trident**をインストールする

これで `TridentOrchestrator`を作成し、Tridentをインストールできます。オプションで、"Tridentインストールをカスタマイズ"をさらに `TridentOrchestrator`仕様の属性を使ってカスタマイズできます。次の例は、TridentとCSIイメージが異なるレジストリに配置されているインストール方法を示しています。

```
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
```

インストールの確認

インストールを確認する方法はいくつかあります。

`TridentOrchestrator`ステータスの使用

`TridentOrchestrator`のステータスは、インストールが成功したかどうかを示し、インストールされた `Trident` のバージョンを表示します。インストール中、`TridentOrchestrator`のステータスは `Installing`から `Installed`に変わります。`Failed`ステータスが表示され、オペレータが自己修復できない場合は、`link:../troubleshooting.html["ログを確認する"]`。

ステータス	概要
インストールしています	オペレーターは、この `TridentOrchestrator` CRを使用してTridentをインストールしています。
インストール済み	Tridentが正常にインストールされました。
アンインストール	オペレーターはTridentをアンインストール中です `spec.uninstall=true`。
アンインストール済み	Tridentがアンインストールされました。
失敗	オペレーターはTridentのインストール、パッチ適用、更新、またはアンインストールができませんでした。オペレーターは自動的にこの状態からの回復を試みます。この状態が続く場合は、トラブルシューティングが必要になります。
更新中	オペレーターは既存のインストールを更新しています。
エラー	その `TridentOrchestrator`は使用されません。別のものがすでに存在します。

ポッド作成ステータスの使用

作成されたポッドのステータスを確認することで、Tridentのインストールが完了したかどうかを確認できます：

```
kubectl get pods -n trident

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

使用 `tridentctl`

`tridentctl`を使用して、インストールされているTridentのバージョンを確認できます。

```
./tridentctl -n trident version

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

Helm を使用した Trident オペレーターの導入（標準モード）

Helm を使用して Trident オペレーターを導入し、Trident をインストールできます。このプロセスは、Tridentで必要なコンテナイメージがプライベートレジストリに保存されていないインストールに適用されます。プライベートイメージレジストリをお持ちの場合は、"オフライン展開のプロセス"を使用してください。

Trident 25.10に関する重要な情報

Trident に関する以下の重要な情報を必ずお読みください。

Tridentに関する重要な情報

- Kubernetes 1.35 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制し、multipath.conf ファイル内の推奨値は `find_multipaths: no`です。

  非マルチパス構成の使用、または multipath.conf ファイルでの `find_multipaths: yes`または `find_multipaths: smart`値の使用は、マウントの失敗を引き起こします。Trident は、 21.07 リリース以降 `find_multipaths: no`の使用を推奨しています。

Helmを使用してTridentオペレーターを導入し、Tridentをインストールする

Trident "Helmチャート" を使用すると、Trident オペレーターを導入し、Trident を 1 つのステップでインストールできます。

"インストールの概要"を確認して、インストールの前提条件を満たしていること、および環境に適したインストールオプションが選択されていることを確認します。

開始する前に

"導入の前提条件"に加えて"Helm バージョン3"が必要です。

手順

1. Trident Helm リポジトリを追加します：

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

2. `helm install`を使用して、次の例のようにデプロイメントの名前を指定します。ここで、 `100..0`はインストールするTridentのバージョンです。

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

> すでにTrident用の名前空間を作成している場合は、 `--create-namespace`パラメータは追加の名前空間を作成しません。

`helm list`を使用して、名前、名前空間、チャート、ステータス、アプリのバージョン、リビジョン番号などのインストールの詳細を確認できます。

インストール中に構成データを渡す

インストール中に構成データを渡す方法は2つあります：

オプション	概要
`--values`（または `-f`）	オーバーライドを含む YAML ファイルを指定します。これは複数回指定することができ、右端のファイルが優先されます。
`--set`	コマンドラインでオーバーライドを指定します。

たとえば、 `debug`のデフォルト値を変更するには、次のコマンドを実行します。ここで、 `100.2602.0`はインストールするTridentのバージョンです：

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

構成オプション

この表と、Helm チャートの一部である `values.yaml`ファイルは、キーとそのデフォルト値のリストを提供します。

オプション	概要	デフォルト
`nodeSelector`	Pod割り当て用のノードラベル	
`podAnnotations`	Podアノテーション	
`deploymentAnnotations`	デプロイメントアノテーション	
`tolerations`	pod割り当ての許容範囲	

オプション	概要	デフォルト

`affinity`	Pod割り当ての親和性	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedulingIgnoredDuringExecution:`
		`  nodeSelectorTerms:`
		`  - matchExpressions:`
		`  - key: kubernetes.io/arch`
		`  operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key: kubernetes.io/os`
		`  operator: In`
		`  values:`
		`  - linux`
		
		values.yaml ファイルからデフォルトのアフィニティを削除しないでください。カスタムアフィニティを提供する場合は、デフォルトのアフィニティを拡張します。

`tridentControllerPluginNodeSelector`	ポッドの追加ノードセレクター。詳細については、[コントローラポッドとノードポッドについて]を参照してください。	
`tridentControllerPluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。詳細については、[コントローラポッドとノードポッドについて]を参照してください。	
`tridentNodePluginNodeSelector`	ポッドの追加ノードセレクター。詳細については、[コントローラポッドとノードポッドについて]を参照してください。	
`tridentNodePluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。詳細については、[コントローラポッドとノードポッドについて]を参照してください。	

オプション	概要	デフォルト
imageRegistry	trident-operator、trident、およびその他のイメージのレジストリを識別します。デフォルトを受け入れる場合は空のままにします。重要：プライベートリポジトリにTridentをインストールする場合、リポジトリの場所を指定するために `imageRegistry` スイッチを使用するときは、リポジトリパスに `/netapp/` を使用しないでください。	""
imagePullPolicy	`trident-operator` のイメージプルポリシーを設定します。	IfNotPresent
imagePullSecrets	trident-operator、trident、およびその他のイメージのイメージプルシークレットを設定します。	
kubeletDir	kubelet の内部状態のホストの場所を上書きできます。	"/var/lib/kubelet"
operatorLogLevel	Tridentオペレータのログレベルを trace、 debug、 info、 warn、 error、または `fatal` に設定できます。	"info"
operatorDebug	Trident オペレータのログレベルをデバッグに設定できます。	true
operatorImage	`trident-operator` の画像の完全な上書きを許可します。	""
operatorImageTag	`trident-operator` イメージのタグを上書きできます。	""
tridentIPv6	Trident が IPv6 クラスターで動作できるようにします。	false
tridentK8sTimeout	ほとんどの Kubernetes API 操作のデフォルトの 30 秒のタイムアウトをオーバーライドします（ゼロ以外の場合は秒単位）。	0
tridentHttpRequestTimeout	HTTPリクエストのデフォルトの90秒のタイムアウトを上書きします。 `0s` はタイムアウトの無限の期間です。負の値は許可されません。	"90s"
tridentSilenceAutosupport	Tridentの定期的なAutoSupportレポートを無効にすることができます。	false

オプション	概要	デフォルト
tridentAutosupportImageTag	Trident AutoSupportコンテナのイメージタグを上書きできます。	<version>
tridentAutosupportProxy	Trident AutoSupport コンテナが HTTP プロキシ経由でホームに通信できるようにします。	""
tridentLogFormat	Tridentログ形式を設定します（(text`または `json）。	"text"
tridentDisableAuditLog	Trident監査ロガーを無効にします。	true
tridentLogLevel	Tridentのログレベルを次のように設定できます： trace、 debug、 info、 warn、 error、または fatal。	"info"
tridentDebug	Tridentのログレベルを `debug`に設定できます。ノード ヘルス チェック (NHC) オペレーターとの統合により、強制デタッチ プロセスを自動化できます。詳細については、"Tridentを使用したステートフルアプリケーションのフェイルオーバーの自動化"を参照してください。	false
tridentLogWorkflows	特定のTridentワークフローでトレースログまたはログ抑制を有効にできます。	""
tridentLogLayers	特定のTridentレイヤーでトレースログまたはログ抑制を有効にできます。	""
tridentImage	Tridentの画像の完全な上書きを許可します。	""
tridentImageTag	Tridentの画像のタグを上書きできます。	""
tridentProbePort	Kubernetes の liveness/readiness プローブに使用されるデフォルトのポートをオーバーライドできます。	""
windows	Windows ワーカー ノードに Trident をインストールできるようにします。	false
enableForceDetach	強制デタッチ機能を有効にできます。	false
excludePodSecurityPolicy	オペレータ Pod セキュリティ ポリシーを作成から除外します。	false

オプション	概要	デフォルト
`cloudProvider`	AKS クラスターでマネージド ID またはクラウド ID を使用する場合は `"Azure"`に設定します。EKS クラスターでクラウド ID を使用する場合は「AWS」に設定します。	""
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（"azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（'"eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role"'）に設定します。	""
`iscsiSelfHealingInterval`	iSCSI 自己修復が呼び出される間隔。	`5m0s`
`iscsiSelfHealingWaitTime`	iSCSI自己修復がログアウトとそれに続くログインを実行することで、古いセッションを解決しようと試みるまでの期間。	`7m0s`
`nodePrep`	Tridentが指定されたデータストレージ プロトコルを使用してボリュームを管理するために、Kubernetesクラスタのノードを準備できるようにします。*現在、 `iscsi`がサポートされている唯一の値です。*注：OpenShift 4.19以降、この機能でサポートされているTridentの最小バージョンは25.06.1です。	

オプション	概要	デフォルト
`enableConcurrency`	スループットを向上させるために、Tridentコントローラーの同時操作を可能にします。 **Tech Preview**：この機能は試験的なもので、現在、ONTAP-NAS（NFSのみ）およびONTAP-SAN（統合ONTAP 9のNVMe）ドライバを使用した限定的な並列ワークフローをサポートしています。これは、既存のONTAP-SANドライバ（統合ONTAP 9のiSCSIおよびFCPストレージ プロトコル）のTech Previewに加えてのものです。	false
`k8sAPIQPS`	Kubernetes API サーバーと通信するときにコントローラーが使用する 1 秒あたりのクエリ数（QPS）の制限。バースト値は QPS 値に基づいて自動的に設定されます。	`100`；オプション

オプション	概要	デフォルト

`resources`	Trident コントローラ、ノード、およびオペレータ ポッドの Kubernetes リソース制限とリクエストを設定します。Kubernetes でのリソース割り当てを管理するために、各コンテナとサイドカーの CPU とメモリを設定できます。
	
	リソース要求と制限の設定の詳細については、"Pod と Container のリソース管理"を参照してください。
	
	• コンテナまたはフィールドの名前を変更しないでください。
	• インデントを変更しないでください。YAML インデントは適切な解析に重要です。
	
	• デフォルトでは制限は適用されません。リクエストのみにデフォルト値があり、指定されていない場合は自動的に適用されます。
	• コンテナ名は、ポッド仕様に表示されるとおりにリストされます。
	• サイドカーは各メイン コンテナの下にリストされます。
	• TORC `status.CurrentInstallationParams `フィールドを確認して、現在適用されている値を表示します。
		`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:`

オプション	概要	デフォルト
`httpsMetrics`	Prometheus メトリック エンドポイントに対して HTTPS を有効にします。	false
`hostNetwork`	Trident コントローラのホスト ネットワークを有効にします。これは、マルチホーム ネットワークでフロントエンド トラフィックとバックエンド トラフィックを分離する場合に便利です。	false

コントローラポッドとノードポッドについて

Tridentは単一のコントローラーポッドとして実行され、クラスター内の各ワーカーノード上のノードポッドも実行されます。ノードポッドは、Tridentボリュームをマウントする可能性のあるホスト上で実行されている必要があります。

Kubernetes"ノードセレクター"および"tolerations と taints"は、ポッドを特定のノードまたは優先ノードで実行するように制限するために使用されます。`ControllerPlugin`および `NodePlugin`を使用して、制約とオーバーライドを指定できます。

- コントローラー プラグインは、スナップショットやサイズ変更などのボリュームのプロビジョニングと管理を処理します。
- ノード プラグインは、ストレージをノードに接続する処理を行います。

```
            memory: 10Mi
          limits:
            cpu:
            memory:
        windows:
          trident-main:
            requests:
              cpu: 6m
              memory: 40Mi
            limits:
              cpu:
              memory:
          livenessprobe:
            requests:
              cpu: 6m
              memory: 40Mi
            limits:
              cpu:
              memory:
          node-driver-registrar:
            requests:
              cpu: 6m
              memory: 40Mi
            limits:
              cpu:
              memory:
      operator:
        requests:
          cpu: 2m
          memory: 40Mi
        limits:
          cpu:
          memory:
    operator:
      requests:
        cpu: 10m
        memory: 40Mi
      limits:
        cpu:
        memory:
```

Helm を使用して Trident オペレーターを展開する（オフラインモード）

Helm を使用して Trident オペレーターを導入し、Trident をインストールできます。このプロセスは、 Trident に必要なコンテナイメージがプライベートレジストリに保存されているインストールに適用されます。プライベートイメージレジストリがない場合は、"標準導入のプロセス"を使用します。

Trident 26.02に関する重要な情報

Trident に関する以下の重要な情報を必ずお読みください。

Tridentに関する重要な情報

- Kubernetes 1.35 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制し、multipath.conf ファイル内の推奨値は `find_multipaths: no`です。

  非マルチパス構成の使用、または multipath.conf ファイルでの `find_multipaths: yes`または `find_multipaths: smart`値の使用は、マウントの失敗を引き起こします。Trident は、 21.07 リリース以降 `find_multipaths: no`の使用を推奨しています。

Helmを使用してTridentオペレーターを導入し、Tridentをインストールする

Trident "Helmチャート" を使用すると、Trident オペレーターを導入し、Trident を 1 つのステップでインストールできます。

"インストールの概要"を確認して、インストールの前提条件を満たしていること、および環境に適したインストールオプションが選択されていることを確認します。

開始する前に

"導入の前提条件"に加えて"Helm バージョン3"が必要です。

プライベートリポジトリにTridentをインストールする際、 `imageRegistry`スイッチを使用してリポジトリの場所を指定する場合は、リポジトリパスに `/netapp/`を使用しないでください。

手順

1. Trident Helm リポジトリを追加します：

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

2. `helm install`を使用して、デプロイメントの名前とイメージレジストリの場所を指定します。"TridentとCSIのイメージ"は1つのレジストリまたは異なるレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例では、 `100.2602.0`はインストールするTridentのバージョンです。

1つのレジストリ内の画像

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

異なるレジストリ内のイメージ

```
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}
```

すでにTrident用の名前空間を作成している場合は、 `--create-namespace`パラメータは追加の名前空間を作成しません。

`helm list`を使用して、名前、名前空間、チャート、ステータス、アプリのバージョン、リビジョン番号などのインストールの詳細を確認できます。

インストール中に構成データを渡す

インストール中に構成データを渡す方法は2つあります：

オプション	概要
`--values`（または `-f`）	オーバーライドを含む YAML ファイルを指定します。これは複数回指定することができ、右端のファイルが優先されます。
`--set`	コマンドラインでオーバーライドを指定します。

たとえば、 `debug`のデフォルト値を変更するには、次のコマンドを実行します。ここで、 `100.2602.0`はインストールするTridentのバージョンです：

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

nodePrep値を追加するには、次のコマンドを実行します：

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

構成オプション

この表と、Helm チャートの一部である `values.yaml`ファイルは、キーとそのデフォルト値のリストを提供します。

values.yaml ファイルからデフォルトのアフィニティを削除しないでください。カスタムアフィニティを提供する場合は、デフォルトのアフィニティを拡張します。

オプション	概要	デフォルト
`nodeSelector`	Pod割り当て用のノードラベル	
`podAnnotations`	Podアノテーション	
`deploymentAnnotations`	デプロイメントアノテーション	
`tolerations`	pod割り当ての許容範囲	

オプション	概要	デフォルト

`affinity`	Pod割り当ての親和性	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedul`
		`ingIgnoredDuringExecu`
		`tion:`
		
		`nodeSelectorTerms:`
		`  -`
		`matchExpressions:`
		`  - key:`
		`kubernetes.io/arch`
		
		`operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key:`
		`kubernetes.io/os`
		
		`operator: In`
		`  values:`
		`  - linux`
		
		values.yaml ファイルからデフォルトのアフィニティを削除しないでください。カスタムアフィニティを提供する場合は、デフォルトのアフィニティを拡張します。

`tridentControllerPluginNodeSelector`	ポッドの追加ノードセレクター。詳細については、"コントローラポッドとノードポッドについて"を参照してください。	
`tridentControllerPluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。詳細については、"コントローラポッドとノードポッドについて"を参照してください。	

オプション	概要	デフォルト
`tridentNodePluginNodeSelector`	ポッドの追加ノードセレクター。詳細については、"コントローラポッドとノードポッドについて"を参照してください。	
`tridentNodePluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。詳細については、"コントローラポッドとノードポッドについて"を参照してください。	
`imageRegistry`	`trident-operator`、`trident`、およびその他のイメージのレジストリを識別します。デフォルトを受け入れる場合は空のままにします。重要：プライベートリポジトリにTridentをインストールする場合、リポジトリの場所を指定するために `imageRegistry` スイッチを使用するときは、リポジトリパスに `/netapp/` を使用しないでください。	""
`imagePullPolicy`	`trident-operator`のイメージプルポリシーを設定します。	`IfNotPresent`
`imagePullSecrets`	`trident-operator`、`trident`、およびその他のイメージのイメージプルシークレットを設定します。	
`kubeletDir`	kubelet の内部状態のホストの場所を上書きできます。	`"/var/lib/kubelet"`
`operatorLogLevel`	Tridentオペレータのログレベルを `trace`、 `debug`、 `info`、 `warn`、 `error`、または `fatal` に設定できます。	`"info"`
`operatorDebug`	Trident オペレータのログレベルをデバッグに設定できます。	`true`
`operatorImage`	`trident-operator`の画像の完全な上書きを許可します。	""
`operatorImageTag`	`trident-operator`イメージのタグを上書きできます。	""
`tridentIPv6`	Trident が IPv6 クラスターで動作できるようにします。	`false`

オプション	概要	デフォルト
`tridentK8sTimeout`	ほとんどの Kubernetes API 操作のデフォルトの 180 秒のタイムアウトをオーバーライドします（ゼロ以外の場合は秒単位）。 `tridentK8sTimeout` パラメータは、Tridentのインストールにのみ適用されます。	`180`
`tridentHttpRequestTimeout`	HTTPリクエストのデフォルトの90秒のタイムアウトを上書きします。 `0s`はタイムアウトの無限の期間です。負の値は許可されません。	`"90s"`
`tridentSilenceAutosupport`	Tridentの定期的なAutoSupportレポートを無効にすることができます。	`false`
`tridentAutosupportImageTag`	Trident AutoSupportコンテナのイメージタグを上書きできます。	`<version>`
`tridentAutosupportProxy`	Trident AutoSupport コンテナが HTTP プロキシ経由でホームに通信できるようにします。	`""`
`tridentLogFormat`	Tridentログ形式を設定します（`text`または `json`）。	`"text"`
`tridentDisableAuditLog`	Trident監査ロガーを無効にします。	`true`
`tridentLogLevel`	Tridentのログレベルを次のように設定できます： `trace`、 `debug`、 `info`、 `warn`、 `error`、または `fatal`。	`"info"`
`tridentDebug`	Tridentのログレベルを `debug`に設定できます。	`false`
`tridentLogWorkflows`	特定のTridentワークフローでトレースログまたはログ抑制を有効にできます。	`""`
`tridentLogLayers`	特定のTridentレイヤーでトレースログまたはログ抑制を有効にできます。	`""`
`tridentImage`	Tridentの画像の完全な上書きを許可します。	`""`
`tridentImageTag`	Tridentの画像のタグを上書きできます。	`""`

オプション	概要	デフォルト
`tridentProbePort`	Kubernetes の liveness/readiness プローブに使用されるデフォルトのポートをオーバーライドできます。	""
`windows`	Windows ワーカー ノードに Trident をインストールできるようにします。	`false`
`enableForceDetach`	強制デタッチ機能を有効にできます。ノード ヘルス チェック (NHC) オペレーターとの統合により、強制デタッチ プロセスを自動化できます。詳細については、"Tridentを使用したステートフルアプリケーションのフェイルオーバーの自動化"を参照してください。	`false`
`excludePodSecurityPolicy`	オペレータ Pod セキュリティ ポリシーを作成から除外します。	`false`
`nodePrep`	Tridentが指定されたデータストレージ プロトコルを使用してボリュームを管理するために、Kubernetesクラスタのノードを準備できるようにします。現在、 **`iscsi`**がサポートされている唯一の値です。 OpenShift 4.19 以降、この機能でサポートされる Trident の最小バージョンは 25.06.1 です。	

オプション	概要	デフォルト

`resources`	Trident コントローラ、ノード、およびオペレータ ポッドの Kubernetes リソース制限とリクエストを設定します。Kubernetes でのリソース割り当てを管理するために、各コンテナとサイドカーの CPU とメモリを設定できます。
	
	リソース要求と制限の設定の詳細については、"Pod と Container のリソース管理"を参照してください。
	
	• コンテナまたはフィールドの名前を変更しないでください。
	• インデントを変更しないでください。YAML インデントは適切な解析に重要です。
	
	• デフォルトでは制限は適用されません。リクエストのみにデフォルト値が設定されます。
	• コンテナ名は、ポッド仕様に表示されるとおりにリストされます。
	• サイドカーは各メイン コンテナの下にリストされます。
	• TORC `status.CurrentInstallationParams `フィールドを確認して、現在適用されている値を表示します。
		`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オペレータのインストールをカスタマイズ

Trident オペレータを使用すると、 `TridentOrchestrator` spec の属性を使用して Trident のインストールをカスタマイズできます。 `TridentOrchestrator` 引数で許可される範囲を超えてインストールをカスタマイズする場合は、 `tridentctl` を使用して、必要に応じて変更できるカスタム YAML マニフェストを生成することを検討してください。

コントローラポッドとノードポッドについて

Tridentは、クラスター内の各ワーカー ノード上で単一のコントローラー ポッドとノード ポッドとして実行されます。ノードポッドは、Tridentボリュームをマウントする可能性のあるホスト上で実行されている必要があります。

Kubernetes"ノードセレクター"および"tolerations と taints"は、ポッドを特定のノードまたは優先ノードで実行するように制限するために使用されます。`ControllerPlugin`および `NodePlugin`を使用して、制約とオーバーライドを指定できます。

- コントローラー プラグインは、スナップショットやサイズ変更などのボリュームのプロビジョニングと管理を処理します。
- ノード プラグインは、ストレージをノードに接続する処理を行います。

構成オプション

> `spec.namespace`は `TridentOrchestrator`で指定され、Tridentがインストールされている名前空間を示します。このパラメータは*Tridentのインストール後は更新できません。更新しようとすると、 `TridentOrchestrator`のステータスが `Failed`に変更されます。Tridentは名前空間間で移行することを意図していません。

この表には `TridentOrchestrator`属性の詳細が記載されています。

パラメータ	概要	デフォルト
`namespace`	Tridentをインストールする名前空間	`"default"`
`debug`	Tridentのデバッグを有効にする	`false`
`enableForceDetach`	`ontap-san`、 `ontap-san-economy`、 `ontap-nas`、および `ontap-nas-economy`のみ。Kubernetes Non-Graceful Node Shutdown（NGNS）と連携して、ノードが正常でなくなった場合に、マウントされたボリュームを持つワークロードを新しいノードに安全に移行する機能をクラスター管理者に付与します。詳細については、"Tridentを使用したステートフルアプリケーションのフェイルオーバーの自動化"を参照してください。	`false`
`windows`	`true`に設定すると、Windows ワーカー ノードへのインストールが有効になります。	`false`

パラメータ	概要	デフォルト
`cloudProvider`	AKS クラスターでマネージド ID またはクラウド ID を使用する場合は `"Azure"` に設定します。 `"AWS"` に設定します（EKS クラスターでクラウド ID を使用する場合）。 `"GCP"` に設定します（GKE クラスターでクラウド ID を使用する場合）。	`""`
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（"azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（'"eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role"'）に設定します。GKEクラスタでクラウドIDを使用する場合は、クラウドID（'"iam.gke.io/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com"'）に設定します。	`""`
`IPv6`	IPv6経由でTridentをインストール	false
`k8sTimeout`	Kubernetes 操作のタイムアウト。 `k8sTimeout`パラメータは、Tridentのインストールにのみ適用されます。	`180sec`
`silenceAutosupport`	AutoSupport バンドルを NetApp に自動的に送信しない	`false`
`autosupportImage`	Autosupport Telemetryのコンテナイメージ	`"netapp/trident-autosupport10"`
`autosupportProxy`	Autosupport テレメトリを送信するためのプロキシのアドレス/ポート	`"http://proxy.example.com:8888"`
`uninstall`	Tridentをアンインストールするために使用するフラグ	`false`
`logFormat`	Tridentで使用するログ形式 [text,json]	`"text"`
`tridentImage`	インストールする Trident イメージ	`"netapp/trident:26.02"`
`imageRegistry`	内部レジストリへのパス。形式は `<registry FQDN>[:port][/subpath]`	`"registry.k8s.io"`
`kubeletDir`	ホスト上の kubelet ディレクトリへのパス	`"/var/lib/kubelet"`
`wipeout`	Tridentを完全に削除するために削除するリソースのリスト	
`imagePullSecrets`	内部レジストリからイメージをプルするためのシークレット	

パラメータ	概要	デフォルト
`imagePullPolicy`	Trident オペレータのイメージプルポリシーを設定します。有効な値は次のとおりです： `Always` 常にイメージをプルします。 `IfNotPresent` ノード上にイメージがまだ存在しない場合にのみイメージをプルします。 `Never` イメージを決してプルしません。	`IfNotPresent`
`controllerPluginNodeSelector`	ポッドの追加ノードセレクター。`pod.spec.nodeSelector`と同じ形式に従います。	デフォルトなし；オプション
`controllerPluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。 `pod.spec.Tolerations`と同じ形式に従います。	デフォルトなし；オプション
`nodePluginNodeSelector`	ポッドの追加ノードセレクター。`pod.spec.nodeSelector`と同じ形式に従います。	デフォルトなし；オプション
`nodePluginTolerations`	ポッドの Kubernetes 許容値をオーバーライドします。 `pod.spec.Tolerations`と同じ形式に従います。	デフォルトなし；オプション
`nodePrep`	Tridentが指定されたデータストレージ プロトコルを使用してボリュームを管理するために、Kubernetesクラスタのノードを準備できるようにします。現在、 **`iscsi`**がサポートされている唯一の値です。 OpenShift 4.19 以降、この機能でサポートされる Trident の最小バージョンは 25.06.1 です。	
`k8sAPIQPS`	Kubernetes API サーバーと通信するときにコントローラーが使用する 1 秒あたりのクエリ数（QPS）の制限。バースト値は QPS 値に基づいて自動的に設定されます。	`100`；オプション
`enableConcurrency`	スループットを向上させるために、Tridentコントローラーの同時操作を可能にします。 **Tech Preview**：この機能は試験的なもので、現在、ONTAP-NAS（NFSのみ）およびONTAP-SAN（統合ONTAP 9のNVMe）ドライバを使用した限定的な並列ワークフローをサポートしています。これは、既存のONTAP-SANドライバ（統合ONTAP 9のiSCSIおよびFCPストレージ プロトコル）のTech Previewに加えてのものです。	false

パラメータ	概要	デフォルト
`resources`	Trident コントローラーとノード ポッドの Kubernetes リソース制限とリクエストを設定します。Kubernetes でのリソース割り当てを管理するために、各コンテナとサイドカーの CPU とメモリを設定できます。 リソース要求と制限の設定の詳細については、"Pod と Container のリソース管理"を参照してください。 • コンテナまたはフィールドの名前を変更しないでください。 • インデントを変更しないでください。YAML インデントは適切な解析に重要です。 • デフォルトでは制限は適用されません。リクエストのみにデフォルト値があり、指定されていない場合は自動的に適用されます。 • コンテナ名は、ポッド仕様に表示されるとおりにリストされます。 • サイドカーは各メイン コンテナの下にリストされます。 • TORC `status.CurrentInstallationParams` フィールドを確認して、現在適用されている値を表示します。	`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:`

パラメータ	概要	デフォルト
`httpsMetrics`	Prometheus メトリック エンドポイントに対して HTTPS を有効にします。	false
`hostNetwork`	Trident コントローラのホスト ネットワークを有効にします。これは、マルチホーム ネットワークでフロントエンド トラフィックとバックエンド トラフィックを分離する場合に便利です。	false

ポッドパラメータのフォーマットの詳細については、"Pod をノードに割り当てる"を参照してください。

サンプル構成

[構成オプション]の属性を使用して `TridentOrchestrator`を定義し、インストールをカスタマイズできます。

基本的なカスタム設定

この例は、 `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml`コマンドの実行後に作成されたもので、基本的なカスタム インストールを表します：

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

```
30Mi
        limits:
          cpu:
          memory:
    node:
      linux:
        trident-
main:
          cpu:
1m

memory: 10Mi
        limits:
          cpu:

memory:
    windows:
      trident-
main:

requests:
          cpu:
6m

memory: 40Mi
        limits:
```

ノードセレクタ

この例では、ノードセレクターを使用してTridentをインストールします。

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

Windows ワーカーノード

この例は、 `cat deploy/crds/tridentorchestrator_cr.yaml`コマンドの実行後に作成され、Windows ワーカー ノードに Trident をインストールします。

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

AKS クラスタ上の管理対象 ID

この例では、AKS クラスターでマネージド ID を有効にするために Trident をインストールします。

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

AKS クラスタのクラウド ID

この例では、AKS クラスター上のクラウド ID で使用するために Trident をインストールします。

```
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'
```

EKS クラスタのクラウド ID

この例では、AKS クラスター上のクラウド ID で使用するために Trident をインストールします。

```
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'"
```

GKE のクラウド ID

この例では、GKE クラスタ上のクラウド ID で使用するために Trident をインストールします。

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

Tridentコントローラーと**Trident Linux**ノードポッド用の**Kubernetes**リソースリクエストと制限の設定

この例では、Tridentコントローラーと Trident Linuxノードポッドに対するKubernetesリソースリクエストと制限を設定します。

免責事項：この例で示されている要求値と制限値は、デモンストレーションのみを目的としています。環境とワークロードの要件に基づいてこれらの値を調整してください。

```
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
```

Kubernetes のリソースリクエストと制限の設定（**Trident** コントローラーと **Trident Windows** および **Linux** ノードポッド）

この例では、TridentコントローラーとTrident WindowsおよびLinuxノードポッドのKubernetesリソースリクエストと制限を設定します。

免責事項：この例で示されている要求値と制限値は、デモンストレーションのみを目的としています。環境とワークロードの要件に基づいてこれらの値を調整してください。

```
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
```