Tridentを使用

Trident

NetApp
July 01, 2026

目次

Tridentを使用

ワーカーノードを準備する

Kubernetes クラスター内のすべてのワーカーノードは、ポッド用にプロビジョニングしたボリュームをマウントできる必要があります。ワーカーノードを準備するには、ドライバーの選択に基づいて、NFS、iSCSI、NVMe/TCP、または FC ツールをインストールする必要があります。

適切なツールの選択

複数のドライバーを組み合わせて使用している場合は、ドライバーに必要なすべてのツールをインストールする必要があります。Red Hat Enterprise Linux CoreOS (RHCOS) の最近のバージョンには、ツールがデフォルトでインストールされています。

NFSツール

"NFSツールをインストールする"使用している場合： `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、または `azure-netapp-files`。

iSCSIツール

"iSCSIツールをインストールする"使用している場合： `ontap-san`、 `ontap-san-economy`、 `solidfire-san`。

NVMeツール

"NVMeツールをインストールする"Nonvolatile Memory Express（NVMe）over TCP（NVMe/TCP）プロトコルに `ontap-san`を使用している場合。

NetAppでは、NVMe/TCPにはONTAP 9.12以降を推奨しています。

SCSI over FCツール

FC および FC-NVMe SAN ホストの設定の詳細については、"FCおよびFC-NVMe SANホストの構成方法"を参照してください。

"FCツールをインストールする" sanType を `ontap-san`（SCSI over FC）で使用している場合 `fcp`。

考慮すべき点： * SCSI over FCはOpenShiftおよびKubeVirt環境でサポートされています。* SCSI over FCはDockerではサポートされていません。* iSCSI自己修復はSCSI over FCには適用されません。

SMBツール

"SMB ボリュームのプロビジョニングの準備"使用している場合： `ontap-nas`SMBボリュームをプロビジョニングします。

ノードサービス検出

Tridentは、ノードがiSCSIまたはNFSサービスを実行できるかどうかを自動的に検出しようとします。

ノード サービス検出では検出されたサービスを識別しますが、サービスが適切に構成されていることは保証されません。逆に、検出されたサービスが存在しない場合でも、ボリュームのマウントが失敗することは保証されません。

イベントを確認する

Tridentは、検出されたサービスを識別するためにノードのイベントを作成します。これらのイベントを確認するには、次のコマンドを実行します：

```
kubectl get event -A --field-selector involvedObject.name=<Kubernetes node name>
```

検出されたサービスを確認する

Tridentは、TridentノードCRの各ノードで有効になっているサービスを識別します。検出されたサービスを表示するには、次のコマンドを実行します：

```
tridentctl get node -o wide -n <Trident namespace>
```

NFSボリューム

オペレーティングシステムのコマンドを使用してNFSツールをインストールします。NFSサービスがブート時に起動されることを確認します。

RHEL 8以降

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

ボリュームをコンテナに接続するときに障害が発生しないように、NFSツールをインストールした後、ワーカーノードを再起動します。

iSCSIボリューム

Tridentは、iSCSIセッションを自動的に確立し、LUNをスキャンし、マルチパスデバイスを検出し、フォーマットして、ポッドにマウントできます。

iSCSI自己修復機能

ONTAPシステムの場合、TridentはiSCSI自己修復を5分ごとに実行します：

1. 必要な iSCSI セッション状態と現在の iSCSI セッション状態を*識別*します。

2. 必要な修復を特定するには、希望する状態と現在の状態を*比較*します。Tridentは修復の優先順位と修復を先取りするタイミングを決定します。
3. 現在のiSCSIセッション状態を目的のiSCSIセッション状態に戻すために必要な*修復を実行*します。

> ⓘ 自己修復アクティビティのログは、それぞれのDaemonsetポッド上の `trident-main`コンテナにあります。ログを表示するには、Tridentのインストール時に `debug`を「true」に設定しておく必要があります。

Trident iSCSI自己修復機能により、次のことを防ぐことができます。

- ネットワーク接続の問題が発生した後に発生する可能性のある、古いまたは異常な iSCSI セッション。古いセッションの場合、Trident はポータルとの接続を再確立するためにログアウトする前に 7 分間待機します。

> ⓘ たとえば、ストレージコントローラ上でCHAPシークレットがローテーションされ、ネットワークの接続が失われた場合、古い（*stale*）CHAPシークレットが残る可能性があります。自己修復はこれを認識し、セッションを自動的に再確立して更新されたCHAPシークレットを適用します。

- iSCSIセッションが見つかりません
- 不足しているLUN

Tridentをアップグレードする前に考慮すべき点

- ノードごとのigroup（23.04以降で導入）のみが使用されている場合、iSCSI自己修復はSCSIバス内のすべてのデバイスに対してSCSI再スキャンを開始します。
- バックエンドスコープのigroup（23.04以降は非推奨）のみが使用されている場合、iSCSI自己修復はSCSIバス内の正確なLUN IDのSCSI再スキャンを開始します。
- ノードごとのigroupとバックエンドスコープのigroupが混在して使用されている場合、iSCSI自己修復はSCSIバス内の正確なLUN IDのSCSI再スキャンを開始します。

iSCSIツールをインストールする

オペレーティングシステムのコマンドを使用してiSCSIツールをインストールします。

開始する前に

- Kubernetes クラスター内の各ノードには一意の IQN が必要です。これは必須の前提条件です。
- RHCOSバージョン4.5以降、またはその他のRHEL互換Linuxディストリビューションを `solidfire-san`ドライバおよびElement OS 12.5以前とともに使用している場合は、 `/etc/iscsi/iscsid.conf`でCHAP認証アルゴリズムがMD5に設定されていることを確認してください。安全なFIPS準拠のCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256は、Element 12.7で利用できます。

```
sudo sed -i 's/^\(node.session.auth.chap_algs\).*/\1 = MD5/'
/etc/iscsi/iscsid.conf
```

- iSCSI PVを使用するRHEL/Red Hat Enterprise Linux CoreOS（RHCOS）で動作するワーカーノードを利用する場合、インラインスペースの再利用を実行するために、 `discard` StorageClass内でmountOption

を指定してください。参照 "Red Hat ドキュメント"。

- 最新バージョンの `multipath-tools`にアップグレードしたことを確認してください。

RHEL 8以降

1. 次のシステムパッケージをインストールします：

```
sudo yum install -y lsscsi iscsi-initiator-utils device-mapper-
multipath
```

2. iscsi-initiator-utils のバージョンが 6.2.0.874-2.el7 以降であることを確認します：

```
rpm -q iscsi-initiator-utils
```

3. スキャンを手動に設定します：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
```

4. マルチパスを有効にする：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> `/etc/multipath.conf`に `find_multipaths no`が `defaults`の下に含まれていることを確認します。

5. `iscsid`と `multipathd`が実行されていることを確認します：

```
sudo systemctl enable --now iscsid multipathd
```

6. 有効化して起動 `iscsi`：

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. 次のシステムパッケージをインストールします：

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools
scsitools
```

2. open-iscsi のバージョンが 2.0.874-5ubuntu2.10 以降（bionic の場合）または 2.0.874-7.1ubuntu6.1 以降（focal の場合）であることを確認します：

```
dpkg -l open-iscsi
```

3. スキャンを手動に設定します：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
```

4. マルチパスを有効にする：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> ⓘ `/etc/multipath.conf`に `find_multipaths no`が `defaults`の下に含まれていることを確認します。

5. `open-iscsi`と `multipath-tools`が有効になっていて実行中であることを確認します：

```
sudo systemctl status multipath-tools
sudo systemctl enable --now open-iscsi.service
sudo systemctl status open-iscsi
```

> ⓘ Ubuntu 18.04の場合、iSCSIデーモンを起動する前に `iscsiadm`でターゲットポートを検出する必要があります。iSCSIデーモンが起動するためです。 `open-iscsi`または、`iscsi`サービスを修正して `iscsid`を自動的に起動することもできます。

iSCSI self-healingを設定または無効にする

以下のTrident iSCSI自己修復設定を構成して、古いセッションを修正できます：

- **iSCSI**自己修復間隔：iSCSI自己修復が呼び出される頻度を決定します（デフォルト：5分）。小さい数値を設定すると実行頻度が高くなり、大きい数値を設定すると実行頻度が低くなるように設定できます。

> ⓘ iSCSI自己修復間隔を0に設定すると、iSCSI自己修復が完全に停止します。iSCSI自己修復を無効にすることは推奨しません。iSCSI自己修復が意図したとおりに動作していない特定のシナリオ、またはデバッグ目的の場合にのみ無効にする必要があります。

- **iSCSI**自己修復待機時間：iSCSI自己修復が異常なセッションからログアウトして再度ログインを試行するまでの待機時間を決定します（デフォルト：7分）。大きい数値に設定すると、正常でないと判断されたセッションはログアウトされるまでの待機時間が長くなり、その後再度ログインが試行されます。また、小さい数値に設定すると、ログアウトしてから再度ログインするまでの時間が短くなります。

Helm

iSCSI自己修復設定を構成または変更するには、helmのインストールまたはhelmの更新中に `iscsiSelfHealingInterval`および `iscsiSelfHealingWaitTime`パラメータを渡します。

次の例では、iSCSI自己修復間隔を3分にし、自己修復待ち時間を6分にします：

```
helm install trident trident-operator-100.2602.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

tridentctl

iSCSI自己修復設定を構成または変更するには、tridentctlのインストールまたは更新中に `iscsi-self-healing-interval`および `iscsi-self-healing-wait-time`パラメータを渡します。

次の例では、iSCSI自己修復間隔を3分にし、自己修復待ち時間を6分にします：

```
tridentctl install --iscsi-self-healing-interval=3m0s --iscsi-self
-healing-wait-time=6m0s -n trident
```

NVMe/TCPボリューム

オペレーティングシステムのコマンドを使用してNVMeツールをインストールします。

- NVMe には RHEL 9 以降が必要です。
- Kubernetes ノードのカーネルバージョンが古すぎる場合、またはカーネルバージョンで NVMe パッケージが利用できない場合は、ノードのカーネルバージョンを NVMe パッケージを含むバージョンに更新する必要がある場合があります。

RHEL 9

```
sudo yum install nvme-cli
sudo yum install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

Ubuntu

```
sudo apt install nvme-cli
sudo apt -y install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

インストールの検証

インストール後、次のコマンドを使用して、Kubernetesクラスタ内の各ノードに一意のNQNがあることを確認します：

```
cat /etc/nvme/hostnqn
```

Tridentは、パスがダウンした場合にNVMeがパスを放棄しないようにするために`ctrl_device_tmo`の値を変更します。この設定は変更しないでください。

SCSI over FCボリューム

Tridentでファイバチャネル（FC）プロトコルを使用して、ONTAPシステム上のストレージリソースをプロビジョニングおよび管理できるようになりました。

前提条件

FC に必要なネットワークとノードの設定を構成します。

ネットワーク設定

1. ターゲットインターフェイスのWWPNを取得します。詳細については、 "network interface show"を参照してください。
2. イニシエーター（ホスト）上のインターフェースのWWPNを取得します。

   対応するホストオペレーティングシステムユーティリティを参照してください。
3. ホストとターゲットのWWPNを使用してFCスイッチのゾーニングを構成します。

   詳細については、それぞれのスイッチベンダーのドキュメントを参照してください。

   詳細については、次のONTAPドキュメントを参照してください：

- "Fibre ChannelおよびFCoEのゾーニング - 概要"
- "FCおよびFC-NVMe SANホストの構成方法"

FCツールをインストールする

オペレーティングシステムのコマンドを使用してFCツールをインストールします。

- FC PVを使用するRHEL／Red Hat Enterprise Linux CoreOS（RHCOS）上のワーカーノードを利用する場合、インラインスペースリクレームを実行するために、 `discard` StorageClassでmountOptionを指定してください。参照 "Red Hat ドキュメント"。

RHEL 8以降

1. 次のシステムパッケージをインストールします：

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. マルチパスを有効にする：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> `/etc/multipath.conf`に `find_multipaths no`が `defaults`の下に含まれていることを確認します。

3. `multipathd`が実行されていることを確認します：

```
sudo systemctl enable --now multipathd
```

Ubuntu

1. 次のシステムパッケージをインストールします：

```
sudo apt-get install -y lsscsi sg3-utils multipath-tools scsitools
```

2. マルチパスを有効にする：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> `/etc/multipath.conf`に `find_multipaths no`が `defaults`の下に含まれていることを確認します。

3. `multipath-tools`が有効になっていて実行中であることを確認します：

```
sudo systemctl status multipath-tools
```

SMB ボリュームのプロビジョニングの準備

`ontap-nas`ドライバーを使用してSMBボリュームをプロビジョニングできます。

ONTAP オンプレミスクラスタ用の `ontap-nas-economy` SMB ボリュームを作成するには、SVM で NFS と SMB/CIFS プロトコルの両方を設定する必要があります。これらのプロトコルのいずれかを設定しないと、SMB ボリュームの作成が失敗します。

`autoExportPolicy` は SMB ボリュームではサポートされません。

開始する前に

SMB ボリュームをプロビジョニングする前に、次のものが必要です。

- Linux コントローラー ノードと、Windows Server 2022 を実行する少なくとも 1 つの Windows ワーカー ノードを備えた Kubernetes クラスター。Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。
- Active Directory のクレデンシャルを含む少なくとも 1 つの Trident シークレット。シークレットを生成するには `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- Windows サービスとして構成された CSI プロキシ。 `csi-proxy`を設定するには、Windows 上で実行されている Kubernetes ノード用の"GitHub：CSI Proxy"または"GitHub：Windows用CSIプロキシ"を参照してください。

手順

1. オンプレミス ONTAP の場合、必要に応じて SMB 共有を作成するか、 Trident に作成させることができます。

   SMB 共有は Amazon FSx for ONTAP に必要です。

   SMB 管理共有は、"Microsoft管理コンソール" 共有フォルダスナップインまたは ONTAP CLI のいずれかを使用して、2 つの方法のいずれかで作成できます。ONTAP CLI を使用して SMB 共有を作成するには：

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

      `vserver cifs share create`コマンドは、共有の作成中に `-path` オプションで指定されたパスをチェックします。指定されたパスが存在しない場合、コマンドは失敗します。

   b. 指定された SVM に関連付けられた SMB 共有を作成します：

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

c. 共有が作成されたことを確認します：

```
vserver cifs share show -share-name share_name
```

詳細については、"SMB共有を作成する"を参照してください。

2. バックエンドを作成するときは、SMB ボリュームを指定するために以下を構成する必要があります。すべての FSx for ONTAP バックエンドの設定オプションについては、"FSx for ONTAP 設定オプションと例"を参照してください。

パラメータ	概要	例
`smbShare`	次のいずれかを指定できます：Microsoft Management ConsoleまたはONTAP CLIを使用して作成されたSMB共有の名前、TridentがSMB共有を作成できるようにする名前、またはパラメータを空白のままにしてボリュームへの共通共有アクセスを防止できます。このパラメータは、オンプレミスONTAPではオプションです。このパラメータは、Amazon FSx for ONTAPバックエンドでは必須であり、空白にすることはできません。	`smb-share`
`nasType`	* `smb`に設定する必要があります。*nullの場合、デフォルトは `nfs`です。	`smb`
`securityStyle`	新しいボリュームのセキュリティ スタイル。**SMB** ボリュームの場合は、 **`ntfs`**または **`mixed`**に設定する必要があります。	`ntfs` または `mixed` SMB ボリューム用
`unixPermissions`	新しいボリュームのモード。**SMB** ボリュームの場合は空のままにする必要があります。	""

バックエンドの設定と管理

バックエンドを設定

バックエンドは、Trident とストレージ システム間の関係を定義します。Trident がそのストレージ システムと通信する方法と、Trident がそこからボリュームをプロビジョニングする方法を指定します。

Tridentは、ストレージ クラスによって定義された要件に一致するバックエンドからストレージ プールを自動的に提供します。ストレージ システムのバックエンドを設定する方法を確認してください。

- "Azure NetApp Files バックエンドを構成する"

- "Google Cloud NetApp Volumes バックエンドを設定する"
- "NetApp HCI または SolidFire バックエンドを設定する"
- "ONTAP または Cloud Volumes ONTAP NAS ドライバを使用してバックエンドを設定する"
- "ONTAP または Cloud Volumes ONTAP SAN ドライバを使用してバックエンドを設定する"
- "Amazon FSx for NetApp ONTAP で Trident を使用"

Azure NetApp Files

Azure NetApp Files バックエンドを構成する

Azure NetApp Files を Trident のバックエンドとして使用します。このバックエンドは NFS および SMB ボリュームをサポートしています。Trident は Azure Kubernetes Service （AKS） クラスターのマネージド ID とワークロード ID をサポートします。

サポートされている**Azure**クラウド環境

Tridentは、複数のAzureクラウド環境でAzure NetApp Filesバックエンドをサポートしています。

サポートされている Azure クラウドは以下のとおりです：

- Azure Commercial
- Azure Government（Azure Government / MAG）

Trident をデプロイするか、Azure NetApp Files バックエンドを設定する際は、Azure Resource Manager と認証エンドポイントが Azure クラウド環境と一致していることを確認してください。

Azure NetApp Files ドライバサポートの確認

Tridentは、以下のAzure NetApp Filesストレージドライバを提供します。

サポートされているアクセスモードには、*ReadWriteOnce* （RWO）、*ReadOnlyMany* （ROX ）、*ReadWriteMany* （RWX）、および *ReadWriteOncePod* （RWOP）があります。

Driver	プロトコル	**volumeMode**	サポートされているアクセスモード	サポートされているファイルシステム
`azure-netapp-files`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	`nfs`, `smb`

レビューに関する考慮事項

- Azure NetApp Files は 50 GiB 未満のボリュームをサポートしていません。Trident は、より小さなボリュームが要求された場合でも、50 GiB のボリュームを作成します。
- Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。
- Azure NetApp Files を非商用 Azure クラウドにデプロイするには、クラウド固有の Azure Resource Manager と認証エンドポイントが必要です。Trident およびすべてのバックエンド構成で、Azure クラウ

ド環境に適したエンドポイントを使用していることを確認してください。

AKS にマネージド ID を使用する

TridentはAKSクラスター向けに"マネージド ID"をサポートしています。

`tridentctl`を使用して `Azure NetApp Files` バックエンドを作成または管理する場合は、正しい `Azure` クラウド環境向けに構成されていることを確認してください。

マネージドIDを使用するには、以下のものが必要です：

- AKS を使用してデプロイされた Kubernetes クラスタ
- AKS Kubernetes クラスターで構成された管理対象 ID
- Tridentが `cloudProvider`を `"Azure"`に設定してインストール済み

Trident オペレータ

編集 `tridentorchestrator_cr.yaml`して `cloudProvider`を `"Azure"`に設定します。

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

Helm

次の例では、Tridentをインストールし、環境変数 `$CP`を使用して `cloudProvider`を設定します：

```
helm install trident trident-operator-100.2602.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

次の例では Trident をインストールし、 `cloud-provider` フラグを `Azure`に設定します：

```
tridentctl install --cloud-provider="Azure" -n trident
```

AKS のワークロード **ID** を使用する

ワークロードIDを使用すると、KubernetesポッドはワークロードIDとして認証することでAzureリソースにアクセスできるようになります。

`tridentctl`を使用して `Azure NetApp Files` バックエンドを作成または管理する場合は、正しい `Azure` クラウド環境向けに構成されていることを確認してください。

ワークロードIDを使用するには、以下のものが必要です。

- AKS を使用してデプロイされた Kubernetes クラスタ
- AKS Kubernetes クラスタで設定されたワークロード ID と oidc-issuer
- Tridentは `cloudProvider`を `"Azure"`に設定し、 `cloudIdentity`をワークロード識別値に設定してインストール済み

Trident オペレータ

`tridentorchestrator_cr.yaml`を編集し、 `cloudProvider`を`"Azure"`に設定します。 `cloudIdentity`を`azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`に設定します。

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

Helm

以下の環境変数を使用して、**cloud-provider (CP)** および **cloud-identity (CI)** フラグの値を設定します。

```
export CP="Azure"
export CI="'azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx'"
```

次の例では、Tridentをインストールし、 `$CP`を使用して `cloudProvider`を設定し、 `$CI`を使用して `cloudIdentity`を設定します：

```
helm install trident trident-operator-100.6.0.tgz --set cloudProvider=$CP --set cloudIdentity="$CI"
```

<code>tridentctl</code>

クラウド プロバイダ および クラウド **ID** フラグの値を、以下の環境変数を使用して設定します：

```
export CP="Azure"
export CI="azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"
```

次の例では、Trident をインストールし、 `cloud-provider`を `$CP`に、 `cloud-identity`を `$CI`に設定します：

```
tridentctl install --cloud-provider=$CP --cloud-identity="$CI" -n
trident
```

Azure NetApp Files バックエンドを設定するための準備

Azure NetApp Files バックエンドを設定する前に、次の要件が満たされていることを確認する必要があります。

サポートされている**Azure**クラウド環境

Tridentは、複数のAzureクラウド環境でAzure NetApp Filesバックエンドをサポートしています。

サポートされている Azure クラウドは以下のとおりです：

- Azure Commercial
- Azure Government（Azure Government / MAG）

環境を準備する際は、Azure サブスクリプション、ID 構成、および Azure NetApp Files リソースが適切な Azure クラウド環境に作成されていることを確認してください。

NFSおよび**SMB**ボリュームの前提条件

Azure NetApp Files を初めて使用する場合、または新しい場所で使用する場合は、Azure NetApp Files をセットアップして NFS ボリュームを作成するための初期設定が必要です。 "Azure：Azure NetApp Files のセットアップと NFS ボリュームの作成"を参照してください。

```
https://azure.microsoft.com/en-us/products/netapp/["Azure NetApp
Files"^]バックエンドを設定して使用するには、次のものが必要です：
```

- `subscriptionID`、 `tenantID`、 `clientID`、 `location`、および `clientSecret`は、AKS クラスターでマネージド ID を使用する場合はオプションです。
- `tenantID`、 `clientID`、および `clientSecret`は、AKSクラスターでクラウドIDを使用する場合はオプションです。
- Azure NetApp Files を非商用 Azure クラウドにデプロイするには、クラウド固有の Azure Resource Manager と認証エンドポイントが必要です。Trident およびすべてのバックエンド構成で、Azure クラウド環境に適したエンドポイントを使用していることを確認してください。

- 容量プール。"Microsoft：Azure NetApp Files の容量プールを作成する"を参照してください。
- Azure NetApp Files に委任されたサブネット。"Microsoft：サブネットを Azure NetApp Files に委任する"を参照してください。
- `subscriptionID`Azure NetApp Files が有効になっている Azure サブスクリプションから。
- `tenantID`、 `clientID`、および `clientSecret`は、十分な権限を持つAzure Active Directoryの"アプリ登録"から、Azure NetApp Filesサービスに対して使用されます。アプリ登録は次のいずれかを使用する必要

があります：

- 所有者または貢献者の役割"Azure によって事前定義済み"。
- "カスタム Contributor ロール"サブスクリプションレベルで(`assignableScopes`）Tridentが必要とするものだけに制限された以下の権限を持ちます。カスタムロールを作成したら、"Azure ポータルを使用してロールを割り当てる"。

カスタム貢献者ロール

```
{
  "id": "/subscriptions/<subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-
definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTarge
ts/read",
          "Microsoft.Network/virtualNetworks/read",
          "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
```

```
ions/delete",
          "Microsoft.Features/features/read",
          "Microsoft.Features/operations/read",
          "Microsoft.Features/providers/features/read",

"Microsoft.Features/providers/features/register/action",

"Microsoft.Features/providers/features/unregister/action",

"Microsoft.Features/subscriptionFeatureRegistrations/read"
        ],
        "notActions": [],
        "dataActions": [],
        "notDataActions": []
      }
    ]
  }
}
```

- Azure `location`少なくとも1つを含む "委任されたサブネット"。Trident 22.01の時点で、 `location`パラメータは、バックエンド構成ファイルの最上位レベルの必須フィールドです。仮想プールで指定された場所の値は無視されます。
- `Cloud Identity`を使用するには、 `client ID`を "ユーザー割り当てマネージド ID"から取得し、そのIDを `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`で指定します。

SMB ボリュームの追加要件

SMB ボリュームを作成するには、次のものが必要です：

- Active Directory が構成され、Azure NetApp Files に接続されている。"Microsoft：Azure NetApp Files の Active Directory 接続の作成と管理"を参照してください。
- Linux コントローラー ノードと、Windows Server 2022 を実行する少なくとも 1 つの Windows ワーカー ノードを備えた Kubernetes クラスター。Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。
- Azure NetApp Files が Active Directory に対して認証できるように、Active Directory の資格情報を含む Trident シークレットが少なくとも 1 つ必要です。シークレットを生成するには `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- Windows サービスとして構成された CSI プロキシ。 `csi-proxy`を設定するには、Windows 上で実行されている Kubernetes ノード用の"GitHub：CSI Proxy"または"GitHub：Windows用CSIプロキシ"を参照してください。

Azure NetApp Files バックエンドの設定オプションと例

Azure NetApp Files の NFS および SMB バックエンド構成オプションについて学習し、構成例を確認します。

バックエンド構成オプション

Tridentは、バックエンド構成（サブネット、仮想ネットワーク、サービスレベル、および場所）を使用して、要求された場所で利用可能で、要求されたサービスレベルとサブネットに一致する容量プール上にAzure NetApp Filesボリュームを作成します。

Azure NetApp Files バックエンドには、次の設定オプションがあります。

パラメータ	概要	デフォルト
`version`	バックエンド構成バージョン。	常に1
`storageDriverName`	ストレージドライバーの名前	"azure-netapp-files"
`backendName`	ストレージバックエンドのカスタム名	ドライバ名 + "_" + ランダムな文字
`subscriptionID`	Azure サブスクリプションのサブスクリプション ID。AKS クラスタでマネージド ID が有効になっている場合はオプションです。	
`tenantID`	アプリ登録からのテナント ID 。AKS クラスタでマネージド ID またはクラウド ID が使用される場合はオプションです。	
`clientID`	アプリ登録からのクライアント ID。AKS クラスターでマネージド ID またはクラウド ID が使用される場合はオプションです。	
`clientSecret`	アプリ登録からのクライアントシークレット。AKS クラスタでマネージド ID またはクラウド ID が使用される場合はオプションです。	
`serviceLevel`	`Standard`、 `Premium`、または `Ultra`のいずれか	""（ランダム）
`location`	新しいボリュームが作成される Azure の場所の名前。AKS クラスターでマネージド ID が有効になっている場合はオプションです。	
`resourceGroups`	検出されたリソースをフィルタリングするためのリソースグループのリスト	"[]"（フィルタなし）
`netappAccounts`	検出されたリソースをフィルタリングするためのNetAppアカウントのリスト	"[]"（フィルタなし）

パラメータ	概要	デフォルト
`capacityPools`	検出されたリソースをフィルタリングするための容量プールのリスト	"[]"（フィルターなし、ランダム）
`virtualNetwork`	委任されたサブネットを持つ仮想ネットワークの名前	""
`subnet`	委任されたサブネットの名前 `Microsoft.Netapp/volumes`	""
`networkFeatures`	ボリュームの VNet 機能のセット。`Basic`または `Standard`を指定できます。ネットワーク機能はすべてのリージョンで利用できるわけではなく、サブスクリプションで有効にする必要がある場合があります。 `networkFeatures`を指定した場合、この機能が有効になっていないと、ボリュームのプロビジョニングが失敗します。	""
`nfsMountOptions`	NFSマウントオプションをきめ細かく制御できます。SMBボリュームでは無視されます。NFSバージョン4.1を使用してボリュームをマウントするには、カンマ区切りのマウントオプションリストに `nfsvers=4`を含めてNFS v4.1を選択します。ストレージクラス定義で設定されたマウントオプションは、バックエンド構成で設定されたマウントオプションを上書きします。	"nfsvers=3"
`limitVolumeSize`	要求されたボリュームサイズがこの値を超える場合、プロビジョニングに失敗します	""（デフォルトでは強制されません）
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例： `\{"api": false, "method": true, "discovery": true}`。トラブルシューティングを行っており、詳細なログダンプが必要な場合を除き、これを使用しないでください。	null
`nasType`	NFS または SMB ボリュームの作成を設定します。オプションは `nfs`、 `smb`、またはnullです。nullに設定すると、デフォルトで NFS ボリュームになります。	`nfs`

パラメータ	概要	デフォルト
`supportedTopologies`	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、"CSI トポロジを使用する"を参照してください。	
`qosType`	QoS タイプ（Auto または Manual ）を表します。	自動
`maxThroughput`	許容される最大スループットを MiB/ 秒単位で設定します。手動 QoS 容量プールに対してのみサポートされます。	`4 MiB/sec`

ネットワーク機能の詳細については、"Azure NetApp Files ボリュームのネットワーク機能を設定する"を参照してください。

Azureクラウド環境について検討する（**26.02**）

26.02リリース以降、Tridentは複数のAzureクラウド環境でAzure NetApp Filesバックエンドの作成と管理をサポートします。

サポートされている Azure クラウドは以下のとおりです：

- Azure Commercial
- Azure Government（Azure Government / MAG）

Trident をデプロイするか、Azure NetApp Files バックエンドを作成する際は、Azure Resource Manager と認証エンドポイントが Azure クラウド環境と一致していることを確認してください。エンドポイントが一致しない場合、 `tridentctl`は認証できず、バックエンドの作成に失敗します。

必要な権限とリソース

PVC の作成時に「容量プールが見つかりません」というエラーが表示される場合は、アプリの登録に必要な権限とリソース （サブネット、仮想ネットワーク、容量プール） が関連付けられていない可能性があります。デバッグが有効になっている場合、Trident はバックエンドの作成時に検出された Azure リソースをログに記録します。適切なロールが使用されていることを確認してください。

`resourceGroups`、 `netappAccounts`、 `capacityPools`、 `virtualNetwork`、および `subnet`の値は、短い名前または完全修飾名を使用して指定できます。短い名前は同じ名前の複数のリソースと一致する可能性があるため、ほとんどの場合、完全修飾名が推奨されます。

vNetが Azure NetApp Files（ANF）ストレージアカウントとは異なるリソースグループに配置されている場合は、バックエンドのresourceGroupsリストを設定する際に仮想ネットワークのリソースグループを指定してください。

`resourceGroups`、 `netappAccounts`、および `capacityPools`の値は、検出されたリソースのセットをこのストレージバックエンドで使用可能なものに制限するフィルタであり、任意の組み合わせで指定できます。完全修飾名は次の形式に従います：

タイプ	フォーマット
リソース グループ	<resource group>
NetAppアカウント	<resource group>/<netapp account>
容量プール	<resource group>/<netapp account>/<capacity pool>
仮想ネットワーク	<resource group>/<virtual network>
サブネット	<resource group>/<virtual network>/<subnet>

ボリュームのプロビジョニング

構成ファイルの特別なセクションで次のオプションを指定することにより、デフォルトのボリュームのプロビジョニングを制御できます。詳細については、[構成例]を参照してください。

パラメータ	概要	デフォルト
exportRule	新規ボリュームのエクスポートルール。 exportRule は、CIDR表記による任意の組み合わせのIPv4アドレスまたはIPv4サブネットをカンマで区切ったリストである必要があります。SMBボリュームでは無視されます。	"0.0.0.0/0"
snapshotDir	`.snapshot`ディレクトリへのアクセス	true、 false （明示的に設定）。
size	新しいボリュームのデフォルトサイズ	「100G」
unixPermissions	新規ボリュームのUnixパーミッション（8進数4桁）。SMBボリュームでは無視されます。	「」（プレビュー機能、サブスクリプションでホワイトリストへの登録が必要）

構成例

次の例は、ほとんどのパラメータをデフォルトのままにする基本構成を示しています。これはバックエンドを定義する最も簡単な方法です。

最小限の構成

これは絶対的に最小限のバックエンド構成です。この構成では、Tridentは、構成された場所にあるAzure NetApp Filesに委任されたすべてのNetAppアカウント、容量プール、サブネットを検出し、それらのプールとサブネットの1つに新しいボリュームをランダムに配置します。 `nasType`が省略されているため、 `nfs`デフォルトが適用され、バックエンドはNFSボリュームをプロビジョニングします。

この構成は、Azure NetApp Filesを使い始めたばかりで、いろいろ試しているときに最適ですが、実際には、プロビジョニングするボリュームに追加のスコープを設定する必要があります。

```
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
  tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
  clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
  clientSecret: SECRET
  location: eastus
```

AKS のマネージド ID

このバックエンド構成では、 `subscriptionID`、 `tenantID`、 `clientID`、および `clientSecret`が省略されています。これらはマネージド ID を使用する場合はオプションです。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - resource-group-1/netapp-account-1/ultra-pool
  resourceGroups:
    - resource-group-1
  netappAccounts:
    - resource-group-1/netapp-account-1
  virtualNetwork: resource-group-1/eastus-prod-vnet
  subnet: resource-group-1/eastus-prod-vnet/eastus-anf-subnet
```

AKS のクラウド ID

このバックエンド構成では、 `tenantID`、 `clientID`、および `clientSecret`が省略されていますが、これらはクラウド ID を使用する場合はオプションです。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
  location: eastus
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
```

容量プールフィルタを使用した特定のサービスレベル設定

このバックエンド構成では、Azureの `eastus`ロケーションの `Ultra`容量プールにボリュームを配置します。Tridentは、そのロケーションでAzure NetApp Filesに委任されたすべてのサブネットを自動的に検出し、そのうちの1つにランダムに新しいボリュームを配置します。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
```

手動 **QoS** 容量プールを使用したバックエンドの例

このバックエンド構成では、ボリュームを Azure の `eastus`ロケーションの手動 QoS 容量プールに配置します。

```
---
version: 1
storageDriverName: azure-netapp-files
backendName: anf1
location: eastus
labels:
  clusterName: test-cluster-1
  cloud: anf
  nasType: nfs
defaults:
  qosType: Manual
storage:
  - serviceLevel: Ultra
    labels:
      performance: gold
    defaults:
      maxThroughput: 10
  - serviceLevel: Premium
    labels:
      performance: silver
    defaults:
      maxThroughput: 5
  - serviceLevel: Standard
    labels:
      performance: bronze
    defaults:
      maxThroughput: 3
```

高度な設定

このバックエンド構成により、ボリュームの配置範囲が単一のサブネットにさらに縮小され、一部のボリュームプロビジョニングのデフォルトも変更されます。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
virtualNetwork: application-group-1/eastus-prod-vnet
subnet: application-group-1/eastus-prod-vnet/my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: "true"
  size: 200Gi
  unixPermissions: "0777"
```

仮想プールの構成

このバックエンド構成では、単一のファイルで複数のストレージプールを定義します。これは、異なるサービスレベルをサポートする複数の容量プールがあり、それらを表すストレージクラスを Kubernetes で作成する場合に便利です。仮想プールラベルは `performance`に基づいてプールを区別するために使用されました。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
resourceGroups:
  - application-group-1
networkFeatures: Basic
nfsMountOptions: vers=3,proto=tcp,timeo=600
labels:
  cloud: azure
storage:
  - labels:
      performance: gold
    serviceLevel: Ultra
    capacityPools:
      - application-group-1/netapp-account-1/ultra-1
      - application-group-1/netapp-account-1/ultra-2
    networkFeatures: Standard
  - labels:
      performance: silver
    serviceLevel: Premium
    capacityPools:
      - application-group-1/netapp-account-1/premium-1
  - labels:
      performance: bronze
    serviceLevel: Standard
    capacityPools:
      - application-group-1/netapp-account-1/standard-1
      - application-group-1/netapp-account-1/standard-2
```

サポートされているトポロジ構成

Tridentは、リージョンとアベイラビリティゾーンに基づいてワークロードのボリュームのプロビジョニングを容易にします。このバックエンド構成の `supportedTopologies`ブロックは、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。ここで指定するリージョンとゾーンの値は、各Kubernetesクラスターノードのラベルのリージョンとゾーンの値と一致する必要があります。これらのリージョンとゾーンは、ストレージクラスで提供できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むストレージクラスの場合、Tridentは指定されたリージョンとゾーンにボリュームを作成します。詳細については、"CSI トポロジを使用する"を参照してください。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
supportedTopologies:
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-1
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-2
```

ストレージクラスの定義

次の `StorageClass`定義は、上記のストレージプールを参照します。

`parameter.selector`フィールドを使用した定義例

`parameter.selector`を使用すると、
`StorageClass`ごとに、ボリュームをホストするために使用される仮想プールを指定できます。
ボリュームには、選択したプールで定義された側面が含まれます。

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: silver
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: bronze
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze
allowVolumeExpansion: true
```

SMB ボリュームの定義例

```
`nasType`、 `node-stage-secret-name`、および `node-stage-secret-
namespace`を使用して、SMBボリュームを指定し、必要なActive
Directory資格情報を提供できます。
```

デフォルトの名前空間での基本設定

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

名前空間ごとに異なるシークレットを使用する

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

ボリュームごとに異なるシークレットを使用する

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` SMB ボリュームをサポートするプールのフィルタ。
`nasType: nfs`または `nasType: null` NFS プールのフィルタ。

バックエンドを作成する

バックエンド構成ファイルを作成したら、次のコマンドを実行します：

```
tridentctl create backend -f <backend-file>
```

非商用 Azure クラウドを使用する場合は、 `tridentctl` が Azure クラウド環境の Azure Resource Manager および認証エンドポイントを使用するように構成されていることを確認してください。バックエンドの作成が失敗した場合は、バックエンドの設定を確認し、ログを表示して原因を特定してください。

```
tridentctl logs
```

構成ファイルの問題を特定して修正したら、create コマンドを再度実行できます。

Google Cloud NetApp Volumes

Google Cloud NetApp Volumes を設定

Google Cloud NetApp Volumes を Trident のバックエンドとして設定し、Kubernetes ワークロード用のストレージをプロビジョニングできます。

概要

Trident は、NAS（NFS および SMB）とブロック（iSCSI）ワークロードの両方で Google Cloud NetApp Volumes をサポートしています。

- NASワークロードは `google-cloud-netapp-volumes`バックエンドを使用します
- ブロック（iSCSI）ワークロードは `google-cloud-netapp-volumes-san`バックエンドを使用します

NASボリュームはファイルベースのストレージを提供し、NFSまたはSMBプロトコルを使用してアクセスされます。これらのボリュームは、複数のポッドまたはノード間での共有アクセスをサポートします。

ブロックボリュームは生のブロックストレージを提供し、Kubernetesノードに接続されたiSCSIデバイスとしてアクセスされます。これらのボリュームは、アプリケーションがブロックレベルのアクセスを必要とする場合に使用されます。

これは以下の環境に適用されます：

- Trident 26.02以降
- Google Kubernetes Engine（GKE）または Red Hat OpenShift
- Google Cloud NetApp Volumes ストレージプール

ブロック（iSCSI）ストレージを構成するには、"ブロックストレージ（iSCSI）の設定"を参照してください。

Cloud IDを使用すると、Kubernetesワークロードは静的な認証情報を使用する代わりに、ワークロードIDとして認証することでGoogle Cloudリソースにアクセスできるようになります。

Google Cloud NetApp Volumes でクラウド ID を使用するには、以下が必要です：

- Google Kubernetes Engine (GKE) を使用してデプロイされた Kubernetes クラスター
- GKEクラスタでワークロードIDが有効になっており、ノードプールでメタデータサーバーが有効になっています。
- Google Cloud NetApp Volumes 管理者ロール（(`roles/netapp.admin`）または同等のカスタムロールを持つ Google Cloud サービスアカウント
- クラウド プロバイダが `GCP`に設定され、クラウドID注釈が構成された状態でTridentがインストールされています

Trident オペレータ

Trident オペレータを使用して Trident をインストールするには、 `tridentorchestrator_cr.yaml`を編集します：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  cloudProvider: "GCP"
  cloudIdentity: "iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com"
```

Helm

Helmを使用してTridentをインストールする際に、クラウド プロバイダとクラウドIDを設定します：

```
helm install trident trident-operator-100.6.0.tgz \
  --set cloudProvider=GCP \
  --set cloudIdentity="iam.gke.io/gcp-service-account: cloudvolumes-
admin-sa@mygcpproject.iam.gserviceaccount.com"
```

tridentctl

クラウド プロバイダとクラウドIDを指定してTridentをインストールします：

```
tridentctl install \
  --cloud-provider=GCP \
  --cloud-identity="iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com" \
  -n trident
```

NASストレージの設定

Google Cloud NetApp Volumes UNIFIEDストレージプールの場合、Tridentはボリューム操作中にUNIFIED固有のネーミングおよび検証ルールを適用します。

ボリュームを検索する際、Tridentは複数の互換性のあるボリューム名のバリアント（ハイフン形式やアンダースコア形式など）を評価することで、インポートと検出の信頼性を向上させることができます。

ドライバの詳細

Tridentは、Google Cloud NetApp VolumesからNASストレージをプロビジョニングするための `google-cloud-

netapp-volumes`ドライバーを提供します。

このドライバは、以下のアクセスモードをサポートしています。

- ReadWriteOnce（RWO）
- ReadOnlyMany（ROX）
- ReadWriteMany（RWX）
- ReadWriteOncePod（RWOP）

Driver	プロトコル	**volumeMode**	サポートされているアクセスモード	サポートされているファイルシステム
`google-cloud-netapp-volumes`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	`nfs, smb`

Trident NASバックエンドを設定する

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
    network: "<vpc-network>"
```

NAS ボリュームのプロビジョニング

NASボリュームは、 `google-cloud-netapp-volumes`バックエンドを使用してプロビジョニングされ、NFSおよびSMBプロトコルをサポートします。

StorageClass（**NFS**ボリューム用）

NFSボリュームをプロビジョニングするには、 `nasType`を `nfs`に設定します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "nfs"
allowVolumeExpansion: true
```

StorageClass（SMBボリューム用）

SMB ボリュームをプロビジョニングするには、 `nasType`を `smb`に設定してクレデンシャルを指定します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
allowVolumeExpansion: true
```

PersistentVolumeClaim の例（RWX）

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwx
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

PersistentVolumeClaim の例（RWO）

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

NASボリュームは `volumeMode: Filesystem`を使用します。

Google Cloud NetApp Volumes の SAN ワークロード向け設定

iSCSI プロトコルを使用して Google Cloud NetApp Volumes からブロックストレージボリュームをプロビジョニングするように Trident を設定できます。SAN ボリュームは、`google-cloud-netapp-volumes-san`ストレージドライバを使用して Flex Unified ストレージプールからプロビジョニングされます。

このドライバはブロックワークロード専用であり、NASプロトコルはサポートしていません。

`google-cloud-netapp-volumes-san`バックエンドは、iSCSIブロックボリュームのプロビジョニングに必要です。 `google-cloud-netapp-volumes`バックエンドはNASプロトコルのみをサポートしており、SANワークロードには使用できません。

概要

Trident は、 `google-cloud-netapp-volumes-san`ドライバを使用して、Google Cloud NetApp Volumes SAN（iSCSI）ワークロードをサポートしています。

SANボリュームはFlex Unifiedストレージプールからプロビジョニングされ、iSCSIブロックデバイスとしてKubernetesノードに提示されます。

これは以下の環境に適用されます：

- Trident 26.02以降
- Google Kubernetes Engine（GKE）または Red Hat OpenShift
- Google Cloud NetApp Volumes Flex統合ストレージプール
- iSCSIベースのワークロード

Flex Unified ストレージプール

Flex Unifiedストレージプールは、iSCSIプロトコルを使用してブロックストレージを提供し、SANプロビジョ

ニングに必要です：

- Flex Unified REGIONAL プールがサポートされています。
- Flex Unified ZONAL プールは、Trident 26.02.1 以降でサポートされています。
- SANワークロードでは、*Flex*サービスレベルのみがサポートされます。

Trident SANバックエンドの設定

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-san
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
      performance: flex
    network: "<vpc-network>"
    serviceLevel: Flex
```

StorageClass を作成する

SANバックエンドを設定したら、 `google-cloud-netapp-volumes-san`ドライバを参照するStorageClassを作成します。

ファイルシステムタイプは、バックエンドではなく StorageClass で定義されます。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

サポートされているファイルシステムの種類：

- `ext4`（デフォルト）
- `ext3`
- `xfs`

> SAN ドライバーは Flex サービス レベルのみをサポートし、 `exportRule`、 `unixPermissions`、 `nasType`、 `snapshotDir`、 `nfsMountOptions`、または階層化関連の設定などの NAS 固有のバックエンド パラメーターは使用しません。

ブロックボリュームのプロビジョニング

ReadWriteOnce（RWO）

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadWriteOncePod（RWOP）

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwop
spec:
  accessModes:
    - ReadWriteOncePod
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadOnlyMany（ROX）

ROXの一般的なパターンは、既存のReadWriteOnceボリュームをクローンし、そのクローンを読み取り専用としてマウントすることです。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rox
spec:
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
  dataSource:
    kind: PersistentVolumeClaim
    name: gcnv-san-rwo
```

ReadWriteMany（RWX）— 生ブロックのみ

ReadWriteMany は、 `volumeMode: Block`の場合にのみサポートされます。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-raw-rwx
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ブロックボリュームの動作

ブロックボリュームはiSCSI LUNとしてプロビジョニングされ、ブロックデバイスとしてKubernetesノードに提供されます。

ブロックボリューム：

- iSCSIプロトコルを使用する
- ファイルシステムと生ブロックの表示をサポート
- Tridentによってアタッチおよび管理される
- 複数のKubernetesアクセスモードをサポート

アクセスモード

Trident によってプロビジョニングされたブロックボリュームは、以下のアクセスモードをサポートします：

- `ReadWriteOnce` （RWO）
- `ReadOnlyMany`（ROX）
- `ReadWriteOncePod`（RWOP）
- `ReadWriteMany`（RWX）は、以下の場合にのみサポートされます。 `volumeMode: Block`

volumeMode の動作

`volumeMode`フィールドは、ブロックボリュームの公開方法を制御します。

- `Filesystem` Trident はボリュームをフォーマットしてマウントします。
- `Block` Trident はデバイスを接続し、rawブロックデバイスとして公開します。

サポートされている操作

`google-cloud-netapp-volumes-san`ドライバーを使用してプロビジョニングされたブロックボリュームがサポートする機能：

- 作成
- 削除
- クローン
- Snapshot
- サイズ変更
- インポート

余分な **GiB** のオーバープロビジョニング動作

Google Cloud NetApp Volumes ブロックボリュームには、内部メタデータのオーバーヘッドが含まれます。このオーバーヘッドにより、カーネルから見えるデバイスサイズは、プロビジョニングされた容量と比較して小さくなります。

テスト結果：

- 初回作成時に約300 KiBのオーバーヘッドが発生します。
- サイズ変更後、最大約 107 MiB のオーバーヘッドが発生します。

Google Cloud NetApp Volumes はGiB単位の割り当てのみを受け入れるため、Trident は以下の方法により、使用可能なデバイスサイズが常にPVCの要求を満たすか、それを超えることを保証します：

- 要求されたサイズを次の整数GiBに切り上げます

- 1 GiB のバッファを追加する

例：

- PVCリクエスト：100 GiB
- Google Cloud NetApp Volumes でプロビジョニングされたサイズ：101 GiB
- アプリケーションから見える使用可能な容量：少なくとも100 GiB

Podの例

ファイルシステムマウントされたブロックボリューム（**RWO**）

```
apiVersion: v1
kind: Pod
metadata:
  name: app-rwo
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeMounts:
    - name: data
      mountPath: /mnt/data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-rwo
```

生ブロックデバイス（**RWX**）

```
apiVersion: v1
kind: Pod
metadata:
  name: app-raw-rwx
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeDevices:
    - name: data
      devicePath: /dev/xda
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-raw-rwx
```

アタッチおよびマウント動作

Google Cloud NetApp Volumes からプロビジョニングされた SAN ボリュームの場合：

- Tridentは、Flex Unifiedストレージプール内に論理ユニット番号（LUN）を作成します。
- 公開中、Trident は LUN をノードごとのホストグループにマッピングします。
- ノードステージング中、Trident：
  - iSCSIターゲットにログインします
  - LUNを検出します
  - マルチパスを設定します
- `volumeMode: Filesystem`の場合、Tridentは必要に応じてデバイスをフォーマットし、マウントします。
- `volumeMode: Block`の場合、Tridentはデバイスを接続し、フォーマットやマウントを行わずに直接Podに公開します。

> SANブロックボリュームは、分散ロックや書き込み調整機能を提供しません。ブロックボリュームが複数のノードからアクセスされる場合（ReadWriteManyと `volumeMode: Block`）、アプリケーションまたはファイルシステムは、並行処理を管理する必要があります。

Google Cloud NetApp Volumes バックエンドを構成する準備をする

Google Cloud NetApp Volumes バックエンドを設定する前に、次の要件が満たされていることを確認する必要があります。

NFSまたは**SMB**ボリュームの前提条件

Google Cloud NetApp Volumes を初めて使用する場合、または新しい場所で使用する場合は、Google Cloud NetApp Volumes をセットアップして NFS または SMB ボリュームを作成するために、初期設定が必要で

す。"開始する前に"を参照してください。

Google Cloud NetApp Volumes バックエンドを構成する前に、次のものを用意してください。

- Google Cloud NetApp Volumes サービスが設定された Google Cloud アカウント。"Google Cloud NetApp Volumes"を参照してください。
- Google Cloud アカウントのプロジェクト番号。"プロジェクトの特定"を参照してください。
- NetApp Volumes Admin (`roles/netapp.admin`）ロールを持つGoogle Cloudサービスアカウント。"Identity and Access Managementのロールと権限"を参照してください。
- GCNV アカウントの API キー ファイル。"サービスアカウントキーを作成する"を参照してください。
- ストレージプール。"ストレージプールの概要"を参照してください。

Google Cloud NetApp Volumes へのアクセスを設定する方法の詳細については、"Google Cloud NetApp Volumes へのアクセスを設定する"を参照してください。

Google Cloud NetApp Volumes バックエンドの設定オプションと例

Google Cloud NetApp Volumes のバックエンド構成オプションについて学習し、構成例を確認します。

バックエンド構成オプション

各バックエンドは、単一の Google Cloud リージョンにボリュームをプロビジョニングします。他のリージョンにボリュームを作成するには、追加のバックエンドを定義できます。

パラメータ	概要	デフォルト
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	`storageDriverName`の値は「google-cloud-netapp-volumes」として指定する必要があります。
`backendName`	（オプション）ストレージバックエンドのカスタム名	ドライバー名 + "_" + API キーの一部
`storagePools`	ボリューム作成用のストレージプールを指定するために使用されるオプションのパラメータ。	
`projectNumber`	Google Cloud アカウントのプロジェクト番号。値は Google Cloud ポータルのホームページにあります。	
`location`	Google Cloud の所在地。Trident が GCNV ボリュームを作成します。リージョンをまたがる Kubernetes クラスタを作成する場合、 `location`で作成されたボリュームは、複数の Google Cloud リージョンにまたがるノードでスケジュールされたワークロードで使用できます。リージョン間のトラフィックには追加コストが発生します。	

パラメータ	概要	デフォルト
apiKey	netapp.admin`ロールを持つGoogle Cloudサービスアカウントの`API`キー。これには、`Google Cloud`サービスアカウントの秘密鍵ファイルの`JSON`形式の内容（バックエンド構成ファイルにそのままコピーされます）が含まれます。 `apiKey`には、次のキーのキーと値のペアを含める必要があります：`type、 project_id、 client_email、 client_id、 auth_uri、 token_uri、 auth_provider_x509_cert_url、および client_x509_cert_url。	
nfsMountOptions	NFSマウントオプションをきめ細かく制御できます。	"nfsvers=3"
limitVolumeSize	要求されたボリュームサイズがこの値を超える場合、プロビジョニングは失敗します。	""（デフォルトでは強制されません）
serviceLevel	ストレージ プールとそのボリュームのサービス レベル。値は `flex、 standard、 premium、または `extreme`です。	
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	""
network	GCNV ボリュームに使用される Google Cloud ネットワーク。	
debugTraceFlags	トラブルシューティング時に使用するデバッグ フラグ。例：`{"api":false, "method":true}`。トラブルシューティングを行っており、詳細なログダンプが必要な場合を除き、これを使用しないでください。	null
nasType	NFS または SMB ボリュームの作成を設定します。オプションは `nfs、 smb、またはnullです。null に設定すると、デフォルトで NFS ボリュームになります。	`nfs`
supportedTopologies	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、"CSI トポロジを使用する"を参照してください。例：`supportedTopologies: - topology.kubernetes.io/region: asia-east1 topology.kubernetes.io/zone: asia-east1-a`	

ボリュームのプロビジョニング オプション

デフォルトのボリュームプロビジョニングは、構成ファイルの `defaults`セクションで制御できます。

パラメータ	概要	デフォルト
exportRule	新しいボリュームのエクスポート ルール。任意の IPv4 アドレスの組み合わせをコンマで区切ったリストにする必要があります。	"0.0.0.0/0"

パラメータ	概要	デフォルト
`snapshotDir`	`.snapshot`ディレクトリへのアクセス	`true`、 `false` （デフォルトの動作は異なる場合があります。明示的に設定）NFSv3の場合は「false」
`snapshotReserve`	Snapshot 用に予約されているボリュームの割合	""（デフォルトの0を受け入れます）
`unixPermissions`	新しいボリュームの UNIX 権限（4 桁の 8 進数）。	""

構成例

次の例は、ほとんどのパラメータをデフォルトのままにする基本構成を示しています。これはバックエンドを定義する最も簡単な方法です。

最小限の構成

これは絶対的に最小限のバックエンド構成です。この構成では、Tridentは構成された場所でGoogle Cloud NetApp Volumesに委任されたすべてのストレージプールを検出し、新しいボリュームをそれらのプールの1つにランダムに配置します。 `nasType`が省略されているため、 `nfs`デフォルトが適用され、バックエンドはNFSボリュームをプロビジョニングします。

この構成は、Google Cloud NetApp Volumes を使い始めたばかりでいろいろ試している場合に最適ですが、実際にはプロビジョニングするボリュームに対して追加のスコープを設定する必要があるかもしれません。

`<id_value>`と `<key_value>`をサービスアカウントの認証情報に置き換えます。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

SMB ボリュームの設定

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv1
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123456789"
  location: asia-east1
  serviceLevel: flex
  nasType: smb
  apiKey:
    type: service_account
    project_id: cloud-native-data
    client_email: trident-sample@cloud-native-
data.iam.gserviceaccount.com
    client_id: "123456789737813416734"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/trident-
sample%40cloud-native-data.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

StoragePoolsフィルターを使用した構成

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  storagePools:
    - premium-pool1-europe-west6
    - premium-pool2-europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

仮想プールの構成

このバックエンド構成では、単一のファイルで複数の仮想プールを定義します。仮想プールは `storage` セクションで定義されます。異なるサービスレベルをサポートする複数のストレージプールがあり、Kubernetesでそれらを表すストレージクラスを作成する場合に役立ちます。仮想プールラベルは、プールを区別するために使用されます。たとえば、以下の例では `performance`ラベルと `serviceLevel` タイプが仮想プールを区別するために使用されます。

一部のデフォルト値をすべての仮想プールに適用できるように設定し、個々の仮想プールのデフォルト値を上書きすることもできます。次の例では、 `snapshotReserve`と `exportRule`がすべての仮想プールのデフォルトとして機能します。

詳細については、"仮想プール"を参照してください。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
```

```
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

GKE のクラウド ID

```
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
```

サポートされているトポロジ構成

Tridentは、リージョンとアベイラビリティゾーンに基づいてワークロードのボリュームのプロビジョニングを容易にします。このバックエンド構成の `supportedTopologies`ブロックは、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。ここで指定するリージョンとゾーンの値は、各Kubernetesクラスターノードのラベルのリージョンとゾーンの値と一致する必要があります。これらのリージョンとゾーンは、ストレージクラスで提供できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むストレージクラスの場合、Tridentは指定されたリージョンとゾーンにボリュームを作成します。詳細については、"CSI トポロジを使用する"を参照してください。

```
---
version: 1
storageDriverName: google-cloud-netapp-volumes
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: asia-east1
serviceLevel: flex
supportedTopologies:
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-a
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-b
```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します：

```
kubectl create -f <backend-file>
```

バックエンドが正常に作成されたことを確認するには、次のコマンドを実行します：

```
kubectl get tridentbackendconfig

NAME                BACKEND NAME        BACKEND UUID
PHASE   STATUS
backend-tbc-gcnv    backend-tbc-gcnv    b2fd1ff9-b234-477e-88fd-713913294f65
Bound   Success
```

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。 `kubectl get tridentbackendconfig <backend-name>`コマンドを使用してバックエンドを記述するか、次のコマンドを実行してログを表示し、原因を特定できます：

```
tridentctl logs
```

構成ファイルの問題を特定して修正したら、バックエンドを削除して、create コマンドを再度実行できます。

ストレージクラスの定義

以下は、上記のバックエンドを参照する基本的な `StorageClass`定義です。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
```

- `parameter.selector`フィールドを使用した定義例：*

  `parameter.selector`を使用すると、各`StorageClass`に対して、ボリュームをホストするために使用されるlink:../trident-concepts/virtual-storage-pool.html["仮想プール"]を指定できます。ボリュームには、選択したプールで定義された側面が含まれます。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: extreme-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=extreme
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: premium-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=standard
  backendType: google-cloud-netapp-volumes
```

ストレージクラスの詳細については、"ストレージクラスを作成する"を参照してください。

SMB ボリュームの定義例

`nasType`、 `node-stage-secret-name`、および `node-stage-secret-namespace`を使用して、SMBボリュームを指定し、必要なActive Directory資格情報を提供できます。任意の権限または権限のないActive Directoryユーザー/パスワードをノードステージシークレットに使用できます。

デフォルトの名前空間での基本設定

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

名前空間ごとに異なるシークレットを使用する

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

ボリュームごとに異なるシークレットを使用する

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` SMB ボリュームをサポートするプールのフィルタ。 `nasType: nfs`または `nasType: null` NFS プールのフィルタ。

PVC定義の例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: gcnv-nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs-sc
```

PVC がバインドされているかどうかを確認するには、次のコマンドを実行します：

```
kubectl get pvc gcnv-nfs-pvc

NAME           STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS AGE
gcnv-nfs-pvc   Bound    pvc-b00f2414-e229-40e6-9b16-ee03eb79a213   100Gi
RWX            gcnv-nfs-sc  1m
```

Google Cloud NetApp Volumes の自動階層化を設定する

自動階層化は、TridentバックエンドパラメータとPersistentVolumeClaimアノテーションを使用して、ボリュームのプロビジョニング時に設定されます。Tridentを使用して、Google Cloud NetApp Volumesの自動階層化を設定できます。

概要

自動階層化により、Trident は非アクティブなデータをパフォーマンス層から容量層に自動的に移動するボリュームをプロビジョニングできます。これにより、頻繁にアクセスされるデータのパフォーマンスを維持しながら、ストレージコストを削減できます。

Trident は、ボリューム作成時にのみ自動階層化設定を適用します。プロビジョニング後の変更は Trident 26.02 ではサポートされていません。

概念

自動階層化

自動階層化機能は、アクセスパターンに基づいて、アクセス頻度の低いデータをパフォーマンス階層から容量階層に移動します。データ転送は非同期で行われるため、即時ではありません。

階層化ポリシー

階層化ポリシーは、ボリュームに対して自動階層化を有効にするかどうかを決定します。

以下のポリシーがサポートされています：* `auto`：アクセスパターンに基づいて自動階層化を有効にします * `none`：自動階層化を無効にします

冷却日数

冷却日数とは、データ ブロックが階層化の対象となるために非アクティブ状態を維持する必要がある最小日数を指定します。冷却日数は、階層化ポリシーが `auto`に設定されている場合にのみ適用されます。

構成モデル

構成スコープ

自動階層化は複数のスコープで設定できます：

- ストレージプールの範囲 環境：プールからプロビジョニングされたすべてのボリュームに適用されます。
- ボリューム スコープ 単一ボリュームに適用されます（PersistentVolumeClaim アノテーション経由）。

Trident は、各設定が定義されている場所に基づいて、有効な構成を決定します。

設定の優先順位

同じ設定が複数のスコープで定義されている場合、Trident は以下の優先順位を適用します：

1. PersistentVolumeClaim アノテーション
2. Tridentバックエンド構成
3. ストレージプールのデフォルト設定

優先順位の高い設定は、優先順位の低い設定を上書きします。

Trident 26.02 でサポートされている機能

Trident 26.02は、Google Cloud NetApp Volumesの次の自動階層化機能をサポートしています：

- ボリュームプロビジョニング時の自動階層化の有効化または無効化
- Trident バックエンド構成での階層化ポリシーの定義
- PVC アノテーションを使用して、階層化ポリシーとボリュームごとの冷却日数を上書きする
- 自動階層化が有効になっているボリュームの冷却日を設定する

Trident 26.02 でサポートされていない機能

次の処理はサポートされていません。

- ボリューム作成後に自動階層化設定を変更する
- Kubernetesアップデートを使用して既存ボリュームの階層化ポリシーを変更する
- Tridentが管理するプロビジョニングワークフロー以外で自動階層化設定を適用する

バックエンド構成パラメータ

以下のパラメータは、Trident バックエンド設定で定義された場合の自動階層化の動作を制御します：

パラメータ	必須	概要
tieringPolicy	いいえ	ボリュームごとの階層化ポリシー (`auto` または `none`)
tieringMinimumCoolingDays	いいえ	データが階層化されるまでの非アクティブ日数（範囲：2〜183、デフォルト：31）

PersistentVolumeClaim アノテーションを使用したボリュームレベルのオーバーライド

サポートされているアノテーション

PersistentVolumeClaimアノテーションを使用すると、ボリュームごとに自動階層化設定を上書きできます。

注釈	概要
`trident.netapp.io/tieringPolicy`	ボリュームの階層化ポリシーを上書きします
`trident.netapp.io/tieringMinimumCoolingDays`	ボリュームの冷却日数の値を上書きします

例：自動階層化オーバーライド付き PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-tiering-pvc
  annotations:
    trident.netapp.io/tieringPolicy: auto
    trident.netapp.io/tieringMinimumCoolingDays: "45"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: google-cloud-netapp-volumes-auto-tiering
  resources:
    requests:
      storage: 500Gi
```

プロビジョニング動作

- 自動階層化設定は、ボリューム作成時にのみ評価および適用されます。
- Tridentは、プロビジョニング後に階層化の設定を調整しません。
- 階層化ポリシーが `none`に設定されている場合、冷却日数は無視されます。

プラットフォームの制限

- 自動階層化は、NASボリューム（NFSおよびSMB）でのみサポートされています。
- ブロックボリューム（iSCSI）は自動階層化をサポートしていません。
- Google Cloud NetApp Volumes のストレージプールでは、Google Cloud で自動階層化が有効になっている必要があります。

サポートされている値

- `tieringMinimumCoolingDays`の有効範囲：2～183
- デフォルト値：31

NetApp HCI または SolidFire バックエンドを設定する

Tridentインストールを使用してElementバックエンドを作成および使用する方法について説明します。

要素ドライバの詳細

Tridentは、 `solidfire-san`クラスタと通信するためのストレージドライバを提供します。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP) です。

`solidfire-san`ストレージドライバは、_file_および_block_ボリュームモードをサポートします。 `Filesystem` volumeModeの場合、Tridentはボリュームを作成し、ファイルシステムを作成します。ファイルシステムのタイプはStorageClassで指定されます。

Driver	プロトコル	VolumeMode	サポートされているアクセスモード	サポートされているファイルシステム
solidfire-san	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムがありません。Rawブロックデバイス。
solidfire-san	iSCSI	Filesystem	RWO、RWOP	xfs、 ext3、 ext4

開始する前に

Element バックエンドを作成する前に、次のものが必要です。

- Element ソフトウェアを実行するサポート対象のストレージ システム。
- NetApp HCI/SolidFireクラスター管理者またはボリュームを管理できるテナント ユーザーのクレデンシャル。
- すべての Kubernetes ワーカーノードに適切な iSCSI ツールがインストールされている必要があります。"ワーカーノードの準備情報"を参照してください。

バックエンド構成オプション

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	デフォルト
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	常に「solidfire-san」
`backendName`	カスタム名またはストレージバックエンド	「solidfire_」+ ストレージ（iSCSI）IP アドレス
`Endpoint`	テナント資格情報を持つSolidFireクラスターのMVIP	
`SVIP`	ストレージ（iSCSI）IP アドレスとポート	
`labels`	ボリュームに適用する任意のJSON 形式のラベルのセット。	""
`TenantName`	使用するテナント名（見つからない場合は作成されます）	
`InitiatorIFace`	iSCSIトラフィックを特定のホストインターフェイスに制限する	"default"
`UseCHAP`	CHAP を使用して iSCSI を認証します。Trident は CHAP を使用します。	true
`AccessGroups`	使用するアクセスグループIDのリスト	「trident」という名前のアクセスグループのIDを検索します
`Types`	QoS仕様	
`limitVolumeSize`	要求されたボリュームサイズがこの値を超える場合、プロビジョニングに失敗します	""（デフォルトでは強制されません）
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例：{"api":false, "method":true}	null

警告 `debugTraceFlags`は、トラブルシューティングを行っており、詳細なログダンプが必要な場合を除き、使用しないでください。

例**1**：**3**種類のボリュームタイプを持つ `**solidfire-san**`ドライバーのバックエンド構成

この例では、CHAP 認証を使用し、特定の QoS 保証を備えた 3 つのボリューム タイプをモデル化するバックエンド ファイルを示します。おそらく、 `IOPS` storage class パラメータを使用して、これらのそれぞれを消費するストレージクラスを定義することになるでしょう。

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
labels:
  k8scluster: dev1
  backend: dev1-element-cluster
UseCHAP: true
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
  - Type: Silver
    Qos:
      minIOPS: 4000
      maxIOPS: 6000
      burstIOPS: 8000
  - Type: Gold
    Qos:
      minIOPS: 6000
      maxIOPS: 8000
      burstIOPS: 10000
```

例**2**： `**solidfire-san**`ドライバーと仮想プールを使用したバックエンドおよびストレージクラスの設定

この例では、仮想プールが設定されたバックエンド定義ファイルと、それらを参照するStorageClassesを示しています。

Tridentは、プロビジョニング時にストレージプールに存在するラベルをバックエンドストレージLUNにコピーします。便宜上、ストレージ管理者は仮想プールごとにラベルを定義し、ラベルごとにボリュームをグループ化できます。

以下に示すサンプルのバックエンド定義ファイルでは、すべてのストレージプールに特定のデフォルトが設定されており、 `type`がSilverに設定されています。仮想プールは `storage`セクションで定義されています。この例では、一部のストレージプールは独自のタイプを設定し、一部のプールは上記で設定されたデフォルト値を上書きします。

```
```

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
UseCHAP: true
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
  - Type: Silver
    Qos:
      minIOPS: 4000
      maxIOPS: 6000
      burstIOPS: 8000
  - Type: Gold
    Qos:
      minIOPS: 6000
      maxIOPS: 8000
      burstIOPS: 10000
type: Silver
labels:
  store: solidfire
  k8scluster: dev-1-cluster
region: us-east-1
storage:
  - labels:
      performance: gold
      cost: "4"
    zone: us-east-1a
    type: Gold
  - labels:
      performance: silver
      cost: "3"
    zone: us-east-1b
    type: Silver
  - labels:
      performance: bronze
      cost: "2"
    zone: us-east-1c
    type: Bronze
  - labels:
      performance: silver
```

```
      cost: "1"
    zone: us-east-1d
```

次のStorageClass定義は上記の仮想プールを参照します。 `parameters.selector`フィールドを使用して、各StorageClassはボリュームをホストするために使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プールで定義された側面が設定されます。

最初のStorageClass (`solidfire-gold-four`）は最初の仮想プールにマップされます。これはGoldの`Volume Type QoS`でゴールドパフォーマンスを提供する唯一のプールです。最後の`StorageClass (`solidfire-silver`）は、シルバーパフォーマンスを提供するストレージプールを呼び出します。Tridentは、どの仮想プールが選択されるかを決定し、ストレージ要件が満たされていることを確認します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold-four
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold; cost=4
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-three
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=3
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze; cost=2
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
```

```
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=1
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
  fsType: ext4
```

詳細情報の参照

- "ボリュームアクセスグループ"

ONTAP SANドライバー

ONTAP SAN ドライバの概要

ONTAP および Cloud Volumes ONTAP SAN ドライバーを使用した ONTAP バックエンドの設定方法について説明します。

ONTAP SAN ドライバーの詳細

Tridentは、ONTAPクラスタと通信するために次のSANストレージドライバを提供します。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP) です。

Driver	プロトコル	volumeMode	サポートされているアクセスモード	サポートされているファイルシステム
`ontap-san`	iSCSI SCSI over FC	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし；rawブロックデバイス
`ontap-san`	iSCSI SCSI over FC	Filesystem	RWO、RWOP ROX と RWX は Filesystem ボリューム モードでは使用できません。	`xfs`、 `ext3`、 `ext4`

Driver	プロトコル	volumeMode	サポートされているアクセスモード	サポートされているファイルシステム
`ontap-san`	NVMe/TCP NVMe/TCPに関する追加の考慮事項を参照してください。	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし；rawブロックデバイス
`ontap-san`	NVMe/TCP NVMe/TCPに関する追加の考慮事項を参照してください。	Filesystem	RWO、RWOP ROX と RWX は Filesystem ボリューム モードでは使用できません。	`xfs`、 `ext3`、 `ext4`
`ontap-san-economy`	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし；rawブロックデバイス
`ontap-san-economy`	iSCSI	Filesystem	RWO、RWOP ROX と RWX は Filesystem ボリューム モードでは使用できません。	`xfs`、 `ext3`、 `ext4`

警告

- `ontap-san-economy`を使用するのは、永続ボリュームの使用数が"サポートされているONTAPボリューム制限"を超えることが予想される場合のみです。
- `ontap-nas-economy`を使用するのは、永続ボリュームの使用数が"サポートされているONTAPボリューム制限"を超えることが予想され、かつ `ontap-san-economy`ドライバーを使用できない場合のみです。
- データ保護、ディザスタリカバリ、モビリティの必要性が予想される場合は、使用しないでください `ontap-nas-economy`。
- NetAppでは、ontap-san以外のすべてのONTAPドライバーでFlexvolの自動拡張を使用することは推奨されません。回避策として、Tridentはスナップショット リザーブの使用をサポートし、それに応じてFlexvolボリュームを拡張します。

ユーザー権限

Tridentは、ONTAPまたはSVM管理者として実行されることが想定されており、通常は `admin`クラスタユーザーまたは `vsadmin`SVMユーザー、または同じロールを持つ別の名前のユーザーを使用します。Amazon FSx for NetApp ONTAP環境では、TridentはONTAPまたはSVM管理者として実行されることが想定されており、クラスタ `fsxadmin`ユーザーまたは `vsadmin`SVMユーザー、または同じロールを持つ別の名前のユーザーを使用します。 `fsxadmin`ユーザーは、クラスタ管理者ユーザーの限定的な代替です。

メモ	`limitAggregateUsage`パラメータを使用する場合は、クラスタ管理者の権限が必要です。Amazon FSx for NetApp ONTAPをTridentで使用する場合、 `limitAggregateUsage`パラメータは `vsadmin`および `fsxadmin`ユーザアカウントでは機能しません。このパラメータを指定すると、設定処理は失敗します。

ONTAP 内でより制限的なロールを作成し、Trident ドライバーで使用することは可能ですが、推奨しません。Trident のほとんどの新しいリリースでは、考慮する必要がある追加の API が呼び出されるため、アップグレードが困難になり、エラーが発生しやすくなります。

NVMe/TCPに関する追加の考慮事項

Tridentは、 `ontap-san`ドライバーを使用して不揮発性メモリエクスプレス（NVMe）プロトコルをサポートします。これには次のものが含まれます：

- IPv6を使用したチャンク アップロード署名要求がサポートされるようになりました。
- NVMe ボリュームの Snapshot とクローン
- NVMe ボリュームのサイズ変更
- Tridentの外部で作成されたNVMeボリュームをインポートして、そのライフサイクルをTridentで管理できるようにする
- NVMe ネイティブマルチパス
- K8sノードの正常または異常シャットダウン（24.06）

Tridentでサポートされていない機能：

- NVMeでネイティブにサポートされているDH-HMAC-CHAP
- デバイスマッパー（DM）マルチパス
- LUKS暗号化

メモ	NVMeはONTAP REST APIでのみサポートされ、ONTAPI（ZAPI）ではサポートされていません。

ONTAP SANドライバを使用してバックエンドを設定する準備をします

ONTAP SAN ドライバーを使用した ONTAP バックエンドの設定要件と認証オプションについて理解します。

要件

すべての ONTAP バックエンドで、Trident では少なくとも 1 つのアグリゲートを SVM に割り当てる必要があります。

メモ	"ASA r2システム"は、ストレージ レイヤの実装において、他のONTAPシステム（ASA、AFF、FAS）とは異なります。ASA r2システムでは、アグリゲートの代わりにストレージの可用性ゾーンが使用されます。ASA r2システムでSVMにアグリゲートを割り当てる方法については、"事項を"ナレッジベースの記事を参照してください。

複数のドライバを同時に実行し、それぞれに対応するストレージクラスを作成できることも覚えておいてくだ

さい。たとえば、 `san-dev`ドライバを使用する `ontap-san`クラスと、 `san-default`ドライバを使用する `ontap-san-economy`クラスを設定することができます。

すべての Kubernetes ワーカー ノードに適切な iSCSI ツールがインストールされている必要があります。詳細については、"ワーカーノードを準備する"を参照してください。

ONTAP バックエンドを認証します

Trident では、 ONTAP バックエンドを認証する 2 つのモードが用意されています。

- 認証情報ベース：必要な権限を持つONTAPユーザーのユーザー名とパスワード。 `admin`または `vsadmin`などの事前定義されたセキュリティログインロールを使用して、ONTAPバージョンとの最大限の互換性を確保することをお勧めします。
- 証明書ベース：Tridentは、バックエンドにインストールされた証明書を使用してONTAPクラスタと通信することもできます。ここで、バックエンド定義には、クライアント証明書、キー、および信頼されたCA証明書（使用する場合）のBase64エンコードされた値が含まれている必要があります（推奨）。

既存のバックエンドを更新して、資格情報ベースの方法と証明書ベースの方法を切り替えることができます。ただし、一度にサポートされる認証方法は 1 つだけです。別の認証方法に切り替えるには、バックエンド構成から既存の方法を削除する必要があります。

警告	*資格情報と証明書の両方*を提供しようとすると、構成ファイルに複数の認証方法が提供されているというエラーが発生し、バックエンドの作成が失敗します。

クレデンシャルベースの認証を有効にする

Trident が ONTAP バックエンドと通信するには、 SVM スコープ / クラスタスコープの管理者のクレデンシャルが必要です。 `admin`や `vsadmin`などの標準の事前定義されたロールを使用することを推奨します。これにより、将来の ONTAP リリースで公開される可能性のある機能 API を将来の Trident リリースで使用できるように、上位互換性が確保されます。カスタムセキュリティログインロールを作成して Trident で使用することもできますが、推奨されません。

サンプルのバックエンド定義は次のようになります：

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: password
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password"
}
```

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに留意してください。バックエンドが作成されると、ユーザ名/パスワードはBase64でエンコードされ、Kubernetesシークレットとして保存されます。バックエンドの作成または更新は、クレデンシャルに関する知識が必要となる唯一のステップです。したがって、これはKubernetes/ストレージ管理者によって実行される管理者専用の操作です。

証明書ベースの認証の有効化

新規および既存のバックエンドは証明書を使用して ONTAP バックエンドと通信できます。バックエンド定義には 3 つのパラメータが必要です。

- clientCertificate：クライアント証明書の Base64 エンコードされた値。
- clientPrivateKey：関連付けられた秘密キーの Base64 エンコードされた値。
- trustedCACertificate：信頼された CA 証明書の Base64 エンコードされた値。信頼できる CA を使用する場合は、このパラメータを指定する必要があります。信頼できる CA が使用されていない場合は、これを無視できます。

一般的なワークフローには次の手順が含まれます。

手順

1. クライアント証明書とキーを生成します。生成時に、Common Name（CN）を認証する ONTAP ユーザーに設定します。

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=admin"
```

2. 信頼できる CA 証明書を ONTAP クラスタに追加します。これはストレージ管理者によってすでに処理されている可能性があります。信頼できる CA が使用されていない場合は無視します。

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. クライアント証明書とキー（手順1から）をONTAPクラスタにインストールします。

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

メモ	このコマンドを実行すると、ONTAPは証明書の入力を求めます。ステップ1で生成された`k8senv.pem`ファイルの内容を貼り付け、`END`を入力してインストールを完了します。

4. ONTAP セキュリティログインロールが `cert`認証方法をサポートしていることを確認します。

```
security login create -user-or-group-name admin -application ontapi
-authentication-method cert
security login create -user-or-group-name admin -application http
-authentication-method cert
```

5. 生成された証明書を使用して認証をテストします。<ONTAP Management LIF>と<vserver name>を管理LIF IPとSVM名に置き換えます。

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. 証明書、キー、および信頼された CA 証明書を Base64 でエンコードします。

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 前の手順で取得した値を使用してバックエンドを作成します。

```
cat cert-backend.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"trustedCACertificate": "QNFinfO...SiqOyN",
"storagePrefix": "myPrefix_"
}

tridentctl create backend -f cert-backend.json -n trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       0 |
+------------+----------------+--------------------------------------
+--------+---------+
```

認証方法を更新するか、クレデンシャルをローテーションする

既存のバックエンドを更新して、別の認証方法を使用したり、資格情報をローテーションしたりすることができます。これは両方向に機能します。ユーザー名/パスワードを使用するバックエンドは証明書を使用するように更新できます。証明書を使用するバックエンドはユーザー名/パスワード ベースに更新できます。これを行うには、既存の認証方法を削除し、新しい認証方法を追加する必要があります。次に、必要なパラメータを含む更新されたbackend.jsonファイルを使用して `tridentctl backend update`を実行します。

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

メモ	パスワードをローテーションする場合、ストレージ管理者はまず ONTAP でユーザーのパスワードを更新する必要があります。続いてバックエンドの更新が行われます。証明書をローテーションする場合、ユーザーに複数の証明書を追加できます。バックエンドは新しい証明書を使用するように更新され、その後古い証明書は ONTAP クラスタから削除できます。

バックエンドを更新しても、すでに作成されているボリュームへのアクセスは中断されず、その後に行われたボリューム接続にも影響はありません。バックエンドのアップデートが成功したということは、TridentがONTAPバックエンドと通信でき、今後のボリューム操作を処理できることを示しています。

Trident 用のカスタム ONTAP ロールを作成します

最小限の権限を持つONTAPクラスタロールを作成することで、Tridentで操作を実行するためにONTAP管理者ロールを使用する必要がなくなります。Tridentバックエンド構成にユーザー名を含めると、Tridentは作成したONTAPクラスタロールを使用して操作を実行します。

Tridentカスタムロールの作成の詳細については、"Tridentカスタムロールジェネレーター"を参照してください。

ONTAPコマンドラインの使用

1. 次のコマンドを使用して新しいロールを作成します：

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. Tridentユーザーのユーザー名を作成します：

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> -vserver
<svm_name\> -comment "user_description"
```

3. ロールをユーザーにマップします：

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

System Managerを使用

ONTAP System Managerで次の手順を実行します。

1. カスタムロールを作成する：
   a. クラスタレベルでカスタムロールを作成するには、* Cluster > Settings *を選択します。

      （または）SVMレベルでカスタムロールを作成するには、*ストレージ > ストレージVM > `required SVM`> 設定 > ユーザーとロール*を選択します。

   b. ユーザーとロール*の横にある矢印アイコン（→*）を選択します。
   c. **Roles***の下の+Add*を選択します。
   d. ロールのルールを定義し、*保存*をクリックします。
2. **Trident**ユーザーに役割をマッピングする：+*ユーザーとロール*ページで次の手順を実行します：
   a. ユーザー*の下にある追加アイコン+*を選択します。
   b. 必要なユーザー名を選択し、*Role*のドロップダウンメニューで役割を選択します。
   c. *保存*をクリックします。

詳細については、次のページを参照してください：

- "ONTAPの管理用のカスタムロール" または "カスタム ロールの定義"
- "ロールとユーザーを操作する"

双方向**CHAP**による接続の認証

Tridentは、 `ontap-san`および `ontap-san-economy`ドライバで双方向CHAPを使用してiSCSIセッションを認証できます。これには、バックエンド定義で `useCHAP`オプションを有効にする必要があります。 `true`に設定すると、TridentはSVMのデフォルトのイニシエータセキュリティを双方向CHAPに設定し、バックエンドフ

ァイルからユーザー名とシークレットを設定します。NetAppは、接続の認証に双方向CHAPを使用することを推奨します。次のサンプル構成を参照してください：

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
```

警告 `useCHAP`パラメータは、一度だけ設定できるブーリアン パラメータです。デフォルトではfalseに設定されています。trueに設定した後は、falseに設定することはできません。

`useCHAP=true`に加えて、 `chapInitiatorSecret`、 `chapTargetInitiatorSecret`、 `chapTargetUsername`、および `chapUsername`フィールドはバックエンド定義に含める必要があります。バックエンドを作成した後、 `tridentctl update`を実行することでシークレットを変更できます。

仕組み

`useCHAP`をtrueに設定すると、ストレージ管理者はTridentにストレージバックエンドでCHAPを設定するように指示します。これには次のものが含まれます：

- SVM で CHAP を設定する：
  - SVMのデフォルトのイニシエータセキュリティタイプがなし（デフォルトで設定）であり、かつボリューム内に既存のLUNが存在しない場合は、Tridentはデフォルトのセキュリティタイプを `CHAP`に設定し、CHAPイニシエータとターゲットのユーザー名とシークレットの構成に進みます。
  - SVMにLUNが含まれている場合、TridentはSVMでCHAPを有効にしません。これにより、SVM上にすでに存在するLUNへのアクセスが制限されなくなります。
- CHAP イニシエーターとターゲットのユーザー名およびシークレットを構成します。これらのオプションは、バックエンド構成で指定する必要があります（上記を参照）。

バックエンドが作成されると、Tridentは対応する `tridentbackend`CRDを作成し、CHAPシークレットとユーザー名をKubernetesシークレットとして保存します。このバックエンドでTridentによって作成されたすべてのPVは、CHAP経由でマウントおよび接続されます。

資格情報をローテーションしてバックエンドを更新する

`backend.json`ファイルでCHAPパラメータを更新することで、CHAP認証情報を更新できます。これには、CHAPシークレットを更新し、 `tridentctl update`コマンドを使用してこれらの変更を反映する必要があります。

警告 バックエンドのCHAPシークレットを更新する場合は、 `tridentctl`を使用してバックエンドを更新する必要があります。ONTAP CLIまたはONTAP System Managerを使用してストレージクラスタの資格情報を更新しないでください。Tridentはこれらの変更を反映できません。

```
cat backend-san.json
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "ontap_san_chap",
    "managementLIF": "192.168.0.135",
    "svm": "ontap_iscsi_svm",
    "useCHAP": true,
    "username": "vsadmin",
    "password": "password",
    "chapInitiatorSecret": "cl9qxUpDaTeD",
    "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
}

./tridentctl update backend ontap_san_chap -f backend-san.json -n trident
+----------------+----------------+--------------------------------------
+--------+---------+
|   NAME         | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+----------------+----------------+--------------------------------------
+--------+---------+
| ontap_san_chap | ontap-san      | aa458f3b-ad2d-4378-8a33-1a472ffbeb5c |
online |       7 |
+----------------+----------------+--------------------------------------
+--------+---------+
```

既存の接続は影響を受けません。Trident が SVM で資格情報を更新しても、引き続きアクティブなままになります。新しい接続では更新された資格情報が使用され、既存の接続は引き続きアクティブなままになります。古い PV を切断して再接続すると、更新された資格情報が使用されるようになります。

ONTAP SAN 構成オプションと例

Trident インストールで ONTAP SAN ドライバを作成して使用する方法について説明しま

す。このセクションでは、バックエンドの設定例と、バックエンドを StorageClasses にマッピングするための詳細について説明します。"ASA r2システム"は、ストレージ レイヤの実装において、他のONTAPシステム（ASA、AFF、FAS）とは異なります。これらの違いは、記載されている特定のパラメータの使用に影響します。"ASA r2システムとその他のONTAPシステムの違いについて詳しくは、こちらをご覧ください"。Trident バックエンド構成では、システムが ASA r2 であることを指定する必要はありません。`ontap-san`を `storageDriverName`として選択すると、Trident は ASA r2 またはその他の ONTAP システムを自動的に検出します。以下の表に記載されているように、一部のバックエンド構成パラメータは ASA r2 システムには適用されません。

メモ	`ontap-san`ドライバー（iSCSI、NVMe/TCP、FCプロトコル）のみがASA r2システムでサポートされています。

バックエンド構成オプション

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	デフォルト
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	`ontap-san` または `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + dataLIF
`managementLIF`	クラスタまたは SVM 管理 LIF の IP アドレス。 完全修飾ドメイン名（FQDN）を指定できます。 Trident が IPv6 フラグを使用してインストールされている場合、IPv6 アドレスを使用するように設定できます。IPv6 アドレスは角括弧で定義する必要があります。例：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。 シームレスなMetroClusterスイッチオーバーについては、MetroCluster の例を参照してください。 メモ 「vsadmin」の資格情報を使用している場合は、 `managementLIF` SVMのものでなければなりません。「admin」の資格情報を使用する場合は、 `managementLIF` クラスターのものである必要があります。	"10.0.0.1"、"[2001:1234:abcd::fefe]"

パラメータ	概要	デフォルト
`dataLIF`	プロトコル LIF の IP アドレス。Trident が IPv6 フラグを使用してインストールされている場合、IPv6 アドレスを使用するように設定できます。IPv6 アドレスは角括弧で定義する必要があります。例：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。**iSCSI** の場合は指定しないでください。***Trident** は**"ONTAP セレクティブLUNマップ"**を使用して、マルチパス セッションを確立するために必要な **iSCSI LIF** を検出します。警告が発生するのは、 **`dataLIF`** が明示的に定義されている場合です。***MetroCluster** の場合は省略します。MetroCluster の例を参照してください。	SVMによって導出された
`svm`	使用するストレージ仮想マシン **MetroCluster** の場合は省略。MetroCluster の例を参照してください。	SVM `managementLIF`が指定されている場合に導出されます
`useCHAP`	ONTAP SAN ドライバーの iSCSI 認証に CHAP を使用します [ブール値]。 `true`に設定すると、バックエンドで指定された SVM のデフォルト認証として、Trident が双方向 CHAP を設定して使用します。詳細については、"ONTAP SANドライバを使用してバックエンドを設定する準備をします"を参照してください。**FCP** または **NVMe/TCP** ではサポートされません。	`false`
`chapInitiatorSecret`	CHAP イニシエータシークレット。次の場合は必須 `useCHAP=true`	""
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	""
`chapTargetInitiatorSecret`	CHAP ターゲット イニシエータ シークレット。次の場合は必須 `useCHAP=true`	""
`chapUsername`	受信ユーザー名。次の場合は必須 `useCHAP=true`	""
`chapTargetUsername`	ターゲットユーザー名。次の場合は必須 `useCHAP=true`	""
`clientCertificate`	クライアント証明書の Base64 エンコードされた値。証明書ベースの認証に使用	""
`clientPrivateKey`	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用	""
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコードされた値。任意。証明書ベースの認証に使用されます。	""
`username`	ONTAP クラスタとの通信に必要なユーザー名。クレデンシャルベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	""

パラメータ	概要	デフォルト
`password`	ONTAP クラスタとの通信に必要なパスワード。クレデンシャルベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	""
`svm`	使用するStorage Virtual Machine	SVM `managementLIF`が指定されている場合に導出されます
`storagePrefix`	SVM で新しいボリュームをプロビジョニングするときに使用されるプレフィックス。後で変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`
`aggregate`	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVM に割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバーの場合、このオプションは無視されます。割り当てられていない場合は、利用可能なアグリゲートのいずれかを使用してFlexGroupボリュームをプロビジョニングできます。 メモ: SVM でアグリゲートが更新されると、Trident Controller を再起動することなく、SVM をポーリングすることで Trident で自動的に更新されます。ボリュームをプロビジョニングするために Trident で特定のアグリゲートを設定している場合、そのアグリゲートの名前が変更されたり SVM から移動されたりすると、SVM アグリゲートのポーリング中にバックエンドが Trident で障害状態に移行します。バックエンドをオンラインに戻すには、アグリゲートを SVM 上に存在するものに変更するか、完全に削除する必要があります。 **ASA r2**システムには指定しないでください。	""
`limitAggregateUsage`	使用率がこのパーセンテージを超える場合、プロビジョニングは失敗します。Amazon FSx for NetApp ONTAP バックエンドを使用している場合は、`limitAggregateUsage`を指定しないでください。提供された `fsxadmin`と `vsadmin`には、Trident を使用してアグリゲートの使用状況を取得して制限するために必要な権限が含まれていません。**ASA r2**システムには指定しないでください。	""（デフォルトでは強制されません）
`limitVolumeSize`	要求されたボリューム サイズがこの値を超える場合、プロビジョニングは失敗します。また、LUNに対して管理するボリュームの最大サイズも制限します。	""（デフォルトでは強制されません）

パラメータ	概要	デフォルト
lunsPerFlexvol	FlexVolあたりの最大LUN数は[50, 200]の範囲でなければなりません	100
debugTraceFlags	トラブルシューティング時に使用するデバッグ フラグ。例：{"api":false, "method":true}トラブルシューティングを行っており、詳細なログ ダンプが必要な場合を除き、使用しないでください。	null
useREST	ONTAP REST APIを使用するためのブーリアン パラメータ。 `useREST`に設定すると `true`、TridentはONTAP REST APIを使用してバックエンドと通信します。`false`に設定すると、TridentはONTAPI（ZAPI）呼び出しを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、`ontapi`アプリケーションへのアクセス権が必要です。これは、事前定義された`vsadmin`および `cluster-admin`ロールで満たされます。Trident 24.06リリースおよびONTAP 9.15.1以降では、`useREST`はデフォルトで`true`に設定されます。ONTAPI（ZAPI）呼び出しを使用するには、 `useREST`を`false`に変更します。 `useREST`は、NVMe/TCP に完全対応しています。 メモ：NVMeはONTAP REST APIでのみサポートされ、ONTAPI（ZAPI）ではサポートされていません。 指定されている場合は、常に **ASA r2** システムの**`true`**に設定します。	ONTAP 9.15.1以降の場合は`true`、それ以外の場合は false。
sanType	iSCSIの場合は iscsi、NVMe/TCPの場合は nvme 、SCSI over Fibre Channel（FC）の場合は `fcp`を選択します。	iscsi 空白の場合

パラメータ	概要	デフォルト
formatOptions	`formatOptions`を使用して、`mkfs`コマンドのコマンドライン引数を指定します。これは、ボリュームがフォーマットされるたびに適用されます。これにより、設定に応じてボリュームをフォーマットできます。デバイス パスを除き、mkfsコマンドのオプションと同様にformatOptionsを指定してください。例："-E nodiscard" • `ontap-san`および`ontap-san-economy`ドライバでiSCSIプロトコルを使用する場合にサポートされます。*さらに、**iSCSI**および**NVMe/TCP**プロトコルを使用する場合、**ASA r2**システムでサポートされます。	
limitVolumePoolSize	ontap-san-economy バックエンドで LUN を使用する場合の最大リクエスト可能 FlexVol サイズ。	""（デフォルトでは強制されません）
denyNewVolumePools	バックエンドが LUN を格納する新しい FlexVol ボリュームを作成できないように制限 `ontap-san-economy`します。新しい PV のプロビジョニングには、既存の Flexvol のみが使用されます。	

formatOptionsの使用に関する推奨事項

Tridentは、フォーマット処理を高速化するために次のオプションを推奨します：

- **-E nodiscard (ext3、ext4):** mkfs 時にブロックを破棄しません（最初にブロックを破棄することは、ソリッド ステート デバイスやスパース / シンプロビジョニング ストレージでは有効です）。これは非推奨のオプション「-K」に代わるもので、ext3、ext4 ファイル システムに適用できます。
- **-K (xfs):** mkfs 時にブロックを破棄しません。このオプションは xfs ファイル システムに適用できます。

Active Directory の認証情報を使用してバックエンド **SVM** に **Trident** を認証

Tridentを設定して、Active Directory（AD）認証情報を使用してバックエンドSVMに認証できます。ADアカウントがSVMにアクセスする前に、クラスターまたはSVMへのADドメイン コントローラアクセスを設定する必要があります。ADアカウントを使用してクラスターを管理するには、ドメイン トンネルを作成する必要があります。詳細については、"ONTAPでActive Directoryドメイン コントローラ アクセスを設定する"を参照してください。

手順

1. バックエンド SVM のドメイン ネーム システム（DNS）設定を構成します：

```
vserver services dns create -vserver <svm_name> -dns-servers
<dns_server_ip1>,<dns_server_ip2>
```

2. 次のコマンドを実行して、Active Directory に SVM のコンピュータ アカウントを作成します：

```
vserver active-directory create -vserver DataSVM -account-name ADSERVER1
-domain demo.netapp.com
```

3. このコマンドを使用して、クラスタまたはSVMを管理するためのADユーザまたはグループを作成します

```
security login create -vserver <svm_name> -user-or-group-name
<ad_user_or_group> -application <application> -authentication-method domain
-role vsadmin
```

4. Tridentバックエンド設定ファイルで、 `username`および `password`パラメータをそれぞれADユーザー名またはグループ名とパスワードに設定します。

ボリュームのプロビジョニング用のバックエンド設定オプション

デフォルトのプロビジョニングは、設定の `defaults`セクションにあるこれらのオプションを使用して制御できます。例については、以下の設定例を参照してください。

パラメータ	概要	デフォルト
`spaceAllocation`	LUNのスペース割り当て	"true" 指定されている場合は、**ASA r2** システム用に **`true`**に設定します。
`spaceReserve`	スペース予約モード。「none」（シン）または「volume」（シック）。**ASA r2** システムの場合は `none`に設定します。	「なし」
`snapshotPolicy`	使用するSnapshotポリシー。**ASA r2**システムの場合は **`none`**に設定。	「なし」
`qosPolicy`	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージ プール/バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。Trident で QoS ポリシー グループを使用するには、ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各構成要素に個別に適用されるようにする必要があります。共有 QoS ポリシー グループは、すべてのワークロードの合計スループットの上限を適用します。	""
`adaptiveQosPolicy`	作成されたボリュームに割り当てるアダプティブ QoS ポリシー グループ。ストレージプール/バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください	""
`snapshotReserve`	スナップショット用に予約されているボリュームの割合。**ASA r2**システムには指定しないでください。	`snapshotPolicy`が「none」の場合は「0」、それ以外の場合は「」
`splitOnClone`	作成時にクローンを親から分離する	"false"

パラメータ	概要	デフォルト
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたボリュームはすべてNAEが有効になります。詳細については、次を参照してください："Tridentと NVE および NAE の連携"。	"false" 指定されている場合は、**ASA r2** システム **`true`**に設定します。
`luksEncryption`	LUKS暗号化を有効にします。"Linux Unified Key Setup（LUKS）を使用する"を参照してください。	"" **ASA r2** システムの場合は **`false`** に設定。
`tieringPolicy`	階層化ポリシーは「なし」を使用する **ASA r2** システムには指定しないでください。	
`nameTemplate`	カスタムボリューム名を作成するためのテンプレート。	""

ボリュームプロビジョニングの例

デフォルトを定義した例を次に示します：

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: <password>
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: standard
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

メモ

`ontap-san`ドライバを使用して作成されたすべてのボリュームについて、TridentはLUNメタデータに対応するためにFlexVolに10%の容量を追加します。LUNは、ユーザーがPVCで要求した正確なサイズでプロビジョニングされます。TridentはFlexVolに10%を追加します（ONTAPでは使用可能なサイズとして表示されます）。ユーザーは要求した使用可能な容量を取得できるようになりました。この変更により、使用可能なスペースが完全に使用されない限り、LUNが読み取り専用になることも防止されます。これはontap-san-economyには適用されません。

`snapshotReserve`を定義するバックエンドの場合、Tridentはボリュームのサイズを次のように計算します：

```
Total volume size = [(PVC requested size) / (1 - (snapshotReserve
percentage) / 100)] * 1.1
```

1.1は、TridentがLUNメタデータに対応するためにFlexVolに追加する10パーセントの追加分です。snapshotReserve = 5%、PVC要求 = 5 GiBの場合、ボリュームの合計サイズは5.79 GiB、使用可能なサイズは5.5 GiBになります。 `volume show`コマンドを実行すると、次の例のような結果が表示されます：

現在、既存のボリュームに対して新しい計算を使用する唯一の方法は、サイズ変更です。

最小限の構成例

次の例は、ほとんどのパラメータをデフォルトのままにする基本構成を示しています。これはバックエンドを定義する最も簡単な方法です。

メモ

Amazon FSx for NetApp ONTAP を Trident とともに使用している場合、NetApp では、IP アドレスではなく LIF の DNS 名を指定することを推奨しています。

ONTAP SANの例

これは、 `ontap-san`ドライバを使用した基本的な設定です。

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
username: vsadmin
password: <password>
```

MetroCluster の例

"SVMのレプリケーションとリカバリ"中のスイッチオーバーとスイッチバック後にバックエンド定義を手動で更新する必要がないように、バックエンドを設定できます。

シームレスなスイッチオーバーとスイッチバックを行うには、 `managementLIF`を使用してSVMを指定し、 `svm`パラメータは省略します。例：

```
version: 1
storageDriverName: ontap-san
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

ONTAP SANエコノミーの例

```
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
username: vsadmin
password: <password>
```

証明書ベースの認証の例

この基本構成例では、 `clientCertificate`、 `clientPrivateKey`、および `trustedCACertificate`（信頼できるCAを使用する場合はオプション）が `backend.json`に入力され、クライアント証明書、秘密キー、信頼できるCA証明書のbase64エンコードされた値をそれぞれ取得します。

```
---
version: 1
storageDriverName: ontap-san
backendName: DefaultSANBackend
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

双方向CHAPの例

これらの例では、 `useCHAP`を `true`に設定してバックエンドを作成します。

ONTAP SAN CHAPの例

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

ONTAP SAN economy CHAPの例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

NVMe/TCPの例

ONTAP バックエンドに NVMe で構成された SVM が必要です。これは、NVMe/TCP の基本的なバックエンド構成です。

```
---
version: 1
backendName: NVMeBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nvme
username: vsadmin
password: password
sanType: nvme
useREST: true
```

SCSI over FC（FCP）の例

ONTAP バックエンドで FC を使用して構成された SVM が必要です。これは FC の基本的なバックエンド構成です。

```
---
version: 1
backendName: fcp-backend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_fc
username: vsadmin
password: password
sanType: fcp
useREST: true
```

nameTemplateを使用したバックエンド構成の例

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap-san-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

formatOptions ontap-san-economy ドライバーの例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: ""
svm: svm1
username: ""
password: "!"
storagePrefix: whelk_
debugTraceFlags:
  method: true
  api: true
defaults:
  formatOptions: -E nodiscard
```

仮想プールを使用したバックエンドの例

これらのサンプルバックエンド定義ファイルでは、すべてのストレージプールに対して特定のデフォルトが設定されています。たとえば、 `spaceReserve`はnone、 `spaceAllocation`はfalse、 `encryption`はfalseです。仮想プールはストレージ セクションで定義されます。

Tridentは「コメント」フィールドにプロビジョニング ラベルを設定します。コメントはFlexVol volumeに設定されます。Tridentはプロビジョニング時に仮想プールに存在するすべてのラベルをストレージ ボリュームにコピーします。便宜上、ストレージ管理者は仮想プールごとにラベルを定義し、ラベルごとにボリュームを

グループ化できます。

これらの例では、一部のストレージプールは独自の `spaceReserve`、 `spaceAllocation`、および `encryption`値を設定し、一部のプールはデフォルト値を上書きします。

ONTAP SANの例

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "40000"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
      adaptiveQosPolicy: adaptive-extreme
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
      qosPolicy: premium
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

ONTAP SANエコノミーの例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
labels:
  store: san_economy_store
region: us_east_1
storage:
  - labels:
      app: oracledb
      cost: "30"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
  - labels:
      app: postgresdb
      cost: "20"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
  - labels:
      app: mysqldb
      cost: "10"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
  - labels:
      department: legal
      creditpoints: "5000"
```

```
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

NVMe/TCPの例

```
---
version: 1
storageDriverName: ontap-san
sanType: nvme
managementLIF: 10.0.0.1
svm: nvme_svm
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceAllocation: "false"
  encryption: "true"
storage:
  - labels:
      app: testApp
      cost: "20"
    defaults:
      spaceAllocation: "false"
      encryption: "false"
```

バックエンドを**StorageClasses**にマッピングする

次のStorageClass定義は[仮想プールを使用したバックエンドの例]を参照しています。 `parameters.selector` フィールドを使用して、各StorageClassはボリュームをホストするために使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プールで定義された側面が設定されます。

- `protection-gold` StorageClassは、 `ontap-san`バックエンドの最初の仮想プールにマッピングされます。これはゴールド レベルの保護を提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- `protection-not-gold`StorageClassは、 `ontap-san`バックエンドの2番目と3番目の仮想プールにマッピングされます。これらは、ゴールド以外の保護レベルを提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- この `app-mysqldb` StorageClass は `ontap-san-economy`バックエンドの3番目の仮想プールにマッピングされます。これは、mysqldb タイプのアプリにストレージ プール構成を提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- この `protection-silver-creditpoints-20k` StorageClassは `ontap-san`バックエンドの2番目の仮想プールにマッピングされます。これは、シルバーレベルの保護と20000クレジットポイントを提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- この `creditpoints-5k` StorageClassは、 `ontap-san`バックエンドの3番目の仮想プールと `ontap-san-economy`バックエンドの4番目の仮想プールにマッピングされます。これらは5000クレジットポイントで提供される唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

- この `my-test-app-sc` StorageClassは、 `testAPP`仮想プールに `ontap-san`ドライバで `sanType: nvme`マッピングされます。これは `testApp`を提供する唯一のプールです。

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: my-test-app-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=testApp"
  fsType: "ext4"
```

Trident は、どの仮想プールが選択されるかを決定し、ストレージ要件が満たされていることを確認します。

ONTAP NASドライバー

ONTAP NAS ドライバの概要

ONTAP および Cloud Volumes ONTAP NAS ドライバーを使用した ONTAP バックエンドの設定方法について説明します。

ONTAP NAS ドライバーの詳細

Tridentは、ONTAPクラスタと通信するために次のNASストレージドライバを提供します。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP) です。

Driver	プロトコル	volumeMode	サポートされているアクセスモード	サポートされているファイルシステム
`ontap-nas`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	"", `nfs`, `smb`
`ontap-nas-economy`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	"", `nfs`, `smb`
`ontap-nas-flexgroup`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	"", `nfs`, `smb`

警告

- `ontap-san-economy`を使用するのは、永続ボリュームの使用数が"サポートされているONTAPボリューム制限"を超えることが予想される場合のみです。
- `ontap-nas-economy`を使用するのは、永続ボリュームの使用数が"サポートされているONTAPボリューム制限"を超えることが予想され、かつ `ontap-san-economy`ドライバーを使用できない場合のみです。
- データ保護、ディザスタリカバリ、モビリティの必要性が予想される場合は、使用しないでください `ontap-nas-economy`。
- NetAppでは、ontap-san以外のすべてのONTAPドライバーでFlexvolの自動拡張を使用することは推奨されません。回避策として、Tridentはスナップショット リザーブの使用をサポートし、それに応じてFlexvolボリュームを拡張します。

ユーザー権限

Tridentは、ONTAPまたはSVM管理者として実行されることが想定されており、通常は `admin`クラスタユーザーまたは `vsadmin`SVMユーザー、または同じロールを持つ別の名前のユーザーを使用します。

Amazon FSx for NetApp ONTAP環境では、TridentはONTAPまたはSVM管理者として実行されることが想定されており、クラスタ `fsxadmin`ユーザーまたは `vsadmin`SVMユーザー、または同じロールを持つ別の名前のユーザーを使用します。 `fsxadmin`ユーザーは、クラスタ管理者ユーザーの限定的な代替です。

メモ

`limitAggregateUsage`パラメータを使用する場合は、クラスタ管理者の権限が必要です。Amazon FSx for NetApp ONTAPをTridentで使用する場合、 `limitAggregateUsage`パラメータは `vsadmin`および `fsxadmin`ユーザアカウントでは機能しません。このパラメータを指定すると、設定処理は失敗します。

ONTAP 内でより制限的なロールを作成し、Trident ドライバーで使用することは可能ですが、推奨しません。Trident のほとんどの新しいリリースでは、考慮する必要がある追加の API が呼び出されるため、アップグレードが困難になり、エラーが発生しやすくなります。

ONTAP NAS ドライバーを使用した ONTAP バックエンドの設定に関する要件、認証オプション、エクスポートポリシーを理解します。25.10リリース以降、NetApp Tridentは"NetApp AFX ストレージ システム"をサポートします。NetApp AFXストレージシステムは、ストレージ レイヤの実装において、他のONTAPシステム（ASA、AFF、FAS）とは異なります。Trident バックエンド構成では、システムが AFX であることを指定する必要はありません。 `ontap-nas`を `storageDriverName`として選択すると、Trident は AFX システムを自動的に検出します。

メモ	`ontap-nas`ドライバ（NFS プロトコル）のみが AFX システムでサポートされています。SMB プロトコルはサポートされていません。

要件

- すべての ONTAP バックエンドで、Trident では少なくとも 1 つのアグリゲートを SVM に割り当てる必要があります。
- 複数のドライバーを実行し、いずれかを指すストレージ クラスを作成できます。たとえば、 `ontap-nas`ドライバーを使用するGoldクラスと、 `ontap-nas-economy`を使用するBronzeクラスを設定できます。
- すべての Kubernetes ワーカーノードに適切な NFS ツールがインストールされている必要があります。詳細については、"ここをクリックしてください。"を参照してください。
- Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。詳細については、SMB ボリュームのプロビジョニングの準備を参照してください。

ONTAP バックエンドを認証します

Trident では、 ONTAP バックエンドを認証する 2 つのモードが用意されています。

- 資格情報ベース：このモードでは、ONTAPバックエンドに対する十分な権限が必要です。ONTAPバージョンとの最大限の互換性を確保するために、 `admin`や `vsadmin`などの事前定義されたセキュリティログインロールに関連付けられたアカウントを使用することをお勧めします。
- 証明書ベース：このモードでは、Trident が ONTAP クラスタと通信するために、バックエンドに証明書をインストールする必要があります。ここで、バックエンド定義には、クライアント証明書、キー、および信頼されたCA証明書（使用する場合）のBase64エンコードされた値が含まれている必要があります（推奨）。

既存のバックエンドを更新して、資格情報ベースの方法と証明書ベースの方法を切り替えることができます。ただし、一度にサポートされる認証方法は 1 つだけです。別の認証方法に切り替えるには、バックエンド構成から既存の方法を削除する必要があります。

警告	*資格情報と証明書の両方*を提供しようとすると、構成ファイルに複数の認証方法が提供されているというエラーが発生し、バックエンドの作成が失敗します。

クレデンシャルベースの認証を有効にする

Trident が ONTAP バックエンドと通信するには、 SVM スコープ / クラスタスコープの管理者のクレデンシャルが必要です。 `admin`や `vsadmin`などの標準の事前定義されたロールを使用することを推奨します。これにより、将来の ONTAP リリースで公開される可能性のある機能 API を将来の Trident リリースで使用できる

ように、上位互換性が確保されます。カスタムセキュリティログインロールを作成して Trident で使用することもできますが、推奨されません。

サンプルのバックエンド定義は次のようになります：

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
credentials:
  name: secret-backend-creds
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "credentials": {
        "name": "secret-backend-creds"
    }
}
```

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに留意してください。バックエンドが作成されると、ユーザ名/パスワードはBase64でエンコードされ、Kubernetesシークレットとして保存されます。バックエンドの作成/更新は、クレデンシャルに関する知識が必要となる唯一のステップです。したがって、これはKubernetes/ストレージ管理者によって実行される管理者専用の操作です。

証明書ベースの認証を有効にする

新規および既存のバックエンドは証明書を使用して ONTAP バックエンドと通信できます。バックエンド定義には 3 つのパラメータが必要です。

- clientCertificate：クライアント証明書の Base64 エンコードされた値。
- clientPrivateKey：関連付けられた秘密キーの Base64 エンコードされた値。
- trustedCACertificate：信頼された CA 証明書の Base64 エンコードされた値。信頼できる CA を使用する場合は、このパラメータを指定する必要があります。信頼できる CA が使用されていない場合は、これを無視できます。

一般的なワークフローには次の手順が含まれます。

手順

1. クライアント証明書とキーを生成します。生成時に、Common Name（CN）を認証する ONTAP ユーザーに設定します。

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=vsadmin"
```

2. 信頼できる CA 証明書を ONTAP クラスタに追加します。これはストレージ管理者によってすでに処理されている可能性があります。信頼できる CA が使用されていない場合は無視します。

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. クライアント証明書とキー（手順1から）をONTAPクラスタにインストールします。

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

4. ONTAP セキュリティログインロールが `cert`認証方法をサポートしていることを確認します。

```
security login create -user-or-group-name vsadmin -application ontapi
-authentication-method cert -vserver <vserver-name>
security login create -user-or-group-name vsadmin -application http
-authentication-method cert -vserver <vserver-name>
```

5. 生成された証明書を使用して認証をテストします。<ONTAP Management LIF>と<vserver name>を管理LIF IPとSVM名に置き換えます。LIF のサービスポリシーが `default-data-management`に設定されていることを確認する必要があります。

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. 証明書、キー、および信頼された CA 証明書を Base64 でエンコードします。

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 前の手順で取得した値を使用してバックエンドを作成します。

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

認証方法を更新するか、クレデンシャルをローテーションする

既存のバックエンドを更新して、別の認証方法を使用したり、資格情報をローテーションしたりすることができます。これは両方向に機能します。ユーザー名/パスワードを使用するバックエンドは証明書を使用するように更新できます。証明書を使用するバックエンドはユーザー名/パスワード ベースに更新できます。これを行うには、既存の認証方法を削除し、新しい認証方法を追加する必要があります。次に、必要なパラメータを含む更新されたbackend.jsonファイルを使用して `tridentctl update backend`を実行します。

```
cat cert-backend-updated.json
```

```
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}
```

```
#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

メモ	パスワードをローテーションする場合、ストレージ管理者はまず ONTAP でユーザーのパスワードを更新する必要があります。続いてバックエンドの更新が行われます。証明書をローテーションする場合、ユーザーに複数の証明書を追加できます。バックエンドは新しい証明書を使用するように更新され、その後古い証明書は ONTAP クラスタから削除できます。

バックエンドを更新しても、すでに作成されているボリュームへのアクセスは中断されず、その後に行われたボリューム接続にも影響はありません。バックエンドのアップデートが成功したということは、TridentがONTAPバックエンドと通信でき、今後のボリューム操作を処理できることを示しています。

Trident 用のカスタム **ONTAP** ロールを作成します

最小限の権限を持つONTAPクラスタロールを作成することで、Tridentで操作を実行するためにONTAP管理者ロールを使用する必要がなくなります。Tridentバックエンド構成にユーザー名を含めると、Tridentは作成したONTAPクラスタロールを使用して操作を実行します。

Tridentカスタムロールの作成の詳細については、"Tridentカスタムロールジェネレーター"を参照してください。

ONTAPコマンドラインの使用

1. 次のコマンドを使用して新しいロールを作成します：

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. Tridentユーザーのユーザー名を作成します：

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> -vserver
<svm_name\> -comment "user_description"
```

3. ロールをユーザーにマップします：

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

System Managerを使用

ONTAP System Managerで次の手順を実行します。

1. カスタムロールを作成する：
   a. クラスタレベルでカスタムロールを作成するには、* Cluster > Settings *を選択します。

      （または）SVMレベルでカスタムロールを作成するには、*ストレージ > ストレージVM > required SVM> 設定 > ユーザーとロール*を選択します。

   b. ユーザーとロール*の横にある矢印アイコン（→*）を選択します。
   c. **Roles***の下の+Add*を選択します。
   d. ロールのルールを定義し、*保存*をクリックします。
2. **Trident**ユーザーに役割をマッピングする：+*ユーザーとロール*ページで次の手順を実行します：
   a. ユーザー*の下にある追加アイコン+*を選択します。
   b. 必要なユーザー名を選択し、*Role*のドロップダウンメニューで役割を選択します。
   c. *保存*をクリックします。

詳細については、次のページを参照してください：

- "ONTAPの管理用のカスタムロール" または "カスタム ロールの定義"
- "ロールとユーザーを操作する"

NFS エクスポート ポリシーを管理する

Trident は NFS エクスポート ポリシーを使用して、プロビジョニングするボリュームへのアクセスを制御します。

Trident は、エクスポート ポリシーを操作するときに 2 つのオプションを提供します。

- Tridentは、エクスポート ルール自体を動的に管理できます。この動作モードでは、ストレージ管理者は、許容される IP アドレスを表す CIDR ブロックのリストを指定します。Tridentは、公開時に、これらの範囲内にある該当するノード IP をエクスポート ルールに自動的に追加します。あるいは、CIDR が指定されていない場合は、ボリュームが公開されるノードで見つかったすべてのグローバル スコープのユニキャスト IP がエクスポート ルールに追加されます。
- ストレージ管理者は、エクスポート ポリシーを作成し、ルールを手動で追加できます。Tridentは、構成で別のエクスポート ポリシー名が指定されていない限り、デフォルトのエクスポート ポリシーを使用します。

エクスポート ポリシーを動的に管理する

Tridentは、ONTAPバックエンドのエクスポート ポリシーを動的に管理する機能を提供します。これにより、ストレージ管理者は、明示的なルールを手動で定義するのではなく、ワーカーノードIPに許可されるアドレス空間を指定できるようになります。これにより、エクスポート ポリシーの管理が大幅に簡素化され、エクスポート ポリシーを変更する際にストレージ クラスタで手動で介入する必要がなくなります。さらに、これにより、ボリュームをマウントしており、指定された範囲内のIPを持つワーカー ノードのみにストレージ クラスタへのアクセスが制限され、きめ細かな自動管理がサポートされます。

メモ	動的エクスポート ポリシーを使用する場合は、ネットワーク アドレス変換（NAT）を使用しないでください。NAT では、ストレージ コントローラは実際の IP ホスト アドレスではなくフロントエンド NAT アドレスを認識するため、エクスポート ルールに一致するものが見つからない場合はアクセスが拒否されます。

例

使用する必要がある構成オプションが 2 つあります。バックエンドの定義の例を次に示します：

```
---
version: 1
storageDriverName: ontap-nas-economy
backendName: ontap_nas_auto_export
managementLIF: 192.168.0.135
svm: svm1
username: vsadmin
password: password
autoExportCIDRs:
  - 192.168.0.0/24
autoExportPolicy: true
```

メモ	この機能を使用する場合、SVMのルートジャンクションに、ノードCIDRブロックを許可するエクスポート ルールを含む、事前に作成されたエクスポートポリシー（たとえばデフォルトのエクスポートポリシー）があることを必ず確認してください。常に、NetAppが推奨するベストプラクティスに従い、Trident専用のSVMを用意してください。

上記の例を使用して、この機能がどのように機能するかを説明します：

- `autoExportPolicy`が `true`に設定されています。これは、Tridentがこのバックエンドでプロビジョニングされた各ボリュームの `svm1` SVM用のエクスポート ポリシーを作成し、`autoexportCIDRs`アドレス ブロックを使用してルールの追加と削除を処理することを示しています。ボリュームがノードに接続されるまで、そのボリュームへの不要なアクセスを防ぐために、ルールのない空のエクスポート ポリシーがボリュームで使用されます。ボリュームがノードに公開されると、Tridentは、指定されたCIDRブロック内のノードIPを含む基礎となるqtreeと同じ名前のエクスポート ポリシーを作成します。これらのIPは、親FlexVol volumeが使用するエクスポート ポリシーにも追加されます。
  - 次に例を示します。
    - バックエンド UUID 403b5326-8482-40db-96d0-d83fb3f4daec
    - `autoExportPolicy` に設定 `true`
    - ストレージ プレフィックス `trident`
    - PVC UUID a79bcf5f-7b6d-4a40-9876-e2551f159c1c
    - trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c という名前の qtree は、FlexVol という名前の `trident-403b5326-8482-40db96d0-d83fb3f4daec`のエクスポート ポリシー、qtree という名前の `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`のエクスポート ポリシー、および SVM 上の `trident_empty`という名前の空のエクスポート ポリシーを作成します。FlexVol エクスポート ポリシーのルールは、qtree エクスポート ポリシーに含まれるすべてのルールのスーパーセットになります。空のエクスポート ポリシーは、接続されていないボリュームによって再利用されます。
- `autoExportCIDRs`にはアドレス ブロックのリストが含まれます。このフィールドはオプションであり、デフォルトは ["0.0.0.0/0", "::/0"] になります。定義されていない場合、Tridentはパブリケーションを持つワーカー ノードで検出されたすべてのグローバル スコープのユニキャスト アドレスを追加します。

この例では、`192.168.0.0/24`アドレス空間が提供されます。これは、このアドレス範囲内にあるKubernetesノードのIPが、公開されているTridentが作成するエクスポートポリシーに追加されることを示します。Tridentが実行されているノードを登録すると、ノードのIPアドレスを取得し、`autoExportCIDRs`で提供されたアドレスブロックと照合します。公開時にIPをフィルタリングした後、Tridentは公開先のノードのクライアントIPのエクスポート ポリシー ルールを作成します。

> `autoExportPolicy`と`autoExportCIDRs`は、バックエンドを作成した後に更新できます。自動的に管理されるバックエンドに新しい `CIDR` を追加したり、既存の `CIDR` を削除したりできます。`CIDR`を削除するときは、既存の接続が切断されないように注意してください。バックエンドの`autoExportPolicy`を無効にして、手動で作成したエクスポートポリシーにフォールバックすることもできます。これには、バックエンド構成で`exportPolicy`パラメータを設定する必要があります。

Trident がバックエンドを作成または更新したら、`tridentctl`または対応する `tridentbackend` CRD を使用してバックエンドを確認できます：

```
./tridentctl get backends ontap_nas_auto_export -n trident -o yaml
items:
- backendUUID: 403b5326-8482-40db-96d0-d83fb3f4daec
  config:
    aggregate: ""
    autoExportCIDRs:
    - 192.168.0.0/24
    autoExportPolicy: true
    backendName: ontap_nas_auto_export
    chapInitiatorSecret: ""
    chapTargetInitiatorSecret: ""
    chapTargetUsername: ""
    chapUsername: ""
    dataLIF: 192.168.0.135
    debug: false
    debugTraceFlags: null
    defaults:
      encryption: "false"
      exportPolicy: <automatic>
      fileSystemType: ext4
```

ノードが削除されると、Trident はすべてのエクスポート ポリシーをチェックして、ノードに対応するアクセス ルールを削除します。管理対象バックエンドのエクスポート ポリシーからこのノード IP を削除することで、Trident は、この IP がクラスター内の新しいノードによって再利用されない限り、不正なマウントを防止します。

既存のバックエンドの場合は、 `tridentctl update backend`でバックエンドを更新することで、Tridentがエクスポート ポリシーを自動的に管理するようになります。これにより、バックエンドのUUIDとqtree名にちなんで名付けられた2つの新しいエクスポート ポリシーが必要に応じて作成されます。バックエンドに存在するボリュームは、アンマウントされて再度マウントされた後、新しく作成されたエクスポート ポリシーを使用します。

メモ	自動管理エクスポート ポリシーを持つバックエンドを削除すると、動的に作成されたエクスポート ポリシーも削除されます。バックエンドが再作成されると、新しいバックエンドとして扱われ、新しいエクスポート ポリシーが作成されます。

ライブノードのIPアドレスが更新された場合は、ノード上のTridentポッドを再起動する必要があります。Tridentは、この IP 変更を反映するために、管理するバックエンドのエクスポート ポリシーを更新します。

SMB ボリュームのプロビジョニングの準備

少しの追加の準備をすれば、 `ontap-nas`ドライバーを使用してSMBボリュームをプロビジョニングできます。

警告	ONTAP オンプレミスクラスタ用の `ontap-nas-economy` SMB ボリュームを作成するには、SVM で NFS と SMB/CIFS プロトコルの両方を設定する必要があります。これらのプロトコルのいずれかを設定しないと、SMB ボリュームの作成が失敗します。

メモ	`autoExportPolicy` は SMB ボリュームではサポートされません。

開始する前に

SMB ボリュームをプロビジョニングする前に、次のものが必要です。

- Linux コントローラー ノードと、Windows Server 2022 を実行する少なくとも 1 つの Windows ワーカー ノードを備えた Kubernetes クラスター。Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。
- Active Directory のクレデンシャルを含む少なくとも 1 つの Trident シークレット。シークレットを生成するには `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- Windows サービスとして構成された CSI プロキシ。 `csi-proxy`を設定するには、Windows 上で実行されている Kubernetes ノード用の"GitHub：CSI Proxy"または"GitHub：Windows用CSIプロキシ"を参照してください。

手順

1. オンプレミス ONTAP の場合、必要に応じて SMB 共有を作成するか、 Trident に作成させることができます。

メモ	SMB 共有は Amazon FSx for ONTAP に必要です。

   SMB 管理共有は、"Microsoft管理コンソール" 共有フォルダスナップインまたは ONTAP CLI のいずれかを使用して、2 つの方法のいずれかで作成できます。ONTAP CLI を使用して SMB 共有を作成するには：

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

      `vserver cifs share create`コマンドは、共有の作成中に `-path` オプションで指定されたパスをチェックします。指定されたパスが存在しない場合、コマンドは失敗します。

   b. 指定された SVM に関連付けられた SMB 共有を作成します：

      ```
      vserver cifs share create -vserver vserver_name -share-name
      share_name -path path [-share-properties share_properties,...]
      [other_attributes] [-comment text]
      ```

   c. 共有が作成されたことを確認します：

      ```
      vserver cifs share show -share-name share_name
      ```

メモ	詳細については、"SMB共有を作成する"を参照してください。

2. バックエンドを作成するときは、SMB ボリュームを指定するために以下を構成する必要があります。すべての FSx for ONTAP バックエンドの設定オプションについては、"FSx for ONTAP 設定オプションと例"を参照してください。

パラメータ	概要	例
`smbShare`	次のいずれかを指定できます：Microsoft Management ConsoleまたはONTAP CLIを使用して作成されたSMB共有の名前、TridentがSMB共有を作成できるようにする名前、またはパラメータを空白のままにしてボリュームへの共通共有アクセスを防止できます。このパラメータは、オンプレミスONTAPではオプションです。このパラメータは、Amazon FSx for ONTAPバックエンドでは必須であり、空白にすることはできません。	`smb-share`
`nasType`	* `smb`に設定する必要があります。*nullの場合、デフォルトは `nfs`です。	`smb`
`securityStyle`	新しいボリュームのセキュリティ スタイル。**SMB** ボリュームの場合は、 **`ntfs`**または **`mixed`**に設定する必要があります。	`ntfs` または `mixed` SMB ボリューム用
`unixPermissions`	新しいボリュームのモード。**SMB** ボリュームの場合は空のままにする必要があります。	""

セキュア**SMB**を有効にする

25.06リリース以降、NetApp Tridentは、 `ontap-nas`および `ontap-nas-economy`バックエンドを使用して作成されたSMBボリュームの安全なプロビジョニングをサポートします。セキュアSMBを有効にすると、アクセス制御リスト（ACL）を使用して、Active Directory（AD）ユーザーおよびユーザーグループにSMB共有への制御されたアクセスを提供できます。

覚えておくべきポイント

- `ontap-nas-economy`ボリュームのインポートはサポートされていません。
- `ontap-nas-economy`ボリュームでは、読み取り専用クローンのみがサポートされています。
- セキュア SMB が有効になっている場合、Trident はバックエンドで指定された SMB 共有を無視します。
- PVC アノテーション、ストレージ クラス アノテーション、およびバックエンド フィールドを更新しても、SMB 共有 ACL は更新されません。
- クローン PVC のアノテーションで指定された SMB 共有 ACL は、ソース PVC の ACL よりも優先されます。
- セキュアな SMB を有効にする際には、有効な AD ユーザーを指定してください。無効なユーザーは ACL に追加されません。
- バックエンド、ストレージ クラス、PVC で同じ AD ユーザーに異なる権限を指定した場合、権限の優先順位は PVC、ストレージ クラス、バックエンドの順になります。
- セキュアSMBは `ontap-nas`管理対象ボリュームのインポートでサポートされており、管理対象外ボリュームのインポートには適用されません。

手順

1. 次の例のように、TridentBackendConfig で adAdminUser を指定してください：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.193.176.x
  svm: svm0
  useREST: true
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

2. ストレージ クラスに注釈を追加します。

   セキュアな SMB を確実に有効にするには、 `trident.netapp.io/smbShareAdUser`アノテーションをストレージ クラスに追加します。アノテーション `trident.netapp.io/smbShareAdUser`に指定されたユーザー値は、 `smbcreds`シークレットで指定されたユーザー名と同じである必要があります。 `smbShareAdUserPermission`には、 `full_control`、 change、または `read`のいずれかを選択できます。デフォルトの権限は `full_control`です。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1. PVCを作成します。

次の例では、PVC を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/snapshotDirectory: "true"
    trident.netapp.io/smbShareAccessControl: |
      read:
        - tridentADtest
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

ONTAP NAS 構成オプションと例

Tridentインストールで使用するONTAP NASドライバーの作成方法と使用方法について説明します。このセクションでは、バックエンドの設定例と、バックエンドを StorageClasses にマッピングするための詳細について説明します。25.10リリース以降、NetApp Tridentは"NetApp AFX ストレージ システム"をサポートします。NetApp AFXストレージシステムは、ストレージ レイヤの実装において、他のONTAPベースのシステム（ASA、AFF、FAS）とは異なります。

メモ	`ontap-nas`ドライバー（NFSプロトコル付き）のみがNetApp AFXシステムでサポートされています。SMBプロトコルはサポートされていません。

バックエンド構成オプション

Tridentバックエンド構成では、システムがNetApp AFXストレージシステムであることを指定する必要はありません。 `ontap-nas`を `storageDriverName`として選択すると、TridentはAFXストレージシステムを自動的に検出します。一部のバックエンド構成パラメータは、AFXストレージシステムには適用できません。

以下の表は、バックエンドの設定オプションを示しています：

パラメータ	概要	デフォルト
`version`		常に1

パラメータ	概要	デフォルト
`storageDriverName`	ストレージドライバーの名前 メモ NetApp AFXシステムでは、`ontap-nas`のみがサポートされます。	`ontap-nas`、`ontap-nas-economy`、または`ontap-nas-flexgroup`
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + dataLIF
`managementLIF`	クラスタまたはSVM管理LIFのIPアドレス（完全修飾ドメイン名（FQDN）も指定可能）Trident が IPv6 フラグを使用してインストールされている場合、IPv6 アドレスを使用するように設定できます。IPv6 アドレスは角括弧で定義する必要があります。例：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。シームレスなMetroClusterスイッチオーバーについては、MetroCluster の例を参照してください。	"10.0.0.1"、"[2001:1234:abcd::fefe]"
`dataLIF`	プロトコル LIF の IP アドレス。NetApp では`dataLIF`を指定することを推奨します。指定しない場合、`Trident` は `SVM` から `dataLIF` を取得します。`NFS` マウント操作に使用する完全修飾ドメイン名（`FQDN`）を指定することで、ラウンドロビン `DNS` を作成し、複数の `dataLIF` 間で負荷分散を行うことができます。初期設定後でも変更可能です。を参照してください。`Trident` が `IPv6` フラグを使用してインストールされている場合、`IPv6` アドレスを使用するように設定できます。`IPv6` アドレスは角括弧で定義する必要があります。例：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。**MetroCluster** の場合は省略します。MetroCluster の例を参照してください。	指定されたアドレス、または指定されていない場合は SVM から導出されます（非推奨）
`svm`	使用するストレージ仮想マシン **MetroCluster** の場合は省略。MetroCluster の例を参照してください。	SVM `managementLIF`が指定されている場合に導出されます
`autoExportPolicy`	自動エクスポート ポリシーの作成と更新を有効にします [ブール値]。`autoExportPolicy`および`autoExportCIDRs`オプションを使用すると、Tridentはエクスポート ポリシーを自動的に管理できます。	false
`autoExportCIDRs`	`autoExportPolicy`が有効になっている場合にKubernetesのノードIPをフィルタリングするためのCIDRのリスト。`autoExportPolicy`および`autoExportCIDRs`オプションを使用すると、Tridentはエクスポート ポリシーを自動的に管理できます。	["0.0.0.0/0", "::/0"]`
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	""
`clientCertificate`	クライアント証明書の Base64 エンコードされた値。証明書ベースの認証に使用	""
`clientPrivateKey`	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用	""

パラメータ	概要	デフォルト
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコードされた値。任意。証明書ベースの認証に使用	""
`username`	クラスタ/SVM に接続するためのユーザー名。クレデンシャルベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	
`password`	クラスター/SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	
`storagePrefix`	SVM で新しいボリュームをプロビジョニングするときに使用されるプレフィックス。設定後は更新できません メモ: ontap-nas-economy と 24 文字以上の storagePrefix を使用する場合、ボリューム名にはストレージ プレフィックスが含まれますが、qtree にはストレージ プレフィックスが埋め込まれません。	「trident」

パラメータ	概要	デフォルト
`aggregate`	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVM に割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバーの場合、このオプションは無視されます。割り当てられていない場合は、利用可能なアグリゲートのいずれかを使用してFlexGroupボリュームをプロビジョニングできます。 メモ: SVM でアグリゲートが更新されると、Trident Controller を再起動することなく、SVM をポーリングすることで Trident で自動的に更新されます。ボリュームをプロビジョニングするために Trident で特定のアグリゲートを設定している場合、そのアグリゲートの名前が変更されたり SVM から移動されたりすると、SVM アグリゲートのポーリング中にバックエンドが Trident で障害状態に移行します。バックエンドをオンラインに戻すには、アグリゲートを SVM 上に存在するものに変更するか、完全に削除する必要があります。 **AFX**ストレージシステムには指定しないでください。	""
`limitAggregateUsage`	使用率がこのパーセンテージを超える場合、プロビジョニングは失敗します。**Amazon FSx for ONTAP**には適用されません。**AFX**ストレージシステムには指定しないでください。	""（デフォルトでは強制されません）

パラメータ	概要	デフォルト
flexgroupAggregateList	プロビジョニング用のアグリゲートのリスト（オプション。設定する場合は、SVMに割り当てる必要があります）。SVMに割り当てられたすべてのアグリゲートは、FlexGroupボリュームのプロビジョニングに使用されます。*ontap-nas-flexgroup*ストレージドライバーでサポートされています。 メモ: SVMでアグリゲートリストが更新されると、Trident ControllerをTridentせずにSVMをポーリングすることで、リストはTridentで自動的に更新されます。Tridentでボリュームをプロビジョニングするために特定のアグリゲートリストを設定している場合、アグリゲートリストの名前が変更されたり、SVMから移動されたりすると、SVMアグリゲートのポーリング中にバックエンドがTridentで障害状態に移行します。バックエンドをオンラインに戻すには、アグリゲートリストをSVM上に存在するリストに変更するか、完全に削除する必要があります。	""
`limitVolumeSize`	要求されたボリューム サイズがこの値を超える場合、プロビジョニングは失敗します。	""（デフォルトでは強制されません）
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例：{"api":false, "method":true} `debugTraceFlags`を使用しないでください。ただし、トラブルシューティングを行っており、詳細なログ ダンプが必要な場合を除きます。	null
`nasType`	NFS または SMB ボリュームの作成を設定します。オプションは `nfs`、 `smb`、またはnullです。null に設定すると、デフォルトで NFS ボリュームになります。指定する場合は、**AFX**ストレージシステムでは常に`**nfs**`に設定してください。	`nfs`
`nfsMountOptions`	NFS マウント オプションのコンマ区切りリスト。Kubernetes 永続ボリュームのマウント オプションは通常ストレージ クラスで指定されますが、ストレージ クラスでマウント オプションが指定されていない場合、Trident はストレージ バックエンドの構成ファイルで指定されたマウント オプションを使用するようになります。ストレージ クラスまたは構成ファイルにマウント オプションが指定されていない場合、Trident は関連付けられている永続ボリュームにマウント オプションを設定しません。	""
`qtreesPerFlexvol`	FlexVol あたりの最大 Qtree 数は、範囲 [50, 300] 内である必要があります	"200"

パラメータ	概要	デフォルト
`smbShare`	次のいずれかを指定できます：Microsoft Management ConsoleまたはONTAP CLIを使用して作成されたSMB共有の名前、TridentがSMB共有を作成できるようにする名前、またはパラメータを空白のままにしてボリュームへの共通共有アクセスを防止できます。このパラメータは、オンプレミスONTAPではオプションです。このパラメータは、Amazon FSx for ONTAPバックエンドでは必須であり、空白にすることはできません。	`smb-share`
`useREST`	ONTAP REST APIを使用するためのブーリアン パラメータ。 `useREST`に設定すると `true`、TridentはONTAP REST APIを使用してバックエンドと通信します。 `false`に設定すると、TridentはONTAPI（ZAPI）呼び出しを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、`ontapi`アプリケーションへのアクセス権が必要です。これは、事前定義された `vsadmin`および`cluster-admin`ロールで満たされます。Trident 24.06リリースおよびONTAP 9.15.1以降では、 `useREST`はデフォルトで `true`に設定されます。ONTAPI（ZAPI）呼び出しを使用するには、 `useREST`を`false`に変更します。指定する場合は、**AFX**ストレージシステムでは常に `**true**`に設定してください。	ONTAP 9.15.1以降の場合は`true`、それ以外の場合は `false`。
`limitVolumePoolSize`	ontap-nas-economy バックエンドで Qtree を使用する場合の最大リクエスト可能 FlexVol サイズ。	""（デフォルトでは強制されません）
`denyNewVolumePools`	バックエンドが Qtree を格納する新しい FlexVol ボリュームを作成できないように制限 `ontap-nas-economy`します。新しい PV のプロビジョニングには、既存の Flexvol のみが使用されます。	
`adAdminUser`	SMB 共有へのフルアクセス権を持つ Active Directory 管理者ユーザーまたはユーザー グループ。このパラメータを使用して、SMB 共有へのフルコントロールの管理者権限を付与します。	

ボリュームのプロビジョニング用のバックエンド設定オプション

デフォルトのプロビジョニングは、設定の `defaults`セクションにあるこれらのオプションを使用して制御できます。例については、以下の設定例を参照してください。

パラメータ	概要	デフォルト
`spaceAllocation`	Qtreeのスペース割り当て	"true"
`spaceReserve`	スペース予約モード：「none」（シン）または「volume」（シック）	「なし」
`snapshotPolicy`	使用するSnapshotポリシー	「なし」

パラメータ	概要	デフォルト
`qosPolicy`	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージプール/バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください	""
`adaptiveQosPolicy`	作成されたボリュームに割り当てるアダプティブ QoS ポリシー グループ。ストレージ プール/バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。ontap-nas-economy ではサポートされていません。	""
`snapshotReserve`	Snapshot 用に予約されているボリュームの割合	`snapshotPolicy`が「none」の場合は「0」、それ以外の場合は「」
`splitOnClone`	作成時にクローンを親から分離する	"false"
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたボリュームはすべてNAEが有効になります。詳細については、次を参照してください：Tridentと NVE および NAE の連携。	"false"
`tieringPolicy`	階層化ポリシーで「none」を使用	
`unixPermissions`	新しいボリュームのモード	NFS ボリュームの場合は「777」、SMB ボリュームの場合は空（該当なし）
`snapshotDir`	`.snapshot`ディレクトリへのアクセスを制御します	`true`、 `false` （明示的に設定）。
`exportPolicy`	使用するエクスポートポリシー	"default"
`securityStyle`	新しいボリュームのセキュリティ スタイル。NFSは `mixed`および `unix`セキュリティ スタイルをサポートします。SMBは `mixed`および `ntfs`セキュリティ スタイルをサポートします。	NFSのデフォルトは `unix`です。SMBのデフォルトは `ntfs`です。
`nameTemplate`	カスタムボリューム名を作成するためのテンプレート。	""

メモ	Trident で QoS ポリシー グループを使用するには、ONTAP 9.8 以降が必要です。共有されていない QoS ポリシーグループを使用し、ポリシーグループが各構成要素に個別に適用されるようにする必要があります。共有 QoS ポリシー グループは、すべてのワークロードの合計スループットの上限を適用します。

ボリュームプロビジョニングの例

デフォルトを定義した例を次に示します：

```
---
version: 1
storageDriverName: ontap-nas
backendName: customBackendName
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
labels:
  k8scluster: dev1
  backend: dev1-nasbackend
svm: trident_svm
username: cluster-admin
password: <password>
limitAggregateUsage: 80%
limitVolumeSize: 50Gi
nfsMountOptions: nfsvers=4
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: premium
  exportPolicy: myk8scluster
  snapshotPolicy: default
  snapshotReserve: "10"
```

`ontap-nas`および `ontap-nas-flexgroups`の場合、Tridentは新しい計算方法を使用して、FlexVolがsnapshotReserveのパーセンテージとPVCで正しくサイズ設定されるようにします。ユーザーがPVCを要求すると、Tridentは新しい計算方法を使用して、より多くのスペースを持つ元のFlexVolを作成します。この計算により、ユーザーはPVCで要求した書き込み可能なスペースを確実に受け取ることができ、要求したスペースよりも少ないスペースを受け取ることはありません。v21.07より前では、ユーザーがPVC（たとえば5GiB）を要求し、snapshotReserveを50パーセントにすると、書き込み可能なスペースは2.5GiBしか得られませんでした。これは、ユーザーが要求したのはボリューム全体であり、`snapshotReserve`はその割合であるためです。Trident 21.07では、ユーザーが要求するのは書き込み可能なスペースであり、Tridentは`snapshotReserve`の数値をボリューム全体の割合として定義します。これは `ontap-nas-economy`には適用されません。これがどのように機能するかを確認するには、次の例を参照してください：

計算は次のとおりです：

```
Total volume size = <PVC requested size> / (1 - (<snapshotReserve
percentage> / 100))
```

snapshotReserve = 50%、PVC要求 = 5 GiBの場合、ボリュームの合計サイズは5/.5 = 10 GiBとなり、使用可能なサイズは5 GiBになります。これは、ユーザーがPVC要求で要求したサイズです。 `volume show`コマンドを実行すると、次の例のような結果が表示されます：

以前のインストールからの既存のバックエンドは、Trident のアップグレード時に上記のようにボリュームをプロビジョニングします。アップグレード前に作成したボリュームの場合は、変更を反映させるためにボリュームのサイズを変更する必要があります。たとえば、 `snapshotReserve=50`を使用した 2 GiB の PVC では、以前は 1 GiB の書き込み可能なスペースを提供するボリュームが作成されました。たとえば、ボリュームのサイズを 3 GiB に変更すると、6 GiB のボリューム上で 3 GiB の書き込み可能な領域がアプリケーションに提供されます。

最小限の構成例

次の例は、ほとんどのパラメータをデフォルトのままにする基本構成を示しています。これはバックエンドを定義する最も簡単な方法です。

メモ	Amazon FSx for NetApp ONTAP を Trident とともに使用している場合は、LIF に IP アドレスではなく DNS 名を指定することを推奨します。

ONTAP NASエコノミーの例

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

ONTAP NAS FlexGroupの例

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

MetroCluster の例

"SVMのレプリケーションとリカバリ"中のスイッチオーバーとスイッチバック後にバックエンド定義を手動で更新する必要がないように、バックエンドを設定できます。

シームレスなスイッチオーバーとスイッチバックを行うには、 `managementLIF`を使用してSVMを指定し、 `dataLIF`および `svm`パラメータは省略します。例：

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

SMB ボリュームの例

```
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
nasType: smb
securityStyle: ntfs
unixPermissions: ""
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

証明書ベースの認証の例

これは最小限のバックエンド構成の例です。 `clientCertificate`、 `clientPrivateKey`、および `trustedCACertificate`（信頼できるCAを使用する場合はオプション）は `backend.json`に入力され、クライアント証明書、秘密キー、信頼できるCA証明書のbase64エンコードされた値をそれぞれ取得します。

```
---
version: 1
backendName: DefaultNASBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

自動エクスポートポリシーの例

この例では、Tridentに動的エクスポート ポリシーを使用してエクスポート ポリシーを自動的に作成および管理するように指示する方法を示します。これは `ontap-nas-economy`ドライバと `ontap-nas-flexgroup`ドライバで同じように機能します。

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-nasbackend
autoExportPolicy: true
autoExportCIDRs:
- 10.0.0.0/24
username: admin
password: password
nfsMountOptions: nfsvers=4
```

IPv6アドレスの例

この例は、 `managementLIF` を使用した IPv6 アドレスを示しています。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

Amazon FSx for ONTAP を使用した **SMB** ボリュームの例

`smbShare`パラメータは、`SMB` ボリュームを使用する `FSx for ONTAP` に必要です。

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
dataLIF: 10.0.0.15
svm: nfs_svm
smbShare: smb-share
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

nameTemplateを使用したバックエンド構成の例

```
---
version: 1
storageDriverName: ontap-nas
backendName: ontap-nas-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

仮想プールを使用したバックエンドの例

以下に示すサンプルのバックエンド定義ファイルでは、すべてのストレージ プールに対して特定のデフォルトが設定されています。たとえば、 `spaceReserve`は none、 `spaceAllocation`は false、 `encryption`は false です。仮想プールはストレージ セクションで定義されます。

Tridentは「コメント」フィールドにプロビジョニング ラベルを設定します。コメントは FlexVol の場合は `ontap-nas`、FlexGroup の場合は `ontap-nas-flexgroup`に設定されます。Trident は、プロビジョニング時に仮想プールに存在するすべてのラベルをストレージ ボリュームにコピーします。便宜上、ストレージ管理者は仮想プールごとにラベルを定義し、ラベルごとにボリュームをグループ化できます。

これらの例では、一部のストレージプールは独自の `spaceReserve`、 `spaceAllocation`、および `encryption`値を設定し、一部のプールはデフォルト値を上書きします。

ONTAP NAS の例

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
svm: svm_nfs
username: admin
password: <password>
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: "false"
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      app: msoffice
      cost: "100"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
      adaptiveQosPolicy: adaptive-premium
  - labels:
      app: slack
      cost: "75"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
```

```
      app: wordpress
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS FlexGroupの例

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "50000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: gold
      creditpoints: "30000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
```

```
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NASエコノミーの例

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: nas_economy_store
region: us_east_1
storage:
  - labels:
      department: finance
      creditpoints: "6000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: engineering
      creditpoints: "3000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      department: humanresource
      creditpoints: "2000"
    zone: us_east_1d
    defaults:
```

```
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

バックエンドを**StorageClasses**にマッピングする

次のStorageClass定義は[仮想プールを使用したバックエンドの例]を参照しています。 `parameters.selector` フィールドを使用して、各StorageClassはボリュームをホストするために使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プールで定義された側面が設定されます。

- この `protection-gold` StorageClass は、 `ontap-nas-flexgroup` バックエンドの最初と 2 番目の仮想プールにマッピングされます。これらは、ゴールド レベルの保護を提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- `protection-not-gold` StorageClassは、 `ontap-nas-flexgroup`バックエンドの3番目と4番目の仮想プールにマッピングされます。これらは、ゴールド以外の保護レベルを提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- `app-mysqldb`StorageClassは、 `ontap-nas`バックエンドの4番目の仮想プールにマッピングされます。これは、mysqldbタイプのアプリ用のストレージプール構成を提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- `protection-silver-creditpoints-20k` StorageClassは `ontap-nas-flexgroup`バックエンドの3番目の仮想プールにマッピングされます。これは、シルバーレベルの保護と20000クレジットポイントを提供する唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- `creditpoints-5k` StorageClassは、 `ontap-nas`バックエンドの3番目の仮想プールと `ontap-nas-economy`バックエンドの2番目の仮想プールにマッピングされます。これらは5000クレジットポイントで提供される唯一のプールです。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

Trident は、どの仮想プールが選択されるかを決定し、ストレージ要件が満たされていることを確認します。

初期設定後にアップデート `dataLIF`

次のコマンドを実行して、更新された dataLIF を含む新しいバックエンド JSON ファイルを提供することにより、初期構成後に dataLIF を変更できます。

```
tridentctl update backend <backend-name> -f <path-to-backend-json-file-
with-updated-dataLIF>
```

メモ	PVC が 1 つまたは複数のポッドに接続されている場合、新しい dataLIF を有効にするには、対応するすべてのポッドを停止してから再度起動する必要があります。

セキュアな **SMB** の例

ontap-nas ドライバを使用したバックエンド構成

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

ontap-nas-economy ドライバを使用したバックエンド構成

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas-economy
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

ストレージ プールを使用したバックエンド構成

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm0
  useREST: false
  storage:
  - labels:
      app: msoffice
    defaults:
      adAdminUser: tridentADuser
  nasType: smb
  credentials:
    name: backend-tbc-ontap-invest-secret
```

ontap-nas ドライバを使用したストレージクラスの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADtest
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

メモ	`annotations`を追加して、セキュアな SMB を有効にしてください。バックエンドまたは PVC で設定された構成に関係なく、アノテーションがないとセキュアな SMB は機能しません。

ontap-nas-economy ドライバを使用したストレージクラスの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser3
parameters:
  backendType: ontap-nas-economy
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

単一の **AD** ユーザを使用した **PVC** の例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      change:
        - tridentADtest
      read:
        - tridentADuser
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

複数の **AD** ユーザを使用した **PVC** の例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-test-pvc
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      full_control:
        - tridentTestuser
        - tridentuser
        - tridentTestuser1
        - tridentuser1
      change:
        - tridentADuser
        - tridentADuser1
        - tridentADuser4
        - tridentTestuser2
      read:
        - tridentTestuser2
        - tridentTestuser3
        - tridentADuser2
        - tridentADuser3
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP で Trident を使用

"Amazon FSx for NetApp ONTAP"は、NetApp ONTAPストレージオペレーティングシステムを基盤とするファイルシステムを実行するフルマネージドAWSサービスです。AWSの拡張性と運用上の簡便性を備えたONTAPの機能、パフォーマンス、管理性を提供します。ファイルシステムはAmazon FSxの主要なリソースであり、オンプレミスのONTAPクラスタに相当します。各ファイルシステムには1つ以上のストレージ仮想マシン（SVM）が含まれており、各SVMにはファイルやディレクトリを格納する1つ以上のボリュームが含まれています。この統合により、Amazon Elastic Kubernetes Service（EKS）で実行されているKubernetesクラスターは、ブロックワークロードおよびファイルワークロード向けにONTAPをバックエンドとする永続ボリュームをプロビジョニングできるようになります。

要件

"Tridentの要件"に加えて、FSx for ONTAPをTridentと統合するには、次のものが必要です：

- `kubectl`がインストールされている既存のAmazon EKSクラスタまたはセルフマネージドKubernetesクラスタ。
- クラスタのワーカーノードからアクセス可能な既存のAmazon FSx for NetApp ONTAPファイルシステムとStorage Virtual Machine（SVM）。
- "NFS または iSCSI"用に準備されたワーカーノード。

メモ	EKS AMI タイプに応じて、Amazon Linux および Ubuntu "Amazon Machine Images"（AMI）に必要なノード準備手順に従ってください。

考慮事項

- SMB ボリューム：
  - SMBボリュームは、 `ontap-nas`ドライバーのみを使用してサポートされます。
  - SMB ボリュームは Trident EKS アドオンではサポートされていません。
  - Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。詳細については、"SMB ボリュームのプロビジョニングの準備"を参照してください。
- Trident 24.02より前では、自動バックアップが有効になっているAmazon FSxファイルシステムで作成されたボリュームは、Tridentで削除できませんでした。Trident 24.02以降でこの問題を防ぐには、Amazon FSx for ONTAPのバックエンド設定ファイルで `fsxFilesystemID`、AWS `apiRegion`、AWS `apikey`、およびAWS `secretKey`を指定します。

メモ	IAM ロールを Trident に指定する場合は、 `apiRegion`、 `apiKey`、および `secretKey`フィールドを Trident に明示的に指定することを省略できます。詳細については、"FSx for ONTAP 設定オプションと例"を参照してください。

Trident SAN/iSCSI と **EBS-CSI** ドライバの同時使用

AWS（EKS、ROSA、EC2、またはその他のインスタンス）でontap-sanドライバー（iSCSIなど）を使用する予定の場合、ノードで必要なマルチパス構成がAmazon Elastic Block Store（EBS）CSIドライバーと競合する可能性があります。同じノード上のEBSディスクに干渉することなくマルチパスが機能するようにするには、マルチパス設定でEBSを除外する必要があります。この例は、EBSディスクをマルチパスから除外しながら、必要なTrident設定を含む `multipath.conf`ファイルを示しています：

```
defaults {
    find_multipaths no
}
blacklist {
    device {
        vendor "NVME"
        product "Amazon Elastic Block Store"
    }
}
```

認証

Tridentには2つの認証モードがあります。

- 認証情報ベース（推奨）：認証情報を AWS Secrets Manager に安全に保存します。ファイルシステムの `fsxadmin`ユーザーまたは SVM 用に設定された `vsadmin`ユーザーを使用できます。

  警告

  Trident は `vsadmin` SVM ユーザーとして、または同じロールを持つ別の名前のユーザーとして実行されることを想定しています。Amazon FSx for NetApp ONTAP には `fsxadmin` ユーザーがあり、これは ONTAP `admin` クラスター ユーザーの限定的な代替品です。`vsadmin` を Trident と併用することを強くお勧めします。

- 証明書ベース：Trident は、SVM にインストールされた証明書を使用して、FSx ファイル システム上の SVM と通信します。

認証を有効にする方法の詳細については、ドライバー タイプの認証を参照してください：

- "ONTAP NAS 認証"
- "ONTAP SAN 認証"

テスト済みの **Amazon** マシンイメージ（**AMI**）

EKS クラスターはさまざまなオペレーティングシステムをサポートしていますが、AWS は特定の Amazon マシンイメージ（AMI）をコンテナと EKS 用に最適化しています。以下の AMI は NetApp Trident 25.02 でテスト済みです。

AMI	NAS	NASエコノミー	iSCSI	iSCSIエコノミー
AL2023_x86_64_STANDARDを使用したチャンク アップロード署名要求がサポートされるようになりました。	はい	はい	はい	はい
AL2_x86_64	はい	はい	○*	○*

BOTTLEROCKET_x86_64を使用したチャンク アップロード署名要求がサポートされるようになりました。	はい**	はい	該当なし	該当なし
AL2023_ARM_64_STANDARDを使用したチャンク アップロード署名要求がサポートされるようになりました。	はい	はい	はい	はい
AL2_ARM_64	はい	はい	○*	○*
BOTTLEROCKET_ARM_64	はい**	はい	該当なし	該当なし

- * ノードを再起動せずに PV を削除することはできません
- ** Trident バージョン 25.02 の NFSv3 では動作しません。

メモ	必要な AMI がここにリストされていない場合、それはサポートされていないという意味ではなく、単にテストされていないという意味です。このリストは、動作することがわかっている AMI のガイドとして機能します。

テストに使用したデバイス：

- EKS version: 1.32
- インストール方法：Helm 25.06 および AWS アドオン 25.06
- NAS については、NFSv3 と NFSv4.1 の両方がテストされました。
- SAN の場合、NVMe-oF ではなく iSCSI のみがテストされました。

実行されたテスト：

- 作成：Storage Class、pvc、pod
- 削除：ポッド、PVC（レギュラー、qtree/lun – エコノミー、AWS バックアップ付き NAS）

詳細情報の参照

- "Amazon FSx for NetApp ONTAP ドキュメント"
- "Amazon FSx for NetApp ONTAP に関するブログ投稿"

IAMロールと**AWS**シークレットを作成する

明示的な AWS 認証情報を提供する代わりに、 AWS IAM ロールとして認証することで、Kubernetes ポッドが AWS リソースにアクセスするように設定できます。

メモ	AWS IAM ロールを使用して認証するには、EKS を使用して Kubernetes クラスタを導入する必要があります。

AWS Secrets Manager シークレットを作成する

Trident はストレージを管理するために FSx vserver に対して API を発行するため、そのためには資格情報が必要になります。これらの認証情報を渡す安全な方法は、AWS Secrets Manager シークレットを使用することです。したがって、まだお持ちでない場合は、vsadmin アカウントの認証情報を含む AWS Secrets Manager シークレットを作成する必要があります。

この例では、Trident CSIの資格情報を保存するためのAWS Secrets Managerシークレットを作成します：

```
aws secretsmanager create-secret --name trident-secret --description
"Trident CSI credentials"\
  --secret-string
"{\"username\":\"vsadmin\",\"password\":\"<svmpassword>\"}"
```

IAM ポリシーを作成する

Tridentを正しく実行するには、AWS権限も必要です。したがって、Tridentに必要な権限を付与するポリシーを作成する必要があります。

次の例では、AWS CLI を使用して IAM ポリシーを作成します：

```
aws iam create-policy --policy-name AmazonFSxNCSIDriverPolicy --policy
-document file://policy.json
  --description "This policy grants access to Trident CSI to FSxN and
Secrets manager"
```

ポリシー **JSON** の例：

```
{
  "Statement": [
    {
      "Action": [
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes",
        "fsx:CreateVolume",
        "fsx:RestoreVolumeFromSnapshot",
        "fsx:DescribeStorageVirtualMachines",
        "fsx:UntagResource",
        "fsx:UpdateVolume",
        "fsx:TagResource",
        "fsx:DeleteVolume"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Resource": "arn:aws:secretsmanager:<aws-region>:<aws-account-
id>:secret:<aws-secret-manager-name>*"
    }
  ],
  "Version": "2012-10-17"
}
```

Pod Identity またはサービス アカウントの関連付け（**IRSA**）用の **IAM** ロールを作成する

EKS Pod Identity を持つ AWS Identity and Access Management （IAM） ロール、またはサービスアカウントの関連付け （IRSA） の IAM ロールを引き受けるように Kubernetes サービスアカウントを設定できます。サービスアカウントを使用するように設定されたすべてのポッドは、ロールがアクセス権限を持つすべての AWS サービスにアクセスできるようになります。

Pod アイデンティティ

Amazon EKS Pod Identity associationsは、Amazon EC2インスタンスプロファイルがAmazon EC2インスタンスにクレデンシャルを提供するのと同様に、アプリケーションのクレデンシャルを管理する機能を提供します。

EKS クラスタに **Pod Identity** をインストールします：

AWS コンソールまたは次の AWS CLI コマンドを使用して、Pod identity を作成できます。

```
aws eks create-addon --cluster-name <EKS_CLUSTER_NAME> --addon-name
eks-pod-identity-agent
```

詳細については、"Amazon EKS Pod Identity Agent を設定する"を参照してください。

trust-relationship.json を作成：

EKS サービス プリンシパルが Pod Identity に対してこのロールを引き受けることができるように、trust-relationship.json を作成します。次に、この信頼ポリシーを持つロールを作成します：

```
aws iam create-role \
  --role-name fsxn-csi-role --assume-role-policy-document file://trust-
relationship.json \
  --description "fsxn csi pod identity role"
```

trust-relationship.json ファイル：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

IAM ロールにロールポリシーをアタッチします：

前の手順のロールポリシーを、作成した IAM ロールにアタッチします：

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:111122223333:policy/fsxn-csi-policy \
  --role-name fsxn-csi-role
```

ポッド **ID** の関連付けを作成します：

IAMロールとTridentサービスアカウント（trident-controller）の間にポッドIDの関連付けを作成する

```
aws eks create-pod-identity-association \
  --cluster-name <EKS_CLUSTER_NAME> \
  --role-arn arn:aws:iam::111122223333:role/fsxn-csi-role \
  --namespace trident --service-account trident-controller
```

サービス アカウントの関連付け（**IRSA**）の **IAM** ロール

AWS CLI の使用：

```
aws iam create-role --role-name AmazonEKS_FSxN_CSI_DriverRole \
  --assume-role-policy-document file://trust-relationship.json
```

trust-relationship.json ファイル：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<account_id>:oidc-
provider/<oidc_provider>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<oidc_provider>:aud": "sts.amazonaws.com",
          "<oidc_provider>:sub":
"system:serviceaccount:trident:trident-controller"
        }
      }
    }
  ]
}
```

`trust-relationship.json` ファイルで次の値を更新します：

- **<account_id>** - AWSアカウントID
- **<oidc_provider>** - EKS クラスタの OIDC。oidc_provider は次のコマンドを実行して取得できます：

```
aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer"\
  --output text | sed -e "s/^https:\/\///"
```

IAM ロールを **IAM** ポリシーにアタッチします：

ロールが作成されたら、次のコマンドを使用して、（上記の手順で作成された）ポリシーをロールにアタッチします：

```
aws iam attach-role-policy --role-name my-role --policy-arn <IAM policy
ARN>
```

OICD プロバイダーが関連付けられていることを確認します：

OIDC プロバイダーがクラスターに関連付けられていることを確認します。次のコマンドを使用して確認できます：

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

出力が空の場合は、次のコマンドを使用して IAM OIDC をクラスタに関連付けます：

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name
--approve
```

eksctl を使用している場合、次の例を使用して EKS のサービス アカウントの IAM ロールを作成します：

```
eksctl create iamserviceaccount --name trident-controller --namespace
trident \
  --cluster <my-cluster> --role-name AmazonEKS_FSxN_CSI_DriverRole
--role-only \
  --attach-policy-arn <IAM-Policy ARN> --approve
```

Tridentをインストール

Trident は、Kubernetes における Amazon FSx for NetApp ONTAP のストレージ管理を合理化し、開発者と管理者がアプリケーションの展開に集中できるようにします。Tridentは次のいずれかの方法を使用してインストールできます：

- Helm
- EKS アドオン

スナップショット機能を利用する場合は、CSI スナップショット コントローラー アドオンをインストールします。詳細については、"CSI ボリュームのスナップショット機能を有効にする" を参照してください。

helm 経由で Trident をインストール

Pod アイデンティティ

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. 次の例を使用して Trident をインストールします：

```
helm install trident-operator netapp-trident/trident-operator --version 100.2502.1 --namespace trident --create-namespace
```

`helm list`コマンドを使用して、名前、ネームスペース、チャート、ステータス、アプリケーションバージョン、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。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                           APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2502.0    25.02.0
```

サービス アカウント アソシエーション（**IRSA**）

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. **cloud provider** と **cloud identity** の値を設定します。

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1  \
--set cloudProvider="AWS" \
--set cloudIdentity="'eks.amazonaws.com/role-arn:
arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'" \
--namespace trident \
--create-namespace
```

`helm list`コマンドを使用して、名前、ネームスペース、チャート、ステータス、アプリケーションバージョン、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。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2510.0    25.10.0
```

メモ

iSCSI を使用する予定の場合は、クライアント マシンで iSCSI が有効になっていることを確認してください。AL2023 Worker node OS を使用している場合は、helm インストールで node prep パラメータを追加することで、iSCSI クライアントのインストールを自動化できます：

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1 --namespace trident --create-namespace --
set nodePrep={iscsi}
```

EKS アドオン経由で **Trident** をインストール

Trident EKS アドオンには最新のセキュリティパッチとバグ修正が含まれており、Amazon EKS で動作することが AWS によって検証されています。EKS アドオンを使用すると、Amazon EKS クラスターの安全性と安定性を常に確保し、アドオンのインストール、設定、更新に必要な作業量を削減できます。

前提条件

AWS EKS の Trident アドオンを設定する前に、以下のものを用意してください：

- アドオン サブスクリプション付きの Amazon EKS クラスタ アカウント

- AWS マーケットプレイスへの AWS 権限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMI タイプ：Amazon Linux 2（AL2_x86_64）または Amazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存の Amazon FSx for NetApp ONTAP ファイルシステム

AWS の **Trident** アドオンを有効にする

管理コンソール

1. Amazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーション ペインで、* Clusters * を選択します。
3. NetApp Trident CSI アドオンを設定するクラスタの名前を選択します。
4. *アドオン*を選択し、*さらにアドオンを取得*を選択します。
5. アドオンを選択するには、次の手順に従います。
   a. **AWS Marketplace** アドオン セクションまでスクロールし、検索ボックスに **"Trident"** と入力します。
   b. Trident by NetApp ボックスの右上隅にあるチェックボックスをオンにします。
   c. **Next** を選択します。
6. *選択したアドオンの構成*設定ページで、次の操作を行います：

メモ	**Pod Identity** 関連付けを使用している場合は、これらの手順をスキップしてください。

   a. 使用する*バージョン*を選択します。
   b. IRSA 認証を使用している場合は、オプション構成設定で使用可能な構成値を必ず設定してください：
      - 使用する*バージョン*を選択します。
      - *アドオン構成スキーマ*に従って、*構成値*セクションの*configurationValues*パラメータを、前の手順で作成した role-arn に設定します（値は次の形式にする必要があります）：

```
{

  "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
  "cloudProvider": "AWS"

}
```

+
競合解決方法として [オーバーライド] を選択した場合、既存のアドオンの 1 つ以上の設定を Amazon EKS アドオン設定で上書きできます。このオプションを有効にせず、既存の設定と競合する場合、操作は失敗します。結果のエラーメッセージを使用して競合のトラブルシューティングを行うことができます。このオプションを選択する前に、Amazon EKS アドオンが自己管理する必要がある設定を管理していないことを確認してください。

7. **Next** を選択します。
8. *確認と追加*ページで、*作成*を選択します。

   アドオンのインストールが完了すると、インストールされたアドオンが表示されます。

AWS CLI

1. `add-on.json`ファイルを作成する：

Pod Identity の場合は、次の形式を使用します：

メモ　ビジネス

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
}
```

IRSA 認証の場合は、次の形式を使用します：

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
  "serviceAccountRoleArn": "<role ARN>",
  "configurationValues": {
    "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
    "cloudProvider": "AWS"
  }
}
```

メモ　`<role ARN>`を、前の手順で作成されたロールの ARN に置き換えます。

2.Trident EKS アドオンをインストールします。

```
aws eks create-addon --cli-input-json file://add-on.json
```

eksctl

次のコマンド例では、Trident EKS アドオンをインストールします：

```
eksctl create addon --name netapp_trident-operator --cluster
<cluster_name> --force
```

Trident EKS アドオンを更新する

管理コンソール

1. Amazon EKS コンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーション ペインで、* Clusters * を選択します。
3. NetApp Trident CSI アドオンを更新するクラスターの名前を選択します。
4. アドオン タブを選択します。
5. **Trident by NetApp** を選択し、*編集*を選択します。
6. *NetApp による Trident の設定*ページで、次の操作を行います：
   a. 使用する*バージョン*を選択します。
   b. * オプションの構成設定 * を展開し、必要に応じて変更します。
   c. 変更を保存 を選択します。

AWS CLI

次の例では、EKS add-on を更新します。

```
aws eks update-addon --cluster-name <eks_cluster_name> --addon-name
netapp_trident-operator --addon-version v25.6.0-eksbuild.1 \
  --service-account-role-arn <role-ARN> --resolve-conflict preserve \
  --configuration-values "{\"cloudIdentity\":
\"'eks.amazonaws.com/role-arn: <role ARN>'\"}"
```

eksctl

- FSxN Trident CSI アドオンの現在のバージョンを確認してください。 `my-cluster`をクラスター名に置き換えます。

  ```
  eksctl get addon --name netapp_trident-operator --cluster my-cluster
  ```

  出力例：

```
NAME                        VERSION             STATUS    ISSUES
IAMROLE    UPDATE AVAILABLE    CONFIGURATION VALUES
netapp_trident-operator    v25.6.0-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- 前の手順の出力の UPDATE AVAILABLE で返されたバージョンにアドオン ソフトウェアを更新します。

```
eksctl update addon --name netapp_trident-operator --version
v25.6.0-eksbuild.1 --cluster my-cluster --force
```

`--force`オプションを削除し、Amazon EKS
アドオン設定のいずれかが既存の設定と競合する場合、Amazon EKS
アドオンの更新は失敗し、競合を解決するためのエラーメッセージが表示されます。このオプションを指定する前に、Amazon EKS
アドオンが管理する必要のある設定を管理していないことを確認してください。このオプションによってそれらの設定が上書きされるためです。この設定の他のオプションの詳細については、link:https://eksctl.io/usage/addons/["アドオン"]を参照してください。Amazon EKS Kubernetes
フィールド管理の詳細については、link:https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-field-management.html["Kubernetes
フィールド管理"]を参照してください。

Trident EKS アドオンをアンインストール/削除します

Amazon EKS アドオンを削除するには、次の 2 つのオプションがあります：

- クラスター上のアドオン ソフトウェアを保持する – このオプションを選択すると、Amazon EKS によるすべての設定の管理が削除されます。また、Amazon EKS が更新を通知し、更新を開始した後に Amazon EKS アドオンを自動的に更新する機能も削除されます。ただし、クラスター上のアドオン ソフトウェアは保持されます。このオプションを選択すると、アドオンは Amazon EKS アドオンではなく、自己管理型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。アドオンを保持するには、コマンドで `--preserve`オプションを保持します。
- アドオン ソフトウェアをクラスタから完全に削除する – NetApp は、クラスタ上にそのアドオンに依存するリソースが存在しない場合のみ、Amazon EKS アドオンをクラスタから削除することを推奨しています。アドオンを削除するには、 `--preserve` オプションを `delete` コマンドから削除してください。

メモ	アドオンに IAM アカウントが関連付けられている場合、 IAM アカウントは削除されません。

管理コンソール

1. Amazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーション ペインで、* Clusters * を選択します。
3. NetApp Trident CSI アドオンを削除するクラスターの名前を選択します。
4. アドオン*タブを選択し、***NetApp**による**Trident***を選択します。
5. *削除*を選択します。
6. *netapp_trident-operator の削除確認*ダイアログで、次の操作を行います：
   a. Amazon EKS によるアドオン設定の管理を停止する場合は、**Preserve on cluster** を選択します。クラスター上のアドオン ソフトウェアを保持し、アドオンのすべての設定を自分で管理する場合は、これを行ってください。
   b. **netapp_trident-operator** と入力します。
   c. *削除*を選択します。

AWS CLI

`my-cluster`をクラスタの名前に置き換えてから、次のコマンドを実行します。

```
aws eks delete-addon --cluster-name my-cluster --addon-name
netapp_trident-operator --preserve
```

eksctl

次のコマンドは、Trident EKS アドオンをアンインストールします：

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

ストレージクラスを設定する

https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass オブジェクト
"^]はプロビジョナーを識別し、プロビジョナーにボリュームのプロビジョニング方法を指示します。このセクションでは、Tridentをプロビジョナーとして指定するKubernetes StorageClassオブジェクトの構成方法を説明します。

StorageClassオブジェクトを作成します

FSx for ONTAP 用の StorageClass を作成すると、Trident はバックエンド構成を自動的に作成します。

メモ	ストレージバックエンドを手動で構成する場合は、[create-a-kubernetes-storageclass-without-automatic-backend-configuration]セクションを参照して、Tridentバックエンドとストレージクラスを個別に作成してください。

必須の**StorageClass**パラメータを指定する

StorageClassの作成時に定義する必要がある次の3つのパラメータ：

パラメータ	必須	タイプ	概要
`fsxFilesystemID`	はい	string	FSx for NetApp ONTAP ファイルシステム ID
`storageDriverName`	はい	string	Tridentストレージドライバ（例： `ontap-nas`または `ontap-san`）
`credentialsName`	はい	string	ONTAP クレデンシャルを含む FSx for ONTAP の Kubernetes Secret の名前

オプションのパラメータを指定します

StorageClassを通じてオプションのバックエンドパラメータを渡すことができます。StorageClass `parameters`セクションで、すべてのオプション値を文字列として定義します。バックエンドパラメータの完全なリストについては、以下を参照してください："FSx for NetApp ONTAP バックエンド構成"。

StorageClass設定ファイルの例。

次の例は、バックエンドの自動設定をトリガーする StorageClass を示しています。

YAML

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-fsx-demo
  annotations:
    description: "Demo StorageClass for FSx for NetApp ONTAP"
provisioner: csi.trident.netapp.io
parameters:
  fsxFilesystemID: "fs-0abc123"
  storageDriverName: "ontap-nas"
  credentialsName: trident-fsx-credentials
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

StorageClassを作成します

設定ファイルを作成したら、次のコマンドを実行してストレージクラスを作成します。

```
kubectl create -f storage-class-ontapnas.yaml
```

KubernetesとTridentの両方で*basic-csi*ストレージクラスが表示され、Tridentがバックエンドでプールを検出しているはずです。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

StorageClassを適用すると、Tridentは自動的にバックエンドを作成します。その後、このStorageClassを参照するPersistentVolumeClaimsを作成できます。

バックエンド構成ステータスを確認する

Tridentは、バックエンド作成の結果をStorageClassアノテーションに記録します。

注釈	概要
`trident.netapp.io/configuratorStatus`	設定結果(`Success`または `Failure`)
`trident.netapp.io/configuratorMessage`	詳細なステータスまたはエラーメッセージ
`trident.netapp.io/configuratorName`	内部コンフィギュレータリソースの名前
`trident.netapp.io/managed`	StorageClass が Trident によって管理されていることを示します
`trident.netapp.io/additionalStoragePools`	このバックエンド用に作成されたストレージプール

ステータスを確認するには、以下を実行してください：

```
kubectl get storageclass ontap-fsx-demo -o yaml
```

`trident.netapp.io/configuratorStatus`が
`Success`に設定されていることを確認してください。値が
`Failure`の場合は、エラーについて
`trident.netapp.io/configuratorMessage`を確認してください。

追加の**FSxN**ファイルシステムを追加する

同じ StorageClass を使い続けながら追加のストレージ容量が必要な場合は、追加の FSxN ファイルシステム ID を追加してください。

StorageClass を編集し、以下のアノテーションを追加します：

```
metadata:
  annotations:
    trident.netapp.io/additionalFsxNFileSystemID: '["fs-
xxxxxxxxxxxxxxxxxx"]'
```

変更を適用すると、Trident はバックエンド構成を更新し、StorageClass アノテーションを更新します。

運用上の考慮事項と制限

- 自動バックエンド構成を持つStorageClassを削除すると、通常、関連するTridentバックエンドが削除されます。これにより、ストレージ接続が阻害され、実行中のワークロードが中断される可能性があります。管理対象のStorageClassを削除する前に、その影響を検証してください。
- 自動バックエンド構成は、AWS FSx for NetApp ONTAP でのみサポートされています。

自動バックエンド構成なしで**Kubernetes StorageClass**を作成する

TridentバックエンドとStorageClassを別々に作成する場合は、次の手順に従ってください。

自動バックエンド構成の仕組みを理解する

Trident は、StorageClass の定義からバックエンド構成を取得します。StorageClass を適用すると、Trident は必要なパラメータを検証し、バックエンドを作成して、StorageClass にステータスのアノテーションを付与します。

TridentはVolumeSnapshotClassを1回だけ作成します。Tridentは、後続のStorageClassesに対して同じVolumeSnapshotClassを再利用します。

Tridentバックエンドを作成する

Tridentバックエンドを作成するには、JSON形式またはYAML形式で設定ファイルを作成する必要があります。ファイルには、使用するストレージの種類（NASまたはSAN）、ファイルシステム、取得元のSVM、およびその認証方法を指定する必要があります。次の例は、NASベースのストレージを定義し、AWSシークレットを使用して使用するSVMのクレデンシャルを保存する方法を示しています：

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

JSON

```
{
  "apiVersion": "trident.netapp.io/v1",
  "kind": "TridentBackendConfig",
  "metadata": {
    "name": "backend-tbc-ontap-nas"
    "namespace": "trident"
  },
  "spec": {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx"
    },
    "managementLIF": null,
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name",
      "type": "awsarn"
    }
  }
}
```

FSx for ONTAPドライバーの詳細

次のドライバを使用して、Trident を Amazon FSx for NetApp ONTAP と統合できます：

ドライバー名	概要
ontap-san	プロビジョニングされた各PVは、独自のAmazon FSx for NetApp ONTAPボリューム内のLUNです。ブロックストレージに推奨されます。
ontap-nas	プロビジョニングされた各PVは、完全なAmazon FSx for NetApp ONTAP ボリュームです。NFSおよびSMBに推奨されます。
ontap-san-economy	プロビジョニングされた各PVは、Amazon FSx for NetApp ONTAP ボリュームあたりの設定可能な数のLUNを持つLUNです。
ontap-nas-economy	プロビジョニングされた各PVはqtreeであり、Amazon FSx for NetApp ONTAPボリュームあたりのqtree数は設定可能です。
ontap-nas-flexgroup	プロビジョニングされた各PVは、完全なAmazon FSx for NetApp ONTAP FlexGroupボリュームです。

ドライバーの詳細については、"NASドライバー"および"SANドライバー"を参照してください。

バックエンドを作成する

設定ファイルを作成した後、以下のコマンドを実行して、Tridentバックエンド構成（TBC）を作成および検証します：

- yaml ファイルから Trident バックエンド構成（TBC）を作成し、次のコマンドを実行します：

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- Trident バックエンド構成（TBC）が正常に作成されたことを確認します：

```
Kubectl get tbc -n trident
```

```
NAME                           BACKEND NAME          BACKEND UUID
PHASE   STATUS

backend-tbc-ontap-nas          tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9   Bound   Success
```

その他の構成オプションの詳細については、以下の[Backend-advanced-configuration-and-examples]セクションを参照してください。

自動バックエンド構成*なし*でストレージクラスを構成する

以下は、TridentおよびFSx for ONTAPで使用するストレージクラス構成の例です。

NFS用ストレージクラス

この例を使用して、NFS を使用するボリュームの StorageClass を設定できます（属性の全リストについては、以下の Trident 属性セクションを参照してください）：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

iSCSI用ストレージクラス

この例を使用して、iSCSIを使用するボリュームのStorageClassを設定します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

NFSv3と**AWS Bottlerocket**を使用したストレージクラス

AWS BottlerocketでNFSv3ボリュームをプロビジョニングするには、必要な `mountOptions`をストレージクラスに追加します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

Trident StorageClass属性

これらのパラメータは、特定のタイプのボリュームをプロビジョニングするためにどのTrident管理ストレージプールを使用するかを決定します。

属性	タイプ	値	オファー	要求	サポート対象
メディア[1]	string	HDD、ハイブリッド、SSD	プールにはこのタイプのメディアが含まれます。ハイブリッドとは両方を意味します	指定されたメディアタイプ	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san
provisioningType	string	薄い、厚い	プールはこのプロビジョニング方法をサポートしています	プロビジョニング方法が指定されました	thick：すべてのONTAP、thin：すべての ONTAP および solidfire-san
backendType	string	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san、azure-netapp-files、ontap-san-economy	プールはこのタイプのバックエンドに属します	バックエンドが指定されました	すべてのドライバー
Snapshot	ブール値	true、false	プールは Snapshot 付きのボリュームをサポートします	スナップショットが有効になっているボリューム	ontap-nas、ontap-san、solidfire-san

属性	タイプ	値	オファー	要求	サポート対象
クローン	ブール値	true、false	プールはボリュームのクローン作成をサポート	クローンが有効なボリューム	ontap-nas、ontap-san、solidfire-san
暗号化	ブール値	true、false	プールは暗号化されたボリュームをサポートします	暗号化が有効になっているボリューム	ontap-nas、ontap-nas-economy、ontap-nas-flexgroups、ontap-san
IOPS	int	正の整数	プールはこの範囲の IOPS を保証できる	ボリュームで保証されるIOPS	solidfire-san

[1]：ONTAP SelectまたはFSx for ONTAPシステムではサポートされていません

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

ストレージクラスを作成します

StorageClassを設定したら、Kubernetesで作成できます。

手順

1. これはKubernetesオブジェクトなので、 `kubectl`を使用してKubernetesで作成します。

```
kubectl create -f storage-class-ontapnas.yaml
```

2. KubernetesとTridentの両方で*basic-csi*ストレージクラスが表示され、Tridentがバックエンドでプールを検出しているはずです。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

SMB ボリュームのプロビジョニング

SMB ボリュームは、 `ontap-nas`ドライバを使用してプロビジョニングできます。ただし、そのためには以下の手順を完了する必要があります："SMB ボリュームのプロビジョニングの準備"。

バックエンドの高度な構成と例

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	例
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + dataLIF
`managementLIF`	クラスタまたはSVM管理LIFのIPアドレス（完全修飾ドメイン名（FQDN）も指定可能）Trident が IPv6 フラグを使用してインストールされている場合、IPv6 アドレスを使用するように設定できます。IPv6 アドレスは角括弧で囲んで定義する必要があります（例：[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]）。もし `fsxFilesystemID`を `aws`フィールドの下で指定した場合、`managementLIF`を指定する必要はありません。なぜならTridentがAWSからSVM `managementLIF`情報を取得するためです。したがって、SVM配下のユーザー（例：vsadmin）の認証情報を必ず指定し、そのユーザーが `vsadmin`ロールを持っている必要があります。	"10.0.0.1"、"[2001:1234:abcd::fefe]"

パラメータ	概要	例
`dataLIF`	プロトコル LIF の IP アドレス。**ONTAP NAS** ドライバー：NetApp は dataLIF を指定することを推奨します。指定しない場合、Trident は SVM から dataLIF を取得します。NFS マウント操作に使用する完全修飾ドメイン名（FQDN）を指定することで、ラウンドロビン DNS を作成し、複数の dataLIF 間で負荷分散を行うことができます。初期設定後でも変更可能です。**ONTAP SAN** ドライバー：iSCSI には指定しないでください。Trident は ONTAP Selective LUN Map を使用して、マルチパスセッションの確立に必要な iSCSI LIF を検出します。dataLIF が明示的に定義されている場合、警告が生成されます。Trident が IPv6 フラグを使用してインストールされている場合、IPv6 アドレスを使用するように設定できます。IPv6 アドレスは角括弧で囲んで定義する必要があります（例：[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]）。	
`autoExportPolicy`	自動エクスポート ポリシーの作成と更新を有効にします [ブール値]。`autoExportPolicy`および`autoExportCIDRs`オプションを使用すると、Tridentはエクスポートポリシーを自動的に管理できます。	`false`
`autoExportCIDRs`	`autoExportPolicy`が有効になっている場合にKubernetesのノードIPをフィルタリングするためのCIDRのリスト。 `autoExportPolicy`および `autoExportCIDRs`オプションを使用すると、Tridentはエクスポート ポリシーを自動的に管理できます。	"["0.0.0.0/0", "::/0"]"
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	""
`clientCertificate`	クライアント証明書の Base64 エンコードされた値。証明書ベースの認証に使用	""
`clientPrivateKey`	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用	""

パラメータ	概要	例
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコードされた値。任意。証明書ベースの認証に使用されます。	""
`username`	クラスタまたは SVM に接続するためのユーザー名。クレデンシャルベースの認証に使用されます。たとえば、vsadmin。	
`password`	クラスターまたは SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
`svm`	使用するStorage Virtual Machine	SVM 管理 LIF が指定されている場合に派生されます。
`storagePrefix`	SVM で新しいボリュームをプロビジョニングするときに使用されるプレフィックス。作成後は変更できません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`
`limitAggregateUsage`	*Amazon FSx for NetApp ONTAP には指定しないでください。*提供された `fsxadmin`と `vsadmin`には、Trident を使用してアグリゲートの使用状況を取得して制限するために必要な権限が含まれていません。	使用しないでください。
`limitVolumeSize`	要求されたボリューム サイズがこの値を超える場合、プロビジョニングは失敗します。また、qtreeとLUNを管理するボリュームの最大サイズを制限し、`qtreesPerFlexvol`オプションにより、FlexVol volumeあたりのqtreeの最大数をカスタマイズできます	""（デフォルトでは強制されません）
`lunsPerFlexvol`	FlexVol volumeあたりの最大LUN数は、[50、200]の範囲にする必要があります。SANのみ。	“100”
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例：{"api":false, "method":true} `debugTraceFlags`を使用しないでください。ただし、トラブルシューティングを行っており、詳細なログ ダンプが必要な場合を除きます。	null

パラメータ	概要	例
nfsMountOptions	NFS マウント オプションのコンマ区切りリスト。Kubernetes 永続ボリュームのマウント オプションは通常ストレージ クラスで指定されますが、ストレージ クラスでマウント オプションが指定されていない場合、Trident はストレージ バックエンドの構成ファイルで指定されたマウント オプションを使用するようになります。ストレージ クラスまたは構成ファイルにマウント オプションが指定されていない場合、Trident は関連付けられている永続ボリュームにマウント オプションを設定しません。	""
nasType	NFS または SMB ボリュームの作成を設定します。オプションは nfs、 smb、または null です。*SMB ボリュームの場合は `smb`に設定する必要があります。*null に設定すると、デフォルトで NFS ボリュームになります。	nfs
qtreesPerFlexvol	FlexVol volume あたりの最大 qtree 数は、[50, 300] の範囲内である必要があります	"200"
smbShare	次のいずれかを指定できます：Microsoft 管理コンソールまたは ONTAP CLI を使用して作成された SMB 共有の名前、または Trident が SMB 共有を作成できるようにするための名前。このパラメータは、Amazon FSx for NetApp ONTAP バックエンドに必要です。	smb-share
useREST	ONTAP REST APIを使用するためのブーリアン パラメータ。 `true`に設定すると、TridentはONTAP REST APIを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、 `ontap`アプリケーションへのアクセス権が必要です。これは、事前定義された `vsadmin`および `cluster-admin`ロールで満たされます。	false

パラメータ	概要	例
aws	AWS FSx for ONTAP の設定ファイルでは以下を指定できます：fsxFilesystemID：AWS FSx ファイルシステムの ID を指定します。 apiRegion：AWS API リージョン名。 apikey：AWS API キー。 secretKey：AWS 秘密キー。	"" "" ""
credentials	AWS Secrets Manager に保存する FSx SVM 認証情報を指定します。- name：SVM の認証情報が含まれるシークレットの Amazon リソースネーム（ARN）。- type：`awsarn`に設定します。詳細については、"AWS Secrets Manager シークレットを作成する" を参照してください。	

ボリュームのプロビジョニング用のバックエンド設定オプション

デフォルトのプロビジョニングは、設定の `defaults`セクションにあるこれらのオプションを使用して制御できます。例については、以下の設定例を参照してください。

パラメータ	概要	デフォルト
spaceAllocation	LUNのスペース割り当て	true
spaceReserve	スペース予約モード：「none」（シン）または「volume」（シック）	none
snapshotPolicy	使用するSnapshotポリシー	none
qosPolicy	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージ プールまたはバックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。Trident で QoS ポリシー グループを使用するには、ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各構成要素に個別に適用されるようにする必要があります。共有 QoS ポリシー グループは、すべてのワークロードの合計スループットの上限を適用します。	""

パラメータ	概要	デフォルト
adaptiveQosPolicy	作成されたボリュームに割り当てるアダプティブ QoS ポリシー グループ。ストレージ プールまたはバックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。ontap-nas-economy ではサポートされていません。	""
snapshotReserve	スナップショット用に予約されているボリュームの割合「0」	もし `snapshotPolicy`が `none`の場合、 `else` ""
splitOnClone	作成時にクローンを親から分離する	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたボリュームはすべてNAEが有効になります。詳細については、次を参照してください："Tridentと NVE および NAE の連携"。	false
luksEncryption	LUKS暗号化を有効にします。"Linux Unified Key Setup（LUKS）を使用する"を参照してください。SANのみ。	""
tieringPolicy	使用する階層化ポリシー none	
unixPermissions	新しいボリュームのモード。**SMB**ボリュームの場合は空白のままにします。	""
securityStyle	新しいボリュームのセキュリティ スタイル。NFSは `mixed`および `unix`セキュリティ スタイルをサポートします。SMBは `mixed`および `ntfs`セキュリティ スタイルをサポートします。	NFSのデフォルトは `unix`です。SMBのデフォルトは `ntfs`です。

PVCの設定

このセクションでは、設定されたKubernetes StorageClassを使用してPVを要求するPersistentVolumeClaim（PVC）を作成する方法について説明します。成功すると、PVをpodにマウントできます。

PVC を作成する

A "*PersistentVolumeClaim*"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。PVC は、特定のサイズまたはアクセス モードのストレージを要求するように構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

TridentバックエンドとStorageClassを作成したら、PVCを作成できます。PVCを作成したら、そのボリュームをポッドにマウントできます。

サンプルマニフェスト

以下の例は、基本的なPVC構成オプションを示しています。

RWX アクセス付き **PVC**

この例では、RWXアクセスを持つ基本的なPVCが、 `basic-csi`という名前のStorageClassに関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-gold
```

iSCSI を使用した **PVC** の例

この例では、RWOアクセスを持つiSCSI用の基本PVCが、StorageClassという名前 `protection-gold`に関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: protection-gold
```

PVCを作成する

手順

1. PVC を作成します。

```
kubectl create -f pvc.yaml
```

2. PVC ステータスを確認します。

```
kubectl get pvc
```

```
NAME          STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                           5m
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

アプリケーションをデプロイする

ストレージ クラスと PVC が作成されると、PV をポッドにマウントできます。このセクションでは、PV をポッドに接続するためのコマンドと構成の例を示します。

サンプルアプリケーションを導入する

手順

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

以下の例は、PVC をポッドに接続するための基本構成を示しています。基本構成：

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

メモ　進捗状況は `kubectl get pod --watch`で確認できます。

2. ボリュームが `/my/mount/path`にマウントされていることを確認します。

```
kubectl exec -it pv-pod -- df -h /my/mount/path
```

```
Filesystem                                                      Size
Used  Avail Use%  Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1%   /my/mount/path
```

これでPodを削除できます。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

EKS クラスター上の **Trident EKS** アドオンを設定する

NetApp Tridentは、Amazon FSx for NetApp ONTAPのKubernetesストレージ管理を合理化し、開発者と管理者がアプリケーションの導入に集中できるようにします。NetApp Trident EKSアドオンには、最新のセキュリティパッチとバグ修正が含まれており、Amazon EKSで動作することがAWSによって検証されています。EKS アドオンを使用すると、Amazon EKS クラスターの安全性と安定性を常に確保し、アドオンのインス

トール、設定、更新に必要な作業量を削減できます。

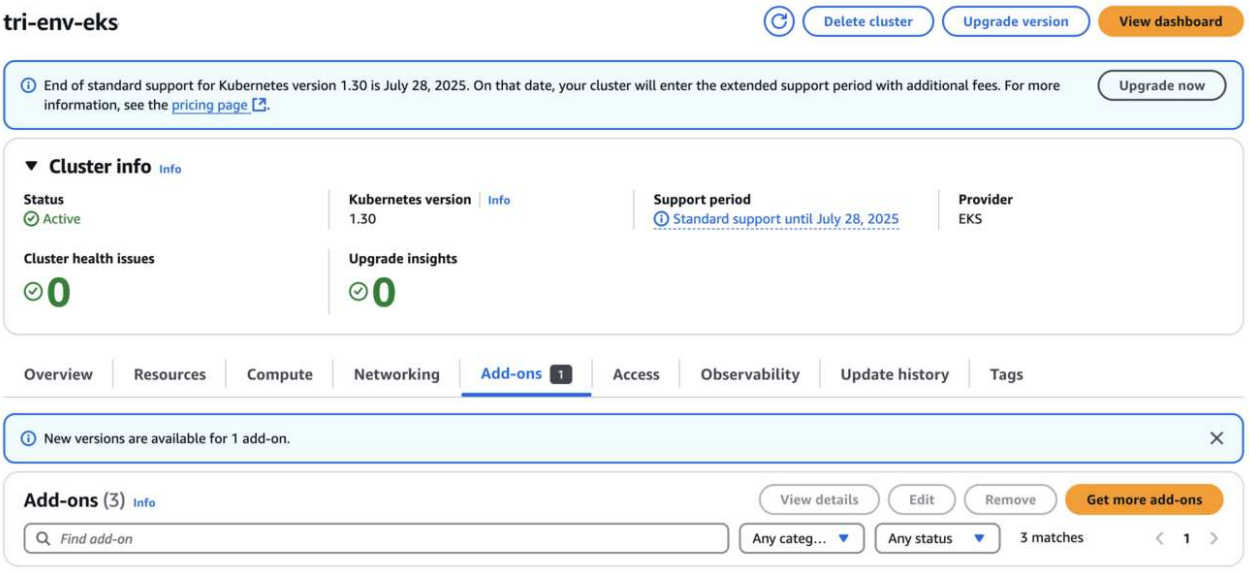
前提条件

AWS EKS の Trident アドオンを設定する前に、以下のものを用意してください：

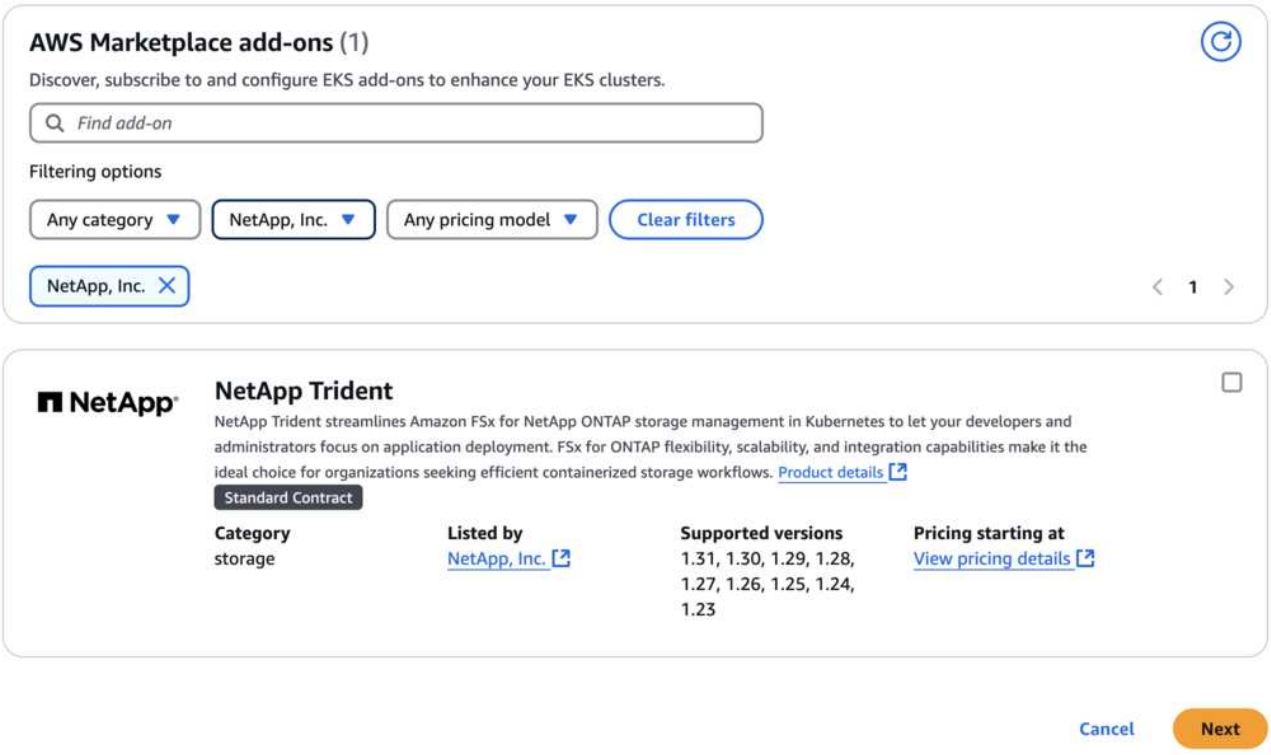
- アドオンを操作する権限を持つ Amazon EKS クラスターアカウント。"Amazon EKS アドオン"を参照してください。
- AWS マーケットプレイスへの AWS 権限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMI タイプ：Amazon Linux 2（AL2_x86_64）または Amazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存の Amazon FSx for NetApp ONTAP ファイルシステム

手順

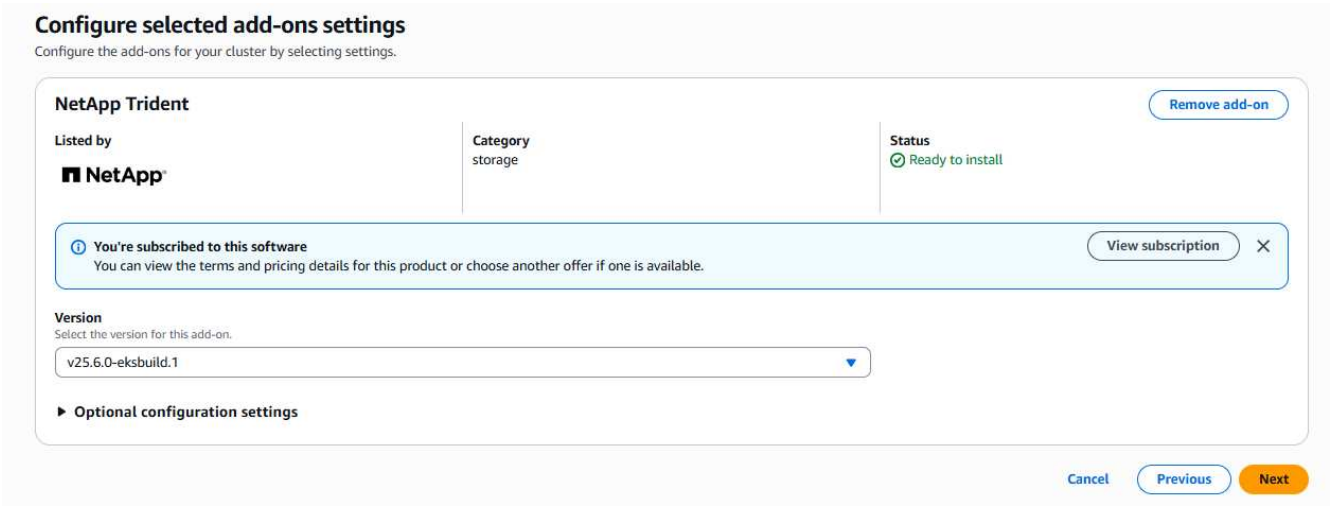
1. EKS ポッドが AWS リソースにアクセスできるようにするには、IAM ロールと AWS シークレットを作成してください。手順については、"IAMロールとAWSシークレットを作成する"を参照してください。
2. EKS Kubernetes クラスターで、 * アドオン * タブに移動します。



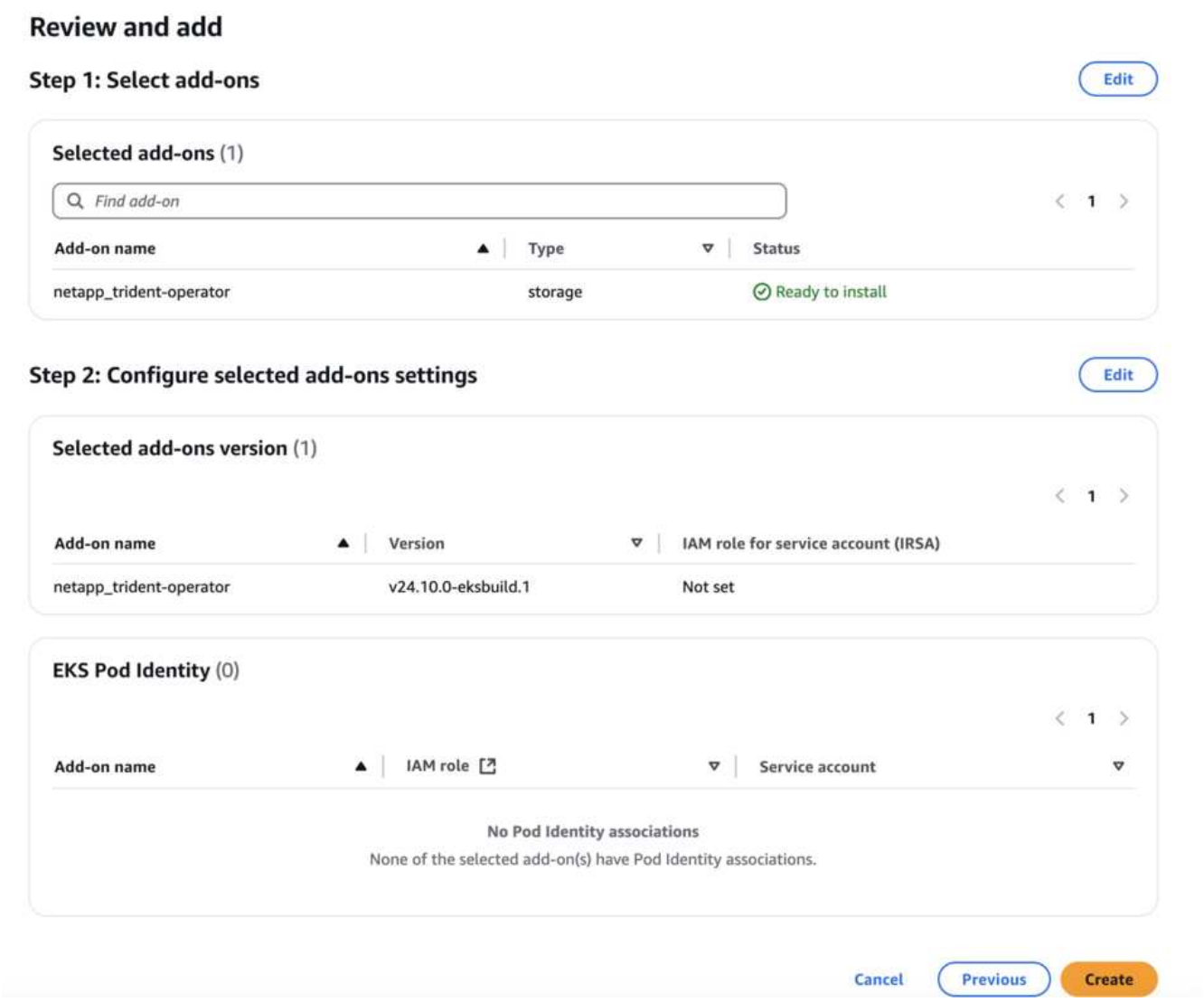
3. **AWS Marketplace** アドオン に移動し、*storage* カテゴリを選択します。



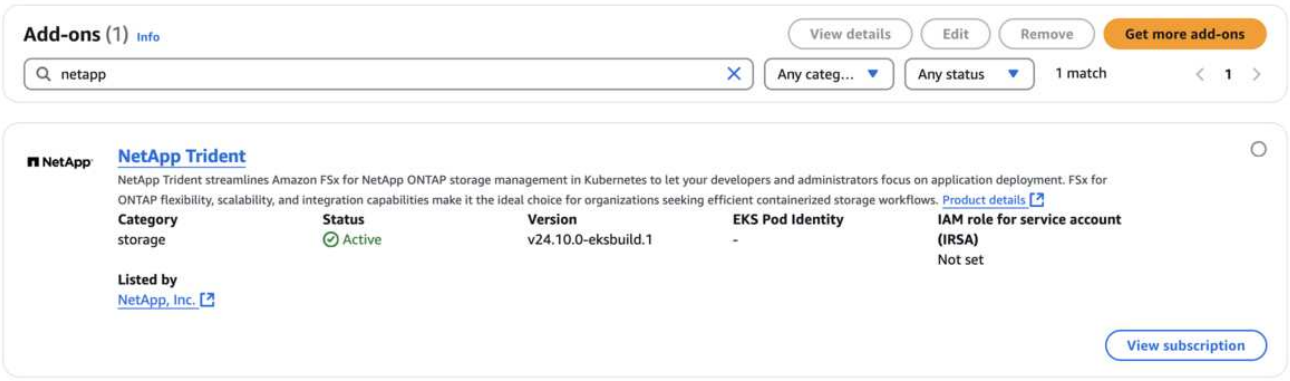
4. **NetApp Trident** を見つけて、Trident アドオンのチェックボックスをオンにし、次へ をクリックします。
5. アドオンの目的のバージョンを選択します。



6. 必要なアドオン設定を構成します。



7. IRSA（サービスアカウントのIAMロール）を使用している場合は、追加の構成手順を参照してください"ここをクリックしてください。"。
8. *Create*を選択します。
9. アドオンのステータスが *Active* であることを確認します。



10. 次のコマンドを実行して、Trident がクラスタに正しくインストールされていることを確認します：

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

11. セットアップを続行し、ストレージ バックエンドを構成します。詳細については、"ストレージバックエンドを設定する"を参照してください。

CLI を使用した Trident EKS アドオンのインストール / アンインストール

CLI を使用して **NetApp Trident EKS** アドオンをインストールします：

次のコマンド例では、Trident EKS アドオンをインストールします：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.0-eksbuild.1（専用バージョンを使用）
```

以下のコマンド例は Trident EKS アドオンバージョン 25.6.1 をインストールします：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.1-eksbuild.1（専用バージョンを使用）
```

以下のコマンド例は Trident EKS アドオンバージョン 25.6.2 をインストールします：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.2-eksbuild.1（専用バージョンを使用）
```

CLI を使用して **NetApp Trident EKS** アドオンをアンインストールします：

次のコマンドは、Trident EKS アドオンをアンインストールします：

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

kubectl でバックエンドを作成する

バックエンドは、Trident とストレージ システム間の関係を定義します。Trident がそのストレージ システムと通信する方法と、Trident がそこからボリュームをプロビジョニングする方法を指定します。Trident のインストール後、次のステップはバックエンドを作成することです。 `TridentBackendConfig` カスタム リソース定義（CRD）を使用すると、Kubernetes インターフェースを介して直接 Trident バックエンドを作成および管理できます。これは、 `kubectl` または Kubernetes ディストリビューションに相当する CLI ツールを使用して実行できます。

`TridentBackendConfig`

`TridentBackendConfig`（`tbc`、 `tbconfig`、 `tbackendconfig`）は、フロントエンドの名前空間CRDであり、 `kubectl`を使用して`Trident`バックエンドを管理できます。`Kubernetes`およびストレージ管理者は、専用のコマンドラインユーティリティ（`tridentctl`）を必要とせずに、Kubernetes CLIを介して直接バックエンドを作成および管理できるようになりました。

`TridentBackendConfig`オブジェクトの作成時に、次のようになります：

- バックエンドは、提供された設定に基づいて Trident によって自動的に作成されます。これは内部的には `TridentBackend`（tbe、 `tridentbackend`）CR として表されます。
- `TridentBackendConfig`は、Tridentによって作成された `TridentBackend`に一意にバインドされています。

各 `TridentBackendConfig`は、 `TridentBackend`との1対1のマッピングを維持します。前者は、バックエンドを設計および構成するためにユーザーに提供されるインターフェースであり、後者は Trident が実際のバックエンド オブジェクトを表す方法です。

警告	`TridentBackend` CRはTridentによって自動的に作成されます。これらを変更*しないでください*。バックエンドを更新する場合は、 `TridentBackendConfig`オブジェクトを変更してください。

`TridentBackendConfig` CRのフォーマットについては、次の例を参照してください：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

また、 "trident-installer"ディレクトリの例で、必要なストレージ プラットフォーム/サービスのサンプル構成を確認することもできます。

`spec`は、バックエンド固有の構成パラメータを受け取ります。この例では、バックエンドは `ontap-san`ストレージ
ドライバを使用し、ここに表されている構成パラメータを使用します。目的のストレージ
ドライバの構成オプションのリストについては、`link:backends.html["ストレージ
ドライバのバックエンド設定情報"^]`を参照してください。

この `spec`セクションには、 `credentials`や `deletionPolicy`フィールドも含まれており、これらは `TridentBackendConfig`CRで新たに導入されました：

- `credentials`：このパラメータは必須フィールドであり、ストレージ システム/サービスでの認証に使用される資格情報が含まれます。これは、ユーザーが作成した Kubernetes シークレットに設定されます。資格情報はプレーンテキストで渡すことができず、エラーが発生します。

- `deletionPolicy`：このフィールドは、 `TridentBackendConfig`が削除されたときに何が起こるかを定義します。次の2つの値のいずれかを取ることができます：
  - `delete`：これにより、 `TridentBackendConfig` CR と関連するバックエンドの両方が削除されます。これがデフォルト値です。
  - `retain`： `TridentBackendConfig` CR が削除されても、バックエンドの定義はそのまま残り、 `tridentctl`で管理できます。削除ポリシーを `retain`に設定すると、ユーザーは以前のリリース（21.04より前）にダウングレードし、作成されたバックエンドを保持できます。このフィールドの値は、 `TridentBackendConfig`の作成後に更新できます。

メモ	バックエンドの名前は `spec.backendName`を使用して設定されます。指定しない場合、バックエンドの名前は `TridentBackendConfig`オブジェクト（metadata.name）の名前に設定されます。 `spec.backendName`を使用してバックエンド名を明示的に設定することを推奨します。
ヒント	`tridentctl`で作成されたバックエンドには、関連付けられた `TridentBackendConfig`オブジェクトがありません。そのようなバックエンドは、 `kubectl`で `TridentBackendConfig`CRを作成することで管理することができます。同一の構成パラメータ（例えば、 `spec.backendName`、 `spec.storagePrefix`、 `spec.storageDriverName`など）を指定する必要があるため、注意してください。Tridentは、新しく作成された `TridentBackendConfig`を既存のバックエンドに自動的にバインドします。

手順の概要

`kubectl`を使用して新しいバックエンドを作成するには、次の操作を行う必要があります：

1. "Kubernetes Secret"を作成します。シークレットには、Trident がストレージ クラスタ / サービスと通信するために必要なクレデンシャルが含まれています。
2. `TridentBackendConfig`オブジェクトを作成します。これには、ストレージクラスタ/サービスに関する詳細が含まれており、前の手順で作成されたシークレットが参照されます。

バックエンドを作成したら、 `kubectl get tbc <tbc-name> -n <trident-namespace>`を使用してそのステータスを確認し、追加の詳細情報を収集できます。

ステップ**1**：**Kubernetes**シークレットを作成する

バックエンドのアクセス資格情報を含むシークレットを作成します。これは各ストレージ サービス/プラットフォームに固有です。次に例を示します：

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: password
```

この表は、各ストレージ プラットフォームの Secret に含める必要があるフィールドをまとめたものです：

ストレージ プラットフォームの **Secret Fields** の説明	シークレット	フィールドの説明
Azure NetApp Files	clientID	アプリ登録からのclient ID
Element（NetApp HCI/SolidFire）	エンドポイント	テナント資格情報を持つSolidFire クラスターのMVIP
ONTAP	ユーザ名	クラスタ/SVM に接続するためのユーザー名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスター/SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用されます
ONTAP	chapUsername	受信ユーザー名。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合
ONTAP	chapTargetUsername	ターゲットユーザー名。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合

ストレージ プラットフォームの **Secret Fields** の説明	シークレット	フィールドの説明
ONTAP	chapTargetInitiatorSecret	CHAP ターゲット イニシエータ シークレット。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合

このステップで作成されたSecretは、次のステップで作成される `TridentBackendConfig`オブジェクトの `spec.credentials`フィールドで参照されます。

ステップ**2**： `TridentBackendConfig` **CR**を作成する

これで、 `TridentBackendConfig` CRを作成する準備ができました。この例では、 `ontap-san`ドライバを使用するバックエンドが、以下に示す `TridentBackendConfig`オブジェクトを使用して作成されます：

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

ステップ**3**： `TridentBackendConfig` **CR**のステータスを確認する

これで、 `TridentBackendConfig` CRを作成したので、ステータスを確認できます。次の例を参照してください：

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                       BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

バックエンドが正常に作成され、 `TridentBackendConfig` CRにバインドされました。

フェーズには次のいずれかの値を指定できます：

- `Bound`： `TridentBackendConfig` CR はバックエンドに関連付けられており、そのバックエンドには `configRef`が `TridentBackendConfig` CR の uid に設定されています。
- `Unbound`： `""`を使用して表されます。 `TridentBackendConfig`オブジェクトはバックエンドにバインドされていません。新しく作成されたすべての `TridentBackendConfig` CR は、デフォルトでこのフェーズにあります。フェーズが変更された後は、再び Unbound に戻ることはできません。
- `Deleting`： `TridentBackendConfig` CR の `deletionPolicy`削除するように設定されました。 `TridentBackendConfig` CR が削除されると、削除中状態に移行します。
  - バックエンドに永続ボリュームクレーム（PVC）が存在しない場合は、 `TridentBackendConfig`を削除すると、`Trident`はバックエンドと `TridentBackendConfig` CRも削除します。
  - バックエンドに 1 つ以上の PVC が存在する場合、削除状態になります。 `TridentBackendConfig` CR もその後削除フェーズに入ります。バックエンドと `TridentBackendConfig`は、すべての PVC が削除された後にのみ削除されます。
- `Lost`： `TridentBackendConfig` CR に関連付けられたバックエンドが誤ってまたは故意に削除され、 `TridentBackendConfig` CR には削除されたバックエンドへの参照がまだ残っています。 `TridentBackendConfig` CR は、 `deletionPolicy`の値に関係なく削除できます。
- `Unknown`：Tridentは `TridentBackendConfig` CRに関連付けられているバックエンドの状態または存在を判断できません。たとえば、APIサーバーが応答しない場合や、 `tridentbackends.trident.netapp.io` CRDが存在しない場合などです。この場合、介入が必要になる可能性があります。

この段階で、バックエンドが正常に作成されました。追加で処理できる操作がいくつかあります。"バックエンドの更新とバックエンドの削除"

（オプション）ステップ**4**：詳細を取得する

バックエンドの詳細情報を取得するには、次のコマンドを実行します。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

```
NAME                      BACKEND NAME         BACKEND UUID
PHASE    STATUS     STORAGE DRIVER    DELETION POLICY
backend-tbc-ontap-san    ontap-san-backend    8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8    Bound    Success    ontap-san         delete
```

さらに、YAML/JSONダンプを取得することもできます `TridentBackendConfig`。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

backendInfo には、 backendName と backendUUID が含まれており、これは TridentBackendConfig CR に応じて作成されたバックエンドのものです。 lastOperationStatus フィールドは、 TridentBackendConfig CR の直近の操作のステータスを表しており、これはユーザーによってトリガーされる場合（例：ユーザーが spec で何かを変更した場合）や、Trident によってトリガーされる場合（例：Trident の再起動時）があります。これは Success または Failed のいずれかになります。 phase は、 TridentBackendConfig CR とバックエンドとの関係のステータスを表します。上記の例では、 phase の値は Bound となっており、これは TridentBackendConfig CR がバックエンドと関連付けられていることを意味します。

`kubectl -n trident describe tbc <tbc-cr-name>`コマンドを実行すると、イベントログの詳細を取得できます。

警告	関連付けられた `TridentBackendConfig`オブジェクトを含むバックエンドを `tridentctl`を使用して更新または削除することはできません。 `tridentctl`と `TridentBackendConfig`の切り替え手順を理解するには、"こちらをご覧ください"を参照してください。

バックエンドを管理する

kubectl を使用してバックエンド管理を実行する

`kubectl`を使用してバックエンド管理操作を実行する方法について説明します。

バックエンドを削除する

`TridentBackendConfig`を削除すると、Tridentに対してバックエンドを削除/保持するように指示します（ `deletionPolicy`に基づく）。バックエンドを削除するには、`deletionPolicy`がdeleteに設定されていることを確認してください。`TridentBackendConfig`だけを削除するには、 `deletionPolicy`がretainに設定されていることを確認してください。これにより、バックエンドが引き続き存在し、`tridentctl`を使用して管理できることが保証されます。

次のコマンドを実行します。

```
kubectl delete tbc <tbc-name> -n trident
```

Tridentは`TridentBackendConfig`で使用されていたKubernetes Secretを削除しません。Kubernetesユーザーはシークレットをクリーンアップする責任があります。シークレットを削除するときは注意が必要です。シークレットは、バックエンドで使用されていない場合にのみ削除する必要があります。

既存のバックエンドを表示する

次のコマンドを実行します。

```
kubectl get tbc -n trident
```

`tridentctl get backend -n trident`または `tridentctl get backend -o yaml -n trident`を実行して、存在するすべてのバックエンドのリストを取得することもできます。このリストには、 `tridentctl`で作成されたバックエンドも含まれます。

バックエンドを更新する

バックエンドを更新する理由は複数考えられます：

- ストレージシステムへのクレデンシャルが変更されました。クレデンシャルを更新するには、`TridentBackendConfig`オブジェクトで使用されているKubernetes Secretを更新する必要があります。Tridentは、提供された最新のクレデンシャルを使用してバックエンドを自動的に更新します。次のコマンドを実行してKubernetes Secretを更新します：

```
kubectl apply -f <updated-secret-file.yaml> -n trident
```

- パラメータ（使用されている ONTAP SVM の名前など）を更新する必要があります。
  - `TridentBackendConfig`オブジェクトは、次のコマンドを使用してKubernetes経由で直接更新できます：

    ```
    kubectl apply -f <updated-backend-file.yaml>
    ```

  - あるいは、次のコマンドを使用して既存の `TridentBackendConfig` CR に変更を加えることができます：

    ```
    kubectl edit tbc <tbc-name> -n trident
    ```

メモ

- バックエンドの更新が失敗した場合、バックエンドは最後の既知の構成のままになります。ログを表示して原因を特定するには、 `kubectl get tbc <tbc-name> -o yaml -n trident`または `kubectl describe tbc <tbc-name> -n trident`を実行します。
- 構成ファイルの問題を特定して修正したら、更新コマンドを再実行できます。

tridentctl でバックエンド管理を実行する

`tridentctl`を使用してバックエンド管理操作を実行する方法について説明します。

バックエンドを作成する

"バックエンド設定ファイル"を作成したら、次のコマンドを実行します：

```
tridentctl create backend -f <backend-file> -n trident
```

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs -n trident
```

設定ファイルの問題を特定して修正したら、 `create`コマンドを再度実行します。

バックエンドを削除する

バックエンドをTridentから削除するには、次の操作を行います：

1. バックエンド名を取得します：

```
tridentctl get backend -n trident
```

2. バックエンドを削除します：

```
tridentctl delete backend <backend-name> -n trident
```

メモ	Tridentがこのバックエンドからプロビジョニングしたボリュームとスナップショットがまだ存在する場合、バックエンドを削除すると、新しいボリュームをプロビジョニングできなくなります。バックエンドは「削除中」の状態で存在し続けます。

既存のバックエンドを表示する

Tridentが認識しているバックエンドを表示するには、次の操作を行います。

- 概要を取得するには、次のコマンドを実行します：

```
tridentctl get backend -n trident
```

- すべての詳細を取得するには、次のコマンドを実行します：

```
tridentctl get backend -o json -n trident
```

バックエンドを更新する

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します：

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

バックエンドの更新が失敗した場合は、バックエンドの構成に問題があるか、無効な更新を試行しました。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs -n trident
```

設定ファイルの問題を特定して修正したら、 `update`コマンドを再度実行します。

バックエンドを使用するストレージクラスを特定する

これは、 `tridentctl`がバックエンドオブジェクトに対して出力するJSONで回答できる質問の種類の例です。これは `jq`ユーティリティを使用しており、インストールする必要があります。

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

これは、 `TridentBackendConfig`を使用して作成されたバックエンドにも適用されます。

バックエンド管理オプション間を移動する

Trident でバックエンドを管理するさまざまな方法について説明します。

バックエンドを管理するオプション

`TridentBackendConfig`の導入により、管理者は、バックエンドを管理する 2 つの独自の方法を利用できるようになりました。これにより、次の疑問が生じます：

- `tridentctl`を使用して作成されたバックエンドは、 `TridentBackendConfig`で管理できますか？
- `TridentBackendConfig`を使用して作成されたバックエンドは、 `tridentctl`を使用して管理できますか？

`tridentctl`を使用してバックエンドを管理 `TridentBackendConfig`

このセクションでは、Kubernetesインターフェイスを通じて `tridentctl`を直接作成し、 `TridentBackendConfig`オブジェクトを作成することで作成されたバックエンドの管理手順について説明します。

これは次のシナリオに適用されます：

- 既存のバックエンドには `TridentBackendConfig`がありません。これは `tridentctl`で作成されたためです。
- `tridentctl`で作成された新しいバックエンド（他の `TridentBackendConfig`オブジェクトが存在する場合）。

どちらのシナリオでも、バックエンドは存在し続け、Tridentがボリュームのスケジュール設定とボリュームに対する操作を行います。管理者には次の2つの選択肢があります（：）

- それを使用して作成されたバックエンドの管理には `tridentctl`を引き続きご利用ください。
- `tridentctl`を使用して作成されたバックエンドを新しい `TridentBackendConfig`オブジェクトにバインドします。これにより、バックエンドは `kubectl`で管理され、 `tridentctl`では管理されなくなります。

  `kubectl`を使用して既存のバックエンドを管理するには、既存のバックエンドにバインドする `TridentBackendConfig`を作成する必要があります。その仕組みの概要は次のとおりです：

  1. Kubernetes シークレットを作成します。このシークレットには、Trident がストレージクラスタ/サービスと通信するために必要なクレデンシャルが含まれています。
  2. `TridentBackendConfig`オブジェクトを作成します。これには、ストレージクラスタ/サービスに関する詳細が含まれており、前の手順で作成されたシークレットが参照されます。同一の設定パラメ

ータ（ `spec.backendName`、 `spec.storagePrefix`、 `spec.storageDriverName`など）を指定するように注意する必要があります。 `spec.backendName`は既存のバックエンドの名前に設定する必要があります。

ステップ**0**：バックエンドを特定する

既存のバックエンドにバインドする `TridentBackendConfig`を作成するには、バックエンドの構成を取得する必要があります。この例では、次の JSON 定義を使用してバックエンドが作成されたと仮定します：

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

ステップ1：Kubernetesシークレットを作成する

次の例に示すように、バックエンドの資格情報を含む Secret を作成します：

```
cat tbc-ontap-nas-backend-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password
```

```
kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

ステップ2： `TridentBackendConfig` CRを作成する

次のステップは、既存の `ontap-nas-backend`に自動的にバインドされる `TridentBackendConfig` CR を作成することです（この例のように）。次の要件が満たされていることを確認してください：

- 同じバックエンド名が `spec.backendName`で定義されています。
- 構成パラメータは元のバックエンドと同一です。
- 仮想プール（存在する場合）は、元のバックエンドと同じ順序を維持する必要があります。
- クレデンシャルはプレーンテキストではなく、Kubernetes Secret を通じて提供されます。

この場合、 `TridentBackendConfig`は次のようになります：

```
cat backend-tbc-ontap-nas.yaml
```

```yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

ステップ3： `TridentBackendConfig` CRのステータスを確認する

`TridentBackendConfig`が作成された後、そのフェーズは`Bound`である必要があります。また、既存のバックエンドと同じバックエンド名とUUIDを反映する必要があります。

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

バックエンドは、 tbc-ontap-nas-backend `TridentBackendConfig`オブジェクトを使用して完全に管理されるようになります。

`TridentBackendConfig`を使用してバックエンドを管理 `tridentctl`

`tridentctl`は、`TridentBackendConfig`を使用して作成されたバックエンドを一覧表示するために使用できます。さらに、管理者は`tridentctl`を通じてそのようなバックエンドを完全に管理することもでき、`TridentBackendConfig`を削除し、 `spec.deletionPolicy`が`retain`に設定されていることを確認できます。

ステップ**0**：バックエンドを特定する

例えば、次のバックエンドが `TridentBackendConfig`を使用して作成されたとします：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

出力から、`TridentBackendConfig`が正常に作成され、バックエンドにバインドされていることがわかります［バックエンドの UUID を確認してください］。

ステップ**1**： **`deletionPolicy`**が**`retain`**に設定されていることを確認します

`deletionPolicy`の値を見てみましょう。これを`retain`に設定する必要があります。これにより、 `TridentBackendConfig` CRが削除されても、バックエンドの定義はそのまま残り、 `tridentctl`で管理できます。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

メモ `deletionPolicy`が `retain`に設定されていない限り、次のステップに進まないでください。

ステップ**2**： `TridentBackendConfig` **CR**を削除する

最後のステップは、 `TridentBackendConfig` CRを削除することです。 `deletionPolicy`が `retain`に設定されていることを確認した後、削除を進めることができます：

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

`TridentBackendConfig`オブジェクトを削除すると、Tridentはバックエンド自体を実際に削除せずに、単に削除するだけです。

ストレージクラスの作成と管理

ストレージクラスを作成する

KubernetesのStorageClassオブジェクトを設定し、ストレージクラスを作成してTridentにボリュームのプロビジョニング方法を指示します。

Kubernetes StorageClassオブジェクトを設定する

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass オブジェクト"^]`は、そのクラスで使用されるプロビジョナーとして`Trident`を識別し、`Trident`にボリュームのプロビジョニング方法を指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

ストレージクラスを作成する

StorageClassオブジェクトを作成したら、ストレージクラスを作成できます。[ストレージクラスのサンプル]には、使用または変更できる基本的なサンプルがいくつか用意されています。

手順

1. これはKubernetesオブジェクトなので、 `kubectl`を使用してKubernetesで作成します。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

2. KubernetesとTridentの両方で*basic-csi*ストレージクラスが表示され、Tridentがバックエンドでプールを検出しているはずです。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

ストレージクラスのサンプル

Tridentは "特定のバックエンド向けのシンプルなストレージクラス定義"を提供します。

あるいは、インストーラーに付属の `sample-input/storage-class-csi.yaml.templ`ファイルを編集し、`*BACKEND_TYPE*`をストレージドライバー名に置き換えることもできます。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

ストレージクラスの管理

既存のストレージクラスを表示したり、デフォルトのストレージクラスを設定したり、ストレージクラスのバックエンドを識別したり、ストレージクラスを削除したりできます。

既存のストレージクラスを表示する

- 既存の Kubernetes ストレージ クラスを表示するには、次のコマンドを実行します：

```
kubectl get storageclass
```

- Kubernetes ストレージ クラスの詳細を表示するには、次のコマンドを実行します：

```
kubectl get storageclass <storage-class> -o json
```

- Tridentの同期されたストレージクラスを表示するには、次のコマンドを実行します：

```
tridentctl get storageclass
```

- Tridentの同期されたストレージクラスの詳細を表示するには、次のコマンドを実行します：

```
tridentctl get storageclass <storage-class> -o json
```

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されました。これは、ユーザーが Persistent Volume Claim（PVC）でストレージクラスを指定しない場合に、Persistent Volume をプロビジョニングするために使用されるストレージクラスです。

- ストレージクラス定義でアノテーション `storageclass.kubernetes.io/is-default-class`を true に設定して、デフォルトのストレージクラスを定義します。仕様によれば、その他の値またはアノテーションの欠如は false として解釈されます。
- 次のコマンドを使用して、既存のストレージ クラスをデフォルトのストレージ クラスとして構成できます：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 同様に、次のコマンドを使用して、デフォルトのストレージ クラス注釈を削除できます：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、この注釈を含む Trident インストーラー バンドルの例もあります。

メモ	クラスター内に一度に存在できるデフォルトのストレージ クラスは 1 つだけです。Kubernetes では、技術的には複数のストレージ クラスを持つことを禁止していませんが、デフォルトのストレージ クラスがまったく存在しないかのように動作します。

ストレージクラスのバックエンドを識別する

これは、 `tridentctl`がTridentバックエンド オブジェクトに対して出力するJSONで答えられる質問の種類の例です。これは `jq`ユーティリティを使用しますが、最初にインストールする必要がある場合があります。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ストレージ クラスを削除する

Kubernetes からストレージ クラスを削除するには、次のコマンドを実行します：

```
kubectl delete storageclass <storage-class>
```

`<storage-class>`は、ストレージ クラスに置き換える必要があります。

このストレージクラスで作成された永続ボリュームはそのまま残り、Tridentで引き続き管理されます。

メモ	Tridentは、作成するボリュームに対して空白 `fsType`を強制します。iSCSIバックエンドの場合、StorageClassで `parameters.fsType`を強制することが推奨されます。既存のStorageClassesを削除し、 `parameters.fsType`を指定して再作成する必要があります。

ボリュームのプロビジョニングと管理

ボリュームをプロビジョニングする

設定済みのKubernetes StorageClass を使用して、PVへのアクセスを要求するPersistentVolumeClaim（PVC）を作成します。その後、PVをポッドにマウントできます。

概要

A "*PersistentVolumeClaim*"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVC は、特定のサイズまたはアクセス モードのストレージを要求するように構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

PVC を作成したら、ボリュームをポッドにマウントできます。

PVC を作成する

手順

1. PVC を作成します。

```
kubectl create -f pvc.yaml
```

2. PVC ステータスを確認します。

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS AGE
pvc-storage   Bound    pv-name    1Gi        RWO                    5m
```

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

メモ　進捗状況は `kubectl get pod --watch`で確認できます。

2. ボリュームが `/my/mount/path`にマウントされていることを確認します。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. これでPodを削除できます。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

サンプルマニフェスト

PersistentVolumeClaim サンプルマニフェスト

これらの例は、基本的な PVC 構成オプションを示しています。

RWO アクセス付き PVC

この例では、RWOアクセスを持つ基本的なPVCが、 `basic-csi`という名前のStorageClassに関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe/TCP を使用した PVC

この例では、RWOアクセスを持つNVMe/TCPの基本的なPVCが、StorageClassという名前 `protection-gold`に関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Podマニフェストのサンプル

これらの例は、PVCをポッドに接続するための基本的な設定を示しています。

基本設定

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

基本的な**NVMe/TCP**構成

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確

認してください。

ボリュームを拡張する

Trident は、Kubernetes ユーザーにボリューム作成後にボリュームを拡張する機能を提供します。iSCSI、NFS、SMB、NVMe/TCP、および FC ボリュームを拡張するために必要な構成に関する情報を見つけます。

iSCSI ボリュームを拡張する

CSI プロビジョナーを使用して iSCSI 永続ボリューム（PV）を拡張できます。

メモ	iSCSIボリューム拡張は、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`ドライバでサポートされており、Kubernetes 1.16以降が必要です。

ステップ**1**：ボリューム拡張をサポートするように**StorageClass**を設定する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドを `true`に設定します。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存のStorageClassの場合は、編集して `allowVolumeExpansion`パラメータを含めます。

手順**2**：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、 `spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順3：**PVC**を接続するポッドを定義する

サイズを変更するには、PVをポッドに接続します。iSCSI PV のサイズを変更する場合、次の 2 つのシナリオがあります：

- PVがポッドに接続されている場合、Tridentはストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
- アタッチされていないPVのサイズを変更しようとすると、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、Tridentはデバイスを再スキャンし、ファイルシステムのサイズを変更します。拡張操作が正常に完了すると、KubernetesはPVCサイズを更新します。

この例では、 `san-pvc`を使用するポッドが作成されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ**4**：**PV**を拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集して、`spec.resources.requests.storage`を2Giに更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

ステップ**5**：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正しく機能したことを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

FCボリュームを拡張する

CSI プロビジョナーを使用して FC 永続ボリューム（PV）を拡張できます。

メモ	FC ボリュームの拡張は、 `ontap-san`ドライバでサポートされており、Kubernetes 1.16 以降が必要です。

ステップ**1**：ボリューム拡張をサポートするように**StorageClass**を設定する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドを `true`に設定します。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存のStorageClassの場合は、編集して `allowVolumeExpansion`パラメータを含めます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、 `spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順3：**PVC**を接続するポッドを定義する

サイズを変更するには、PVをポッドに接続します。FC PVのサイズを変更する場合、次の2つのシナリオがあります：

- PVがポッドに接続されている場合、Tridentはストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
- アタッチされていないPVのサイズを変更しようとすると、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、Tridentはデバイスを再スキャンし、ファイルシステ

ムのサイズを変更します。拡張操作が正常に完了すると、KubernetesはPVCサイズを更新します。

この例では、 `san-pvc`を使用するポッドが作成されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ**4**：**PV**を拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集して、 `spec.resources.requests.storage`を2Giに更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

ステップ**5**：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正しく機能したことを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

NFSボリュームを拡張する

Tridentは、 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、および `azure-netapp-files`バックエンドでプロビジョニングされたNFS PVのボリューム拡張をサポートしています。

ステップ**1**：ボリューム拡張をサポートするように**StorageClass**を設定する

NFS PVのサイズを変更するには、管理者はまず、 `allowVolumeExpansion`フィールドを `true`に設定して、ボリューム拡張を可能にするようにストレージクラスを設定する必要があります：

```
cat storageclass-ontapnas.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

このオプションを使用せずにすでにストレージクラスを作成している場合は、 `kubectl edit storageclass`を使

用して既存のストレージクラスを編集し、ボリュームの拡張を可能にすることができます。

手順2：作成したStorageClassを使用してPVCを作成する

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident は、この PVC に対して 20 MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES    STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ3：PVを拡張する

新しく作成した20 MiBのPVを1 GiBにサイズ変更するには、PVCを編集して
`spec.resources.requests.storage`を1 GiBに設定します：

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

ステップ**4**：拡張を検証する

PVC、PV、および Trident ボリュームのサイズを確認することで、サイズ変更が正しく機能したことを検証できます：

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE   | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
```

RWX NVMe サブシステムの制限について

ReadWriteMany （RWX）ボリュームのうち、NVMeプロトコルを使用するものは、ボリュームあたり64ノードという拡張性の制限があります。以下では、制限事項、関連するNVMeサブシステムアーキテクチャの説明、および必要な解決手順の概要を示します。

64ノード制限について理解する

NVMeプロトコルでReadWriteMany（RWX）ボリュームを使用する予定がある場合、Kubernetesクラスタでは、単一のRWX NVMeボリュームを64ノードを超えてマウントすることはできません。

同じRWX NVMe PersistentVolumeClaim をマウントするワークロードを64ノードを超えてスケジュールしないでください。

この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。

NVMeサブシステムモデルを理解する

ボリュームごとのサブシステムモデル（**Trident 26.02**より前のリリース）

Trident 26.02より前のリリースでは、RWX NVMeボリュームはボリュームごとのサブシステムモデルを使用してプロビジョニングされます。各RWX NVMeボリュームは、ONTAP上の専用のNVMeサブシステムにマッピングされます。

このモデルはシンプルですが、スケーラビリティの上限が低くなっています。大規模な Kubernetes クラスターでは、各 RWX ボリュームが専用のサブシステムを消費するため、サブシステムコントローラの制限にすぐに達してしまいます。

スーパーサブシステムモデル（**Trident 26.02**で導入）

Trident 26.02以降、RWX NVMeボリュームは共有スーパーサブシステムモデルを使用します。複数のRWX NVMeボリュームが同じNVMeサブシステムを共有します。

各スーパーサブシステムは、最大1024個のネームスペース（ボリューム）をサポートします。このモデルはRWXワークロードのスケーラビリティを大幅に向上させ、ONTAPサブシステムの制限に達する可能性を低減します。

各RWX NVMeボリュームは最大64ノードをサポートします。

エラーの兆候を特定する

RWX NVMe ボリュームを大規模に作成または接続すると、次のようなエラーが発生する可能性があります：

```
Maximum number of controllers reached. No more controllers can be created.
```

このエラーは、ONTAP NVMe サブシステムコントローラの制限に達したことを示しています。

サブシステム制限エラーを解決する

ボリュームごとのサブシステム制限を超えてスーパーサブシステムモデルを活用するには、Trident 26.02以降にアップグレードしてください。

スーパーサブシステムモデルを適用するために **Trident** をアップグレードする

RWX NVMe ボリュームにスーパーサブシステムモデルを適用するには：

1. Trident をバージョン 26.02 以降にアップグレードしてください。
2. RWX NVMeボリュームを使用するすべてのポッドのレプリカ数をゼロに縮小します。
3. ワークロードがRWX NVMeボリュームをアクティブに使用していないことを確認してください。
4. ポッドをスケールアップして元に戻します。

この再起動シーケンスにより、RWX NVMe ボリュームがスーパーサブシステムモデルを使用して接続されることが保証されます。

- この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。
- 64ノードの制限は、RWX NVMe ボリュームごとに適用されます。

- その他のアクセスモードおよびプロトコルには影響はありません。

コントローラーのスケーラビリティ

Tridentは、複数のストレージドライバ間での並行処理を改善することで、コントローラのスケーラビリティを向上させます。お客様は、どの Trident ドライバが一般提供時にコントローラのスケーラビリティをサポートし、どのドライバが Trident 26.02 でテクニカルプレビューとして利用可能かを確認できます。これにより、スケーラブルな Kubernetes 環境において、十分な情報に基づいた導入の意思決定と適切なリスク管理が可能になります。

主要な概念と定義

コントローラーのスケーラビリティ

コントローラーのスケーラビリティとは、Trident コントローラーが、複数のストレージ操作を単一のロックの背後で直列化するのではなく、並列に処理できる能力を指します。これらの操作には、ボリュームの作成、削除、サイズ変更、スナップショットの作成と削除、ボリュームの公開と非公開、およびバックエンド管理が含まれます。

コントローラーのスケーラビリティが有効になっている場合、異なるボリュームおよびバックエンドに対する操作が同時に実行されます。これにより、スループットが向上し、PersistentVolumeClaim および VolumeSnapshot の同時操作数が多い環境でのエンドツーエンドの操作時間が短縮されます。

コントローラーのスケーラビリティサポート

Tridentは、ストレージドライバに応じて異なる成熟度レベルでコントローラのスケーラビリティをサポートします。

一般提供

以下のドライバは、Trident 26.02 の一般提供開始時点でコントローラのスケーラビリティをサポートしています：

- `ontap-san`
- `ontap-nas`
- `google-cloud-netapp-volumes`

メモ

`google-cloud-netapp-volumes`と `google-cloud-netapp-volumes-san`のドライバは異なります。 `google-cloud-netapp-volumes`のみがサポートされています。バックエンド構成または例では `google-cloud-netapp-volumes-san`を使用しないでください。

コントローラーのスケーラビリティを有効にする

コントローラーのスケーラビリティは、 `enableConcurrency`設定オプションによって制御されます。このオプションは、Trident のインストール時または既存のデプロイメントを更新することで、明示的に有効にする

必要があります。

Tridentオペレーターの配置

Tridentオペレータでコントローラのスケーラビリティを有効にするには、 `TridentOrchestrator`カスタムリソースで `enableConcurrency`を `true`に設定します。

新規インストール

`TridentOrchestrator` CR を作成または編集し、 `enableConcurrency` を `true` に設定します：

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

CR を適用します：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

既存のインストール

既存の `TridentOrchestrator` CR にパッチを適用して、コントローラーのスケーラビリティを有効にします：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

設定が適用されたことを確認します。

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm デプロイメント

Helm でコントローラーのスケーラビリティを有効にするには、 `enableConcurrency` の値を `true` に設定します。

新規インストール

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

既存のインストール

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

または、カスタムの `values.yaml`ファイルで `enableConcurrency`を `true`に設定します：

```
# values.yaml
enableConcurrency: true
```

次に、値ファイルを使用してインストールまたはアップグレードします：

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl デプロイメント

コントローラーのスケーラビリティを `tridentctl`で有効にするには、インストール中に `--enable-concurrency`フラグを渡してください。

新規インストール

```
tridentctl install -n trident --enable-concurrency
```

既存のインストール

既存の `tridentctl`ベースのデプロイメントでコントローラーのスケーラビリティを有効にするには、アンインストールしてからフラグを使用して再インストールしてください：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

コントローラーのスケーラビリティが有効になっていることを確認します

コントローラーのスケーラビリティを有効にした後、コントローラーポッドのログを確認して、Trident コントローラーが同時実行を有効にした状態で動作していることを確認します：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

同時実行が有効になっていることを示すログエントリが表示されるはずです。

技術プレビュー

以下のドライバーは、Trident 26.02 においてコントローラーのスケーラビリティをテクニカルプレビューとしてサポートしています：

- `nas-eco`
- `san-eco`

これらのドライバーについて：

- コントローラーの並行処理は評価およびテストに利用可能です。
- 今後のリリースでは動作が変わる可能性があります
- 本番環境での使用は推奨されません

並行動作

コントローラーのスケーラビリティが有効になっている場合：

- Tridentは、単一のグローバルロックをリソースごとのきめ細かいロックに置き換えます
- 同じリソースを変更する操作は、データの一貫性を維持するために直列化されます。
- リソースから読み取りのみを行う操作は、そのリソースに対する他の読み取り操作と同時に実行できます。
- Trident は、バックエンドストレージシステムへの過負荷を防ぐため、管理 LIF ごとの同時 ONTAP API リクエスト数を 20 件に制限します。
- 複数のバックエンドが同じ管理 LIF を共有する場合、この 20 リクエストの制限も共有されます。

既知の制限事項と考慮事項

コントローラーのスケーラビリティに関しては、以下の点を考慮する必要があります。

- 並行処理はTridentコントローラによって内部的に管理されます
- このリリースでは、ユーザーが設定可能な同時実行制限はありません。
- 全体のスループットは以下によって異なります：
  - 使用中のストレージドライバ
  - バックエンドの応答性
  - Kubernetes APIサーバーのパフォーマンス
- 高い同時実行性はバックエンドストレージシステムへの負荷を増加させる可能性があります

注意点と制限事項

以下の制限が Trident 26.02 に適用されます：

- コントローラの拡張性動作は、すべてのドライバで同一ではありません。
- テクニカルプレビュー版ドライバーでは、以下の現象が発生する可能性があります。
  - 高負荷時の性能が不安定
  - リリース間の動作の変化
- 並列実行のため、同時実行操作のデバッグはより複雑になる可能性があります。
- メトリクスとログには、インターリーブされた操作出力が表示される場合があります

推奨

- 高い拡張性を必要とする本番環境では、一般提供版（GA）ドライバーを使用してください。
- 非本番環境でテクニカルプレビュー版ドライバーを評価する
- 大規模運用時のバックエンドとコントローラーのパフォーマンスを監視する
- 自動化スクリプトでは、操作の順序を仮定しないようにしてください。

自動ボリューム拡張

自動ボリューム拡張により、Trident によってプロビジョニングされた永続ボリュームは、使用容量が定義されたしきい値に達すると自動的に拡張されます。この機能により、運用上の負担が軽減され、容量不足によるアプリケーションの中断を防ぐことができます。自動ボリューム拡張は、Autogrow ポリシーを使用して実装されます。Autogrow ポリシーでは以下を定義します：

- 拡張をトリガーする利用率の閾値
- ボリュームが増加する量
- ボリュームが到達できる最大サイズ

メモ	定義された使用率しきい値を超えると、ボリュームのサイズは自動的に増加します。ボリュームが自動的に縮小されることはありません。

要件

ボリュームの自動拡張を構成する前に、次の要件が満たされていることを確認してください。

- Trident 26.02以降
- カスタムリソースを作成するためのロールベースのアクセス制御権限 `TridentAutogrowPolicy`
- StorageClassesで構成 `allowVolumeExpansion: true`
- サポート対象の ONTAP プロトコル：
  - Network File System（NFS）

- Internet Small Computer Systems Interface（iSCSI）
- 不揮発性メモリエクスプレス（NVMe）
- Fiber Channel Protocol（FCP）

制限事項

- ONTAP 9.16.1 より前の ONTAP Non-Volatile Memory Express raw ブロックボリュームは、自動拡張をサポートしていません。
- ストレージエリアネットワークボリュームの場合、 `growthAmount`が50メビバイト以下であれば、Tridentは、結果のサイズが `maxSize`を超えない限り、サイズ変更前に値を自動的に51メビバイトに増やします。
- 既存環境においては、ボリューム公開の移行動作により、一部の既存ボリュームで自動拡張が機能しない場合があります。
- ボリュームが `maxSize`に達すると、それ以上の拡張は発生しません。
- 自動ボリューム拡張でサポートされているプロトコル：
  - Network File System（NFS）
  - Internet Small Computer Systems Interface（iSCSI）
  - 不揮発性メモリエクスプレス（NVMe）
  - Fiber Channel Protocol（FCP）

ボリュームの自動拡張ポリシーによるプロビジョニング

自動拡張ポリシーは、次の2つのレベルで構成できます：

- ストレージクラスレベル：すべてのボリュームのデフォルトを設定します（アノテーションを使用）
- PVC レベル：ストレージクラスのデフォルトをオーバーライドします（アノテーションを使用）

自動成長ポリシーを作成する

自動拡張ポリシーを使用すると、ボリュームが定義された容量しきい値に達したときにボリュームを自動的に拡張できます。

以下のものを用意してください：

- Trident 26.02以降がインストールされている
- `TridentAutogrowPolicy`リソースを作成するためのロールベースのアクセス制御権限
- ワークロードの増加要件の理解

ボリュームの自動拡張ポリシーは、定義された容量しきい値に達したときにボリュームが自動的に拡張される方法を定義します。

自動拡張ポリシーはワークフローのどの時点でも作成できます：

- StorageClassesとボリュームが作成される前
- StorageClassesが存在した後

- ボリュームのプロビジョニング後

この柔軟性により、既存のリソースを再作成せずに自動拡張を導入できます。

自動拡張ポリシーの仕様

自動拡張ポリシーは、次のように定義される Kubernetes カスタムリソースです：

フィールド	概要	フォーマット	必須	例	デフォルト
名前	一意のポリシー識別子	文字列	はい	production-db-ポリシー	なし
usedThreshold	拡張をトリガーする容量の割合	パーセンテージ文字列	はい	「80%」	なし
growthAmount	しきい値に達したときに増加する量	パーセンテージまたはサイズ	いいえ	「10%」または「5Gi」	「10%」
maxSize	最大ボリュームサイズの制限	Kubernetesの数量	いいえ	「500Gi」	無制限

自動成長ポリシーを作成する

手順

1. 自動拡張ポリシーを定義する YAML ファイルを作成します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. クラスターにポリシーを適用します：

```
kubectl apply -f autogrow-policy.yaml
```

3. ポリシーが作成されたことを確認します：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

期待される出力

```
NAME                 USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow    80%              10%             Success
```

ポリシーの状態

ポリシーを作成したら、Tridentが仕様を検証し、次のいずれかの状態を割り当てます：

状態	概要	必要なアクション
成功	ポリシーは検証され、使用できる状態です。	なし。
失敗	検証エラーが検出されました。	仕様を確認して修正してください。
削除中	削除中です。	完了するまでお待ちください。

ポリシーを**StorageClass**に関連付ける

`trident.netapp.io/autogrowPolicy`アノテーションを使用して、自動拡張ポリシーをStorageClassに関連付けることができます。そのStorageClassからプロビジョニングされたすべてのボリュームは、ポリシーを継承します。

メモ　StorageClassには `allowVolumeExpansion: true`が必要です。

手順

1. Autogrow Policyアノテーションを使用してStorageClassを作成または変更します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. StorageClassを適用します。

```
kubectl apply -f storageclass.yaml
```

3. アノテーションを確認します：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

期待される出力

```
production-db-policy
```

ポリシーの優先順位

自動拡張ポリシー注釈がStorageClassとPVCの両方に設定されている場合、Tridentは次の優先順位ルールを適用します：

1. *PVC注釈が優先されます。*PVCが `trident.netapp.io/autogrowPolicy`を設定した場合、その値が常に使用されます。
2. **StorageClass** アノテーションは、**PVC** にアノテーションがない場合にのみ適用されます。
3. *どちらにもアノテーションがない場合、*ボリュームの自動拡張ポリシーは適用されません。

StorageClassアノテーション	**PVC**アノテーション	効果的な行動
`trident.netapp.io/autogrowPolicy: standard-agp`	未設定	用途 `standard-agp`
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	使用 `logs-policy`（PVCはStorageClassをオーバーライドします）。
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	自動拡張ポリシーなし（PVC は自動拡張を無効にします）。
未設定	`trident.netapp.io/autogrowPolicy: dev-policy`	用途 `dev-policy`
未設定	未設定	自動拡張ポリシーなし。

設定例

次の例は、さまざまなユースケースにおける一般的な Autogrow Policy 構成を示しています。

本番データベースに対する保守的なポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

固定増分によるログストレージ

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

デフォルト付きの最小限のポリシー

`growthAmount`と `maxSize`を省略すると、Tridentはデフォルトの(`10%`（成長、無制限のサイズ）を使用します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

カスタム**maxSize**およびデフォルト**growthAmount**を使用したポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

無制限の**maxSize**による積極的な成長

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

小数パーセンテージを含むポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

メモ	小数点以下のパーセンテージがサポートされています。小数点以下3桁以上を指定すると、Tridentは値を小数点以下3桁に丸めます。

NAS StorageClass（自動拡張付き）

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass（自動拡張付き）

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

自動成長ポリシーの管理

自動拡張ポリシーを作成したら、必要に応じてそれを表示、更新、削除できます。特定のポリシーを使用しているボリュームを監視することもできます。

自動拡張ポリシーを表示

すべてのポリシーを一覧表示する

`kubectl`を使用して、クラスター内のすべての自動拡張ポリシーを一覧表示します：

```
kubectl get tridentautogrowpolicy
```

あるいは、 `tridentctl`：

```
tridentctl get autogrowpolicy
```

ポリシーの詳細を表示

ポリシーの完全な仕様とステータスを表示するには：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

YAML 形式でポリシーとそれに関連付けられたボリュームを表示するには：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

自動成長ポリシーを更新する

既存のポリシーを変更して、しきい値、増加量、または最大サイズを変更できます。変更は、ポリシーを使用するすべてのボリュームに対して直ちに有効になります。

重要	今回の変更は、現在このポリシーを使用しているすべてのボリュームに影響します。可能であれば、変更内容はまず非本番環境でテストしてください。

手順

1. ポリシーを編集します：

   ```
   kubectl edit tridentautogrowpolicy production-db-policy
   ```

2. 必要に応じて `spec`フィールドを変更します：

   ```
   spec:
     usedThreshold: "75%"     # Changed from 80%
     growthAmount: "20%"      # Changed from 10%
     maxSize: "1Ti"           # Changed from 500Gi
   ```

3. 保存して終了します。変更は直ちに有効になります。

更新に関する考慮事項

- 即時効果： ポリシーを使用するすべてのボリュームは、次回の成長評価時に新しいパラメータを採用します。
- ボリュームの再起動は不要です： 変更は次の拡張操作に適用されます。
- **Test first:** 可能な場合は、非本番環境で変更を検証します。
- *変更を伝える：*共有ポリシーを変更するときにチームに通知します。

自動成長ポリシーを削除する

自動拡張ポリシーは、ボリュームがアクティブに使用されているときに誤って削除されるのを防ぐためにファイナライザー保護を使用します。

手順

1. ポリシーを削除します：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. ボリュームがまだポリシーを使用している場合、削除は `Deleting`状態になります。影響を受けるボリュームを確認します：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 影響を受ける各ボリュームからポリシーを削除します。次のいずれかのオプションを選択します。

   - オプション**A**：明示的に自動拡張を無効にする アノテーションを `"none"`に設定します：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

   - オプション**B**：注釈を完全に削除する：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

削除動作

シナリオ	動作
このポリシーを使用するボリュームはありません	ポリシーは直ちに削除されます。
ボリュームはポリシーを使用しています	ポリシーが `Deleting`状態になります。ファイナライザーは、すべてのボリュームが削除されるまで完了をブロックします。
すべてのボリュームがポリシーから削除されます	ファイナライザーが削除され、ポリシーが削除されます。

自動拡張ポリシーの使用状況を監視する

ポリシーを使用してボリュームをチェックする

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

ボリュームが使用するポリシーを確認する

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

ポリシーイベントを監視する

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

サポートされるプロトコル

Autogrow は次のストレージプロトコルをサポートしています：

- NFS
- iSCSI
- FCP
- NVMe

メモ	SANボリュームの場合、構成された `growthAmount`が50 MiB以下である場合、Tridentは、結果のサイズが `maxSize`を超えない限り、サイズ変更操作の増加量を自動的に51MBに増加します。

既知の制限事項

- **ONTAP NVMe raw**ブロックボリューム： ONTAP 9.16.1より前のバージョンで作成されたボリュームは、自動拡張をサポートしていません。
- 既存のボリューム（ブラウンフィールド展開）： 有効な自動拡張ポリシーが適用されている場合でも、既存のボリュームでは自動拡張が機能しない可能性があります。これは、ボリューム出版物の移行が進行中であるためです。移行が完了したことを確認するには、Tridentコントローラーログで `"Migration completed"`メッセージを確認してください。

よくある質問

Tridentはいつ閾値を評価しますか？

Tridentはボリュームの使用状況を継続的に監視します。使用容量が `usedThreshold`を超えると、Tridentは内部サイズ変更要求を作成し、設定された `growthAmount`でボリュームを拡張します。

たとえば、このポリシーは容量の80%で拡張をトリガーし、ボリュームを毎回10%ずつ最大500 GiBまで増加させます：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

ボリュームがすでにプロビジョニングされた後にポリシーを適用できますか？

はい。自動拡張ポリシーはいつでも作成でき、 `trident.netapp.io/autogrowPolicy`アノテーションを追加または更新することで既存のPVCに適用できます。PVCまたはStorageClassを再作成する必要はありません。

既存の PVC にポリシーを適用するには：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

既存のStorageClassにポリシーを適用するには：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

StorageClassと**PVC**の両方に自動拡張ポリシーを設定するとどうなりますか？

PVC 注釈は常に優先されます。PVC に `trident.netapp.io/autogrowPolicy`注釈がある場合、 Trident はStorageClassの指定に関係なくその値を使用します。詳細については、"ポリシーの優先順位"を参照してください。

例えば、このStorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

そして、この PVC は StorageClass ポリシーを上書きします：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Tridentは `logs-policy`を `database-pvc`に使用し、 `standard-agp`は使用しません。

特定のボリュームの自動拡張を無効にするにはどうすればよいですか？

PVC注釈を `"none"`に設定します。これにより、そのボリュームのStorageClassレベルのポリシーが上書きされます：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

自動拡張が無効になっていることを確認できます：

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

期待される出力

```
none
```

ボリュームが**maxSize**に達するとどうなりますか？

Tridentはボリュームの拡張を停止します。使用量が `usedThreshold`を超えて増加し続けても、そのボリュームに対してそれ以上のサイズ変更要求は作成されません。

例えば、このポリシーでは、Tridentが100 GiBに達するとボリュームの拡張が停止します。

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

無制限に成長させるには、 `maxSize`を省略するか、 `0`に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

ボリュームを再起動せずにポリシーを変更できますか？

はい。ポリシーを更新すると、そのポリシーを使用しているすべてのボリュームは、次回の拡張評価時に新しいパラメータを採用します。ボリュームの再起動は必要ありません。

既存のポリシーを更新するには：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

必要に応じてフィールドを変更します：

```
spec:
  usedThreshold: "75%"     # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

保存して終了します。更新されたポリシーを確認します：

```
kubectl get tridentautogrowpolicy production-db-policy
```

期待される出力

```
NAME                    USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy    75%              20%             Success
```

ポリシーが失敗状態になっているのはなぜですか？

A `Failed`状態は、ポリシー仕様に検証エラーが含まれていることを示します。エラーの詳細を表示するには、次のコマンドを実行します：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

一般的な原因には、無効な `usedThreshold`（1〜99%である必要があります）、 `growthAmount`が`maxSize`を超えている場合、または無効なKubernetes数量フォーマットなどがあります。仕様を修正して再適用してください：

```
kubectl apply -f autogrow-policy.yaml
```

ポリシーを削除できないのはなぜですか？

ポリシーはファイナライザー保護を使用します。ボリュームがまだポリシーを使用している場合、削除は`Deleting`状態になり、すべてのボリュームがポリシーから削除されるまで待機します。

影響を受けるボリュームを特定します：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

次に、各 PVC から注釈を削除します：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

すべてのボリュームが削除されると、ファイナライザーが解放され、ポリシーが削除されます。

自動拡張はすべての**ONTAP**バックエンドで動作しますか？

Autogrow は、NFS、iSCSI、FCP、NVMe プロトコルをサポートします。ただし、NVMe raw ブロックボリュームには ONTAP 9.16.1 以降が必要です。

ブラウンフィールド展開内の既存のボリュームでは、自動拡張が有効になる前にボリューム公開の移行を完了する必要がある場合があります。Tridentコントローラーログを確認して、移行ステータスを検証します：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
completed"
```

次のStorageClass例では、NAS および SAN バックエンドに設定された自動拡張を示しています：

NASバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SANバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN ボリュームの最小増加量はどれくらいですか？

SAN ボリュームの場合、有効な最小増加量は 51 MB です。 `growthAmount`を 50 MiB 以下に設定すると、Trident はサイズ変更操作のために自動的に増加量を 51 MB に増加します。

例えば、このポリシーは `growthAmount`の `"40Mi"`を設定しますが、Tridentはこれを使用するすべてのSANボリュームに51MBの拡張を適用します（

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

この自動調整を回避するには、 `growthAmount` を 50 MiB を超える値に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

ボリュームをインポート

`tridentctl import`を使用するか、Tridentインポート
アノテーションを使用して永続ボリューム要求（PVC）を作成することで、既存のストレージ
ボリュームをKubernetes PVとしてインポートできます。

概要と考慮事項

次の目的でボリュームをTridentにインポートすることができます：

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタを再構築する
- 災害復旧時にアプリケーションデータを移行する

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- TridentでインポートできるのはRW（読み書き）タイプのONTAPボリュームのみです。DP（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをTridentにインポートする前に、ミラー関係を解除する必要があります。
- アクティブな接続なしでボリュームをインポートすることをお勧めします。アクティブに使用されている

ボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

警告	これは、Kubernetes が以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、ブロックボリュームの場合に特に重要です。その結果、データ破損が発生する可能性があります。

- ただし `StorageClass`PVCで指定する必要がありますが、Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。したがって、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在する場合でも、インポートは失敗しません。
- 既存のボリュームサイズが決定され、PVCに設定されます。ボリュームがストレージドライバによってインポートされると、ClaimRefを使用してPVCへの参照を持つPVが作成されます。
  - 回収ポリシーは初期設定では PV で `retain`に設定されています。Kubernetes が PVC と PV を正常にバインドすると、回収ポリシーはストレージクラスの回収ポリシーと一致するように更新されます。
  - ストレージクラスの回収ポリシーが `delete`の場合、PV が削除されると、ストレージボリュームも削除されます。
- デフォルトでは、TridentがPVCを管理し、バックエンドのFlexVolボリュームとLUNの名前を変更します。 `--no-manage`フラグを渡して管理されていないボリュームをインポートし、 `--no-rename`フラグを渡してボリューム名を保持できます。
  - `--no-manage`* - `--no-manage`フラグを使用する場合、Tridentはオブジェクトのライフサイクル中、PVCまたはPVに対して追加の操作は実行されません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などの他の操作も無視されます。
  - `--no-rename`* - `--no-rename`フラグを使用する場合、`Trident`はボリュームをインポートするときに既存のボリューム名を保持し、ボリュームのライフサイクルを管理します。このオプションは、 `ontap-nas`、 `ontap-san`（ASA r2システムを含む）、および `ontap-san-economy`ドライバーでのみサポートされます。

ヒント	これらのオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、それ以外の場合は Kubernetes の外部でストレージボリュームのライフサイクルを管理する場合に役立ちます。

- PVC と PV に注釈が追加されます。注釈には、ボリュームがインポートされたことと、PVC と PV が管理されているかどうかを示すという 2 つの目的があります。この注釈は変更または削除しないでください。

ボリュームをインポートする

`tridentctl import`を使用するか、`Trident`インポートアノテーションを使用して`PVC`を作成することで、ボリュームをインポートできます。

メモ	PVC アノテーションを使用する場合は、 `tridentctl`をダウンロードしたり使用してボリュームをインポートしたりする必要はありません。

tridentctlの使用

手順

1. PVCを作成するために使用するPVCファイル（例： `pvc.yaml`）を作成します。PVCファイルには `name`、 `namespace`、 `accessModes`、および `storageClassName`を含める必要があります。必要に応じて、PVC定義で `unixPermissions`を指定することもできます。

   以下は最小仕様の例です：

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: my_claim
     namespace: my_namespace
   spec:
     accessModes:
       - ReadWriteOnce
     storageClassName: my_storage_class
   ```

メモ	必須パラメータのみを入力してください。PV名やボリュームサイズなどの追加パラメータは、インポートコマンドの失敗の原因となる可能性があります。

2. `tridentctl import`コマンドを使用して、ボリュームを含むTridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（例：ONTAP FlexVol、Element Volume）を指定します。 `-f`引数は、PVCファイルへのパスを指定するために必要です。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

PVCアノテーションの使用

手順

1. 必要なTridentインポートアノテーションを含むPVC YAMLファイル（例： `pvc.yaml`）を作成します。PVCファイルには以下を含める必要があります：

   - `name`および `namespace`メタデータ内
   - `accessModes`、 `resources.requests.storage`、および `storageClassName`仕様
   - 注釈：
     - `trident.netapp.io/importOriginalName`：バックエンドのボリューム名
     - `trident.netapp.io/importBackendUUID`：ボリュームが存在するバックエンドUUID
     - `trident.netapp.io/notManaged`（オプション）：管理されていないボリュームの場合は `"true"`に設定します。デフォルトは `"false"`です。

       以下は、管理対象ボリュームをインポートするための仕様例です：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. PVC YAML ファイルを Kubernetes クラスターに適用します：

```
kubectl apply -f <pvc-file>.yaml
```

Trident はボリュームを自動的にインポートし、PVC にバインドします。

例

サポートされているドライバーの次のボリュームインポート例を確認してください。

ONTAP NAS と ONTAP NAS FlexGroup

Tridentは、 `ontap-nas`および `ontap-nas-flexgroup`ドライバーを使用したボリュームインポートをサポートしています。

メモ

- Tridentは `ontap-nas-economy`ドライバを使用したボリュームインポートをサポートしていません。
- `ontap-nas`および `ontap-nas-flexgroup`ドライバは、重複するボリューム名を許可しません。

`ontap-nas`ドライバーで作成された各ボリュームは、ONTAPクラスタ上のFlexVolボリュームです。 `ontap-nas`ドライバーを使用したFlexVolボリュームのインポートも同様に機能します。ONTAPクラスタにすでに存在するFlexVolボリュームは、 `ontap-nas` PVCとしてインポートできます。同様に、FlexGroupボリュームは `ontap-nas-flexgroup` PVCとしてインポートできます。

tridentctl を使用した **ONTAP NAS** の例

次の例は、 `tridentctl`を使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、 `ontap_nas`というバックエンド上の `managed_volume`というボリュームをインポートします：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
```

管理されていないボリューム

`--no-manage`引数を使用する場合、Tridentはボリュームの名前を変更しません。

次の例は、 `unmanaged_volume`を `ontap_nas`バックエンドでインポートします：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
```

PVC アノテーションを使用した **ONTAP NAS** の例

次の例は、PVC アノテーションを使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、 `81abcb27-ea63-49bb-b606-0a5315ac5f21`から `ontap_volume1`という名前の `ontap-nas`ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

管理されていないボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、バックエンド `34abcb27-ea63-49bb-b606-0a5315ac5f34`からという名前 `ontap-volume2`の `1Gi` `ontap-nas` ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Tridentは、 `ontap-san`（iSCSI、NVMe/TCP、FC）および `ontap-san-economy`ドライバーを使用したボリュームインポートをサポートしています。

Tridentは、単一のLUNを含むONTAP SAN FlexVolボリュームをインポートできます。これは `ontap-san`ドライバと一致しており、各PVCに対してFlexVolボリュームを作成し、FlexVolボリューム内にLUNを作成します。TridentはFlexVolボリュームをインポートし、PVC定義に関連付けます。Tridentは、複数のLUNを含む `ontap-san-economy`ボリュームをインポートできます。

次の例は、管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています：

管理ボリューム

管理対象ボリュームの場合、TridentはFlexVolボリュームの名前を `pvc-<uuid>`形式に変更し、FlexVolボリューム内のLUNの名前も `lun0`に変更します。

次の例では、 `ontap-san-managed`バックエンド上に存在するFlexVol volumeを `ontap_san_default`インポートします：

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理されていないボリューム

次の例は、 `unmanaged_example_volume`を `ontap_san`バックエンドでインポートします：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

次の例に示すように、Kubernetes ノード IQN と IQN を共有する igroup にマップされた LUN がある場合は、次のエラーが表示されます： `LUN already mapped to initiator(s) in this group`ボリュームをインポートするには、イニシエーターを削除するか、LUN のマップを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economyの例

以下の例は、 `ontap-san-economy`バックエンドの管理ボリュームと非管理ボリュームをインポートする方法を示しています。

管理ボリューム

管理ボリュームをインポートすると、TridentがFlexVolの所有権を取得し、名前を変更します。同じFlexVolから複数のLUNをインポートする場合は、この名前変更を考慮する必要があります。

次の例では、 `lun1`をFlexVol `toimport`から `vol-managed-saneco`という名前の管理ボリュームとしてインポートします：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

インポート後 `lun1`、TridentはFlexVolの名前を変更します（例： `trident_lun_pool_xyz`）。同じFlexVolから追加のLUNをインポートするには、新しいFlexVol名を使用します：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

メモ	`ontap-san-economy`バックエンドは一度に1つのLUNをインポートします。スクリプトを使用すれば、複数のインポートを自動化できます。

管理されていないボリューム

管理されていないボリュームをインポートすると、TridentはFlexVolの所有権を取得しません。ただし、FlexVolとLUNはTridentの命名規則に従う必要があります。

FlexVol命名形式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX`は、バックエンド設定の `storagePrefix`の値です。デフォルトは `trident`です。
- `RANDOMSTRING`は任意の文字列です。

LUN の命名要件

LUN には `lun0`という名前を付ける必要があります。

例

もしあなたの `storagePrefix`が `xyz`の場合、LUNへの完全パスは次のとおりです：

```
trident_lun_pool_xyz_randomstring/lun0
```

要素

Tridentは、 `solidfire-san`ドライバを使用して、NetApp Element ソフトウェアおよびNetApp HCI ボリュームのインポートをサポートします。

メモ	Element ドライバは重複したボリューム名をサポートします。ただし、 Trident は重複したボリューム名がある場合はエラーを返します。回避策として、ボリュームのクローンを作成し、一意のボリューム名を指定して、クローンされたボリュームをインポートします。

次の例は、バックエンド `element_default`上の `element-managed`ボリュームをインポートします。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Tridentは `azure-netapp-files`ドライバを使用したボリュームインポートをサポートしています。

メモ	Azure NetApp Files ボリュームをインポートするには、ボリュームパスでボリュームを識別します。ボリュームパスは、 `:/`の後のボリュームのエクスポートパスの部分です。たとえば、マウントパスが `10.0.0.2:/importvol1`の場合、ボリュームパスは `importvol1`です。

次の例は、バックエンド `azurenetappfiles_40517`上の `azure-netapp-files`ボリュームを、ボリュームパス `importvol1`でインポートします。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetApp Volumes

Tridentは `google-cloud-netapp-volumes`ドライバを使用したボリュームインポートをサポートしています。

次の例では、ボリューム `testvoleasiaeast1`を持つバックエンド `backend-tbc-gcnv1`上のボリュームをインポートします。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

次の例では、同じリージョンに2つのボリュームが存在する場合に `google-cloud-netapp-volumes`ボリュームをインポートします：

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

ボリューム名とラベルをカスタマイズする

Tridentを使用すると、作成したボリュームに意味のある名前とラベルを割り当てることができます。これにより、ボリュームを識別し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできるようになります。バックエンドレベルでテンプレートを定義して、カスタムボリューム名とカスタムラベルを作成することもできます。作成、インポート、またはクローンを作成するボリュームはすべてテンプレートに準拠します。

開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

- ボリュームの作成、インポート、およびクローン処理。
- `ontap-nas-economy`ドライバの場合、Qtreeボリュームの名前のみが名前テンプレートに準拠します。
- `ontap-san-economy`ドライバの場合、LUN名のみが名前テンプレートに準拠します。

制限事項

- カスタムボリューム名は ONTAP オンプレミスドライバーのみと互換性があります。
- カスタムラベルは、 `ontap-san`、 `ontap-nas`、および `ontap-nas-flexgroup`ドライバでのみサポートされます。
- カスタムボリューム名は既存のボリュームには適用されません。

カスタマイズ可能なボリューム名の主な動作

- 名前テンプレートの構文が無効であるために障害が発生した場合、バックエンドの作成は失敗します。ただし、テンプレートの適用に失敗した場合、ボリュームは既存の命名規則に従って命名されます。
- バックエンド構成の名前テンプレートを使用してボリュームに名前を付ける場合、ストレージプレフィックスは適用されません。必要なプレフィックス値をテンプレートに直接追加できます。

名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベル、あるいはその両方で定義できます。

ルートレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

プールレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

名前 **template** の例

例**1**：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

例**2**：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

考慮すべき点

1. ボリュームのインポートの場合、既存のボリュームに特定の形式のラベルがある場合にのみラベルが更新されます。例えば：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`
2. 管理対象ボリュームのインポートの場合、ボリューム名はバックエンド定義のルートレベルで定義された名前テンプレートに従います。
3. Trident は、ストレージプレフィックスでのスライス演算子の使用をサポートしていません。
4. テンプレートによって一意のボリューム名が生成されない場合は、Tridentがいくつかのランダムな文字を追加して、一意のボリューム名を作成します。
5. NASエコノミーボリュームのカスタム名が64文字を超える場合、Tridentは既存の命名規則に従ってボリューム名を付けます。その他のすべてのONTAPドライバーでは、ボリューム名が名前制限を超えると、ボリューム作成プロセスが失敗します。

名前空間間で**NFS**ボリュームを共有する

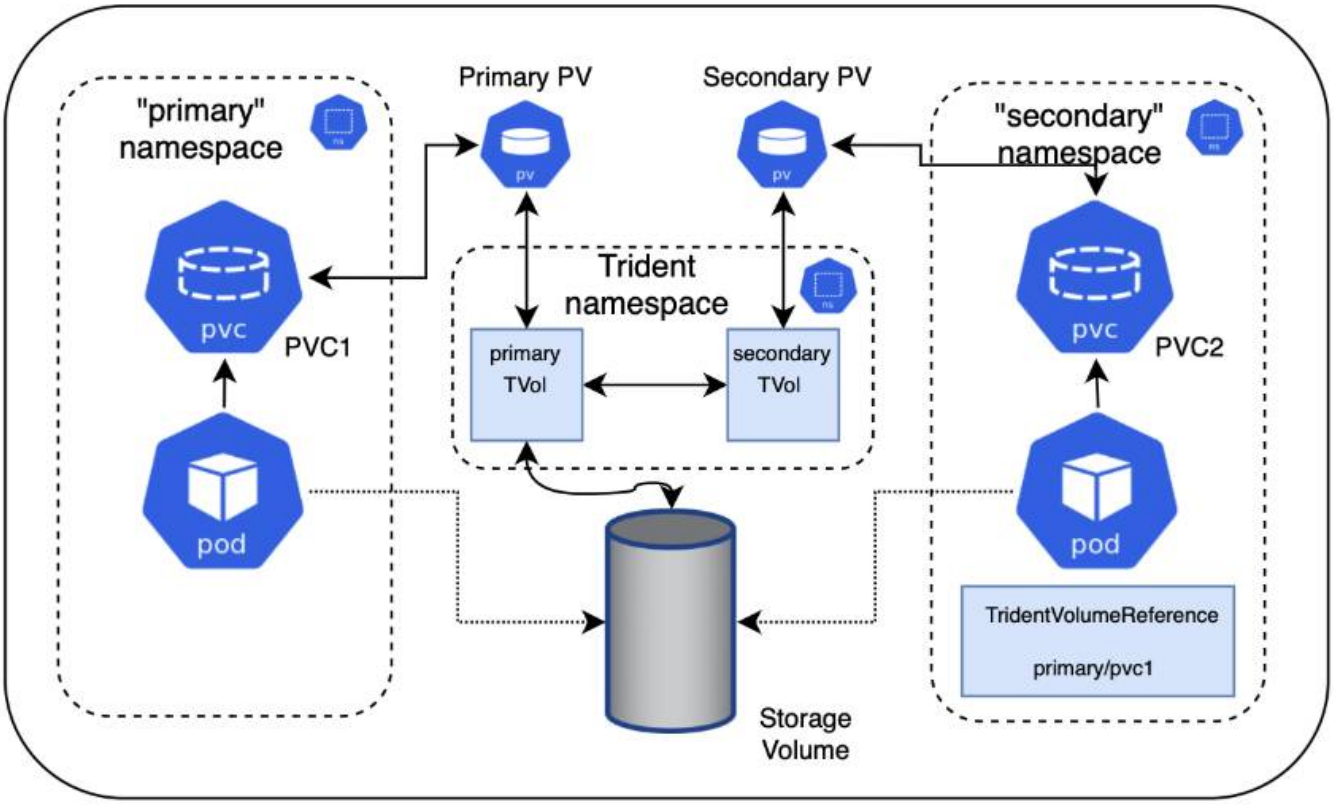
Tridentを使用すると、プライマリ名前空間にボリュームを作成し、それを1つ以上のセカンダリ名前空間で共有できます。

機能

TridentVolumeReference CR を使用すると、ReadWriteMany（RWX）NFS ボリュームを 1 つ以上の Kubernetes 名前空間にわたって安全に共有できます。この Kubernetes ネイティブソリューションには、次の利点があります：

- セキュリティを確保するための複数レベルのアクセス制御
- すべての Trident NFS ボリュームドライバーに対応
- tridentctl やその他の非ネイティブ Kubernetes 機能に依存しない

この図は、2 つの Kubernetes 名前空間間での NFS ボリュームの共有を示しています。



クイック スタート

わずか数ステップで NFS ボリューム共有を設定できます。

1 ボリュームを共有するようにソース **PVC** を設定する

ソース名前空間の所有者は、ソース PVC 内のデータにアクセスする権限を付与します。

2 宛先ネームスペースに **CR** を作成する権限を付与します

クラスタ管理者は、宛先名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。

3 宛先名前空間に**TridentVolumeReference**を作成する

宛先名前空間の所有者は、ソースPVCを参照するTridentVolumeReference CRを作成します。

4 宛先ネームスペースに従属**PVC**を作成する

宛先ネームスペースの所有者は、ソース PVC のデータソースを使用するために従属 PVC を作成します。

ソースネームスペースと宛先ネームスペースを設定する

セキュリティを確保するには、名前空間間の共有には、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による連携とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. ＊ソースネームスペースの所有者＊：ソースネームスペースでPVC(`pvc1`を作成し、(`namespace2`宛てに共有する権限を付与するために、 `shareToNamespace`アノテーションを使用します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident は PV とそのバックエンド NFS ストレージボリュームを作成します。

メモ

- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。例えば、 `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、 `*`を使用します。例： `trident.netapp.io/shareToNamespace: *`
- PVCを更新して、 `shareToNamespace`アノテーションをいつでも含めることができます。

2. *クラスタ管理者：*適切なRBACが設定されていることを確認し、宛先ネームスペースの所有者が宛先ネームスペースにTridentVolumeReference CRを作成する権限を付与します。
3. 宛先名前空間の所有者： ソース名前空間を参照するTridentVolumeReference CRを宛先名前空間に作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者： 宛先ネームスペース(`namespace2`）でPVC(`pvc2`）を作成し、ソー

スPVCを指定するために `shareFromPVC`アノテーションを使用します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

メモ	デスティネーション PVC のサイズは、ソース PVC のサイズ以下である必要があります。

結果

Tridentは、デスティネーションPVCの `shareFromPVC`アノテーションを読み取り、デスティネーションPVを、独自のストレージリソースを持たず、ソースPVを指してソースPVストレージリソースを共有する従属ボリュームとして作成します。デスティネーションPVCとPVは正常にバインドされているように見えます。

共有ボリュームを削除する

複数の名前空間間で共有されているボリュームを削除できます。Tridentは、ソース名前空間のボリュームへのアクセスを削除し、ボリュームを共有する他の名前空間へのアクセスを維持します。ボリュームを参照するすべての名前空間が削除されると、Tridentはボリュームを削除します。

`tridentctl get`を使用して従属ボリュームを照会する

[`tridentctl`ユーティリティを使用して、 `get`コマンドを実行すると、従属ボリュームを取得できます。詳細については、link：../trident-reference/trident-cl.html[`tridentctl`コマンドとオプション]を参照してください。

```
Usage:
  tridentctl get [option]
```

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを従属ソースボリュームに制限します。
- `--subordinateOf string`：ボリュームの下位にクエリを制限します。

制限事項

- Tridentは、宛先ネームスペースが共有ボリュームに書き込むのを防ぐことはできません。共有ボリュームのデータが上書きされないようにするには、ファイルロックまたはその他のプロセスを使用する必要があります。
- ソースPVCへのアクセスを取り消すには、 `shareToNamespace`または `shareFromNamespace`アノテーションを削除するか、 `TridentVolumeReference`CRを削除しても取り消すことはできません。アクセスを取り消すには、従属PVCを削除する必要があります。
- 従属ボリュームではスナップショット、クローン、ミラーリングは実行できません。

詳細情報

クロスネームスペースボリュームアクセスの詳細については、次を参照してください：

- "NetAppTV"でデモをご覧ください。

名前空間を越えてボリュームを複製する

Tridentを使用すると、同じKubernetesクラスター内の別の名前空間の既存のボリュームまたはボリュームスナップショットを使用して、新しいボリュームを作成できます。

前提条件

ボリュームを複製する前に、ソースと宛先のバックエンドが同じタイプであり、同じストレージクラスを持っていることを確認してください。

メモ	名前空間をまたがるクローン作成は、 `ontap-san`および `ontap-nas`ストレージドライバでのみサポートされます。読み取り専用クローンはサポートされていません。

クイック スタート

わずか数ステップでボリュームのクローンを設定できます。

1. ボリュームをクローニングするソース **PVC** を設定します

ソース名前空間の所有者は、ソース PVC 内のデータにアクセスする権限を付与します。

2. 宛先ネームスペースに **CR** を作成する権限を付与します

クラスタ管理者は、宛先名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。

3. 宛先名前空間に**TridentVolumeReference**を作成する

宛先名前空間の所有者は、ソースPVCを参照するTridentVolumeReference CRを作成します。

4. 宛先ネームスペースにクローン **PVC** を作成する

宛先ネームスペースの所有者は、ソースネームスペースから PVC をクローニングするための PVC を作成します。

ソースネームスペースと宛先ネームスペースを設定する

セキュリティを確保するために、名前空間間でボリュームを複製するには、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による共同作業とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. ソースネームスペースの所有者： ソースネームスペース(`namespace1`）でPVC(`pvc1`）を作成し、`cloneToNamespace`アノテーションを使用して、宛先ネームスペース(`namespace2`）と共有する権限を付与します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident は PV とそのバックエンドストレージボリュームを作成します。

メモ

- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。例えば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、 `*`を使用します。例： `trident.netapp.io/cloneToNamespace: *`
- PVCを更新して、 `cloneToNamespace`アノテーションをいつでも含めることができます。

2. クラスタ管理者： 適切なRBACが設定されていることを確認し、宛先名前空間の所有者が宛先名前空間にTridentVolumeReference CRを作成する権限を付与します(`namespace2`）。
3. 宛先名前空間の所有者： ソース名前空間を参照するTridentVolumeReference CRを宛先名前空間に作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者： 宛先ネームスペース(namespace2）でPVC(pvc2）を作成し、cloneFromPVC`または `cloneFromSnapshot、および `cloneFromNamespace`アノテーションを使用してソースPVCを指定します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

制限事項

- ontap-nas-economy ドライバーを使用してプロビジョニングされた PVC の場合、読み取り専用クローンはサポートされません。

SnapMirrorを使用したボリュームのレプリケート

Tridentは、ディザスタリカバリのためにデータをレプリケートする、1つのクラスタ上のソースボリュームとピアクラスタ上のデスティネーションボリューム間のミラー関係をサポートします。 Trident Mirror Relationship（TMR）と呼ばれる名前空間付きのカスタムリソース定義（CRD）を使用して、次の操作を実行できます：

- ボリューム間のミラー関係を作成する（PVC）
- ボリューム間のミラー関係を削除する

- ミラー関係を解除
- 災害時（フェイルオーバー）にセカンダリ ボリュームを昇格する
- 計画されたフェイルオーバーまたは移行中に、クラスタ間でのアプリケーションのロスレス移行を実行します

レプリケーションの前提条件

始める前に、次の前提条件が満たされていることを確認してください：

ONTAP クラスタ

- **Trident**：ONTAPをバックエンドとして利用するソースとデスティネーションの両方のKubernetesクラスタに、Tridentバージョン22.10以降が存在している必要があります。
- ライセンス：ONTAP SnapMirror データ保護バンドルを使用する非同期ライセンスは、ソースとデスティネーション クラスタの両方で有効にする必要があります。詳細については、 "ONTAP における SnapMirror ライセンスの概要"を参照してください。

  ONTAP 9.10.1以降では、すべてのライセンスがNetAppライセンス ファイル（NLF）として提供されます。これは、複数の機能を有効にする単一のファイルです。詳細については、"ONTAP Oneに含まれるライセンス" を参照してください。

メモ	SnapMirror非同期保護のみがサポートされています。

ピアリング

- * クラスタと SVM * ： ONTAP ストレージ バックエンドはピアリングする必要があります。詳細については、 "クラスタとSVMのピアリングの概要"を参照してください。

重要	2つのONTAPクラスタ間のレプリケーション関係で使用されるSVM名が一意であることを確認してください。

- **Trident**および**SVM**：ピアリングされたリモートSVMは、デスティネーション クラスタのTridentで使用できる必要があります。

サポートされているドライバ

NetApp Tridentは、次のドライバーでサポートされているストレージクラスを使用して、NetApp SnapMirror テクノロジによるボリュームレプリケーションをサポートします： **`ontap-nas`**：**NFS** `ontap-san`：iSCSI **`ontap-san`**：**FC** `ontap-san`：NVMe/TCP（最小ONTAPバージョン9.15.1が必要）

メモ	SnapMirrorを使用したボリュームレプリケーションは、ASA r2システムではサポートされていません。ASA r2システムの詳細については、"ASA r2ストレージシステムの詳細"を参照してください。

ミラー **PVC** を作成する

次の手順に従って、CRD の例を使用して、プライマリ ボリュームとセカンダリ ボリューム間のミラー関係を作成します。

手順

1. プライマリ Kubernetes クラスタで次の手順を実行します：

   a. `trident.netapp.io/replication: true`パラメータを使用してStorageClassオブジェクトを作成します。

      例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 以前に作成したStorageClassを使用してPVCを作成します。

      例

      ```
      kind: PersistentVolumeClaim
      apiVersion: v1
      metadata:
        name: csi-nas
      spec:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 1Gi
        storageClassName: csi-nas
      ```

   c. ローカル情報を含むMirrorRelationship CRを作成します。

      例

      ```
      kind: TridentMirrorRelationship
      apiVersion: trident.netapp.io/v1
      metadata:
        name: csi-nas
      spec:
        state: promoted
        volumeMappings:
        - localPVCName: csi-nas
      ```

      Tridentはボリュームの内部情報とボリュームの現在のデータ保護（DP）状態を取得

し、MirrorRelationshipのstatusフィールドに入力します。

d. TridentMirrorRelationship CR を取得して、PVC の内部名と SVM を取得します。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. セカンダリ Kubernetes クラスタで次の手順を実行します：

a. StorageClassをtrident.netapp.io/replication: trueパラメータで作成します。

例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

b. デスティネーションとソースの情報を含むMirrorRelationship CRを作成します。

例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Tridentは、設定された関係ポリシー名（またはONTAPのデフォルト）でSnapMirror関係を作成し、初期化します。

c. 以前に作成したStorageClassを使用してPVCを作成し、セカンダリ（SnapMirrorデスティネーション）として機能させます。

例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

TridentはTridentMirrorRelationship CRDを確認し、関係が存在しない場合はボリュームの作成に失敗します。関係が存在する場合、Tridentは新しいFlexVolボリュームが、MirrorRelationshipで定義されたリモートSVMとピアリングされたSVM上に配置されるようにします。

ボリュームレプリケーションの状態

Tridentミラー リレーションシップ（TMR）は、PVC間のレプリケーション リレーションシップの一方の端を表すCRDです。デスティネーションTMRには状態があり、Tridentに望ましい状態を伝えます。デスティネーションTMRの状態は次のとおりです：

- 確立済み：ローカル PVC はミラー関係のデスティネーション ボリュームであり、これは新しい関係です。

- 昇格：ローカルPVCはReadWriteでマウント可能ですが、現在ミラー関係は有効ではありません。
- 再確立：ローカル PVC はミラー関係のデスティネーション ボリュームであり、以前もそのミラー関係にありました。
  - デスティネーション ボリュームがソース ボリュームと関係があった場合、デスティネーション ボリュームの内容が上書きされるため、再確立された状態を使用する必要があります。
  - ボリュームが以前にソースと関係していなかった場合、再確立された状態は失敗します。

計画外のフェイルオーバー中にセカンダリ **PVC** を昇格する

セカンダリ Kubernetes クラスタで次の手順を実行します：

- TridentMirrorRelationshipの_spec.state_フィールドを `promoted`に更新します。

計画されたフェイルオーバー中にセカンダリ **PVC** を昇格する

計画されたフェイルオーバー（移行）中に、次の手順を実行してセカンダリ PVC を昇格します：

手順

1. プライマリ Kubernetes クラスタで、PVC のスナップショットを作成し、スナップショットが作成されるまで待機します。
2. プライマリKubernetesクラスタで、SnapshotInfo CRを作成して内部の詳細を取得します。

   例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. セカンダリKubernetesクラスタで、*TridentMirrorRelationship* CRの_spec.state_フィールドを_promoted_に更新し、_spec.promotedSnapshotHandle_をスナップショットのinternalNameに更新します。
4. セカンダリKubernetesクラスタで、TridentMirrorRelationshipのステータス（status.stateフィールド）がpromotedになっていることを確認します。

フェイルオーバー後にミラー関係を復元する

ミラー関係をリストアする前に、新しいプライマリとして作成する側を選択します。

手順

1. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.remoteVolumeHandle_フィールドの値が更新されていることを確認します。
2. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.mirror_フィールドを `reestablished`に更新します。

追加操作

Tridentは、プライマリボリュームとセカンダリボリュームで次の操作をサポートします：

プライマリ**PVC**を新しいセカンダリ**PVC**にレプリケートする

プライマリ PVC とセカンダリ PVC が既に存在することを確認します。

手順

1. 確立されたセカンダリ（デスティネーション）クラスタからPersistentVolumeClaimとTridentMirrorRelationship CRDを削除します。
2. プライマリ（ソース）クラスタから TridentMirrorRelationship CRD を削除します。
3. 新しいセカンダリ（デスティネーション） PVC を確立するために、プライマリ（ソース）クラスタ上に新しい TridentMirrorRelationship CRD を作成します。

ミラーリングされたプライマリまたはセカンダリ **PVC** のサイズを変更する

PVCは通常通りサイズ変更できます。ONTAPは、データの量が現在のサイズを超える場合、デスティネーションのflevxolを自動的に拡張します。

PVCからレプリケーションを削除する

レプリケーションを削除するには、現在のセカンダリ ボリュームで次のいずれかの操作を実行します：

- セカンダリ PVC 上のMirrorRelationshipを削除します。これにより、レプリケーション関係が切断されます。
- または、spec.state フィールドを *promoted* に更新します。

PVC を削除する（以前にミラーリングされたもの）

Tridentはレプリケートされた PVC をチェックし、ボリュームの削除を試みる前にレプリケーション関係を解放します。

TMRを削除する

ミラー関係の片側でTMRを削除すると、残りのTMRは_promoted_状態に移行してから、Tridentが削除を完了します。削除対象として選択されたTMRがすでに_promoted_状態にある場合、既存のミラー関係は存在せず、TMRは削除され、Tridentはローカル PVC を *ReadWrite* に昇格します。この削除により、ONTAPのローカルボリュームのSnapMirrorメタデータが解放されます。このボリュームが将来ミラー関係で使用される場合は、新しいミラー関係を作成するときに、_established_ボリュームレプリケーション状態を持つ新しいTMRを使用する必要があります。

ONTAPがオンラインのときにミラー関係を更新する

ミラー関係は、確立された後はいつでも更新できます。 `state: promoted`または `state: reestablished`フィールドを使用して関係を更新できます。デスティネーションボリュームを通常のReadWriteボリュームに昇格する場合は、_promotedSnapshotHandle_を使用して、現在のボリュームをリストアする特定のSnapshotを指定できます。

ONTAPがオフラインのときにミラー関係を更新する

CRDを使用すると、TridentがONTAPクラスタに直接接続しなくてもSnapMirror更新を実行できます。TridentActionMirrorUpdateの次の例の形式を参照してください：

例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state`は、TridentActionMirrorUpdate CRDの状態を反映しています。*Succeeded*、*In Progress*、または_Failed_のいずれかの値を取ることができます。

CSI トポロジを使用する

Trident は、 "CSI Topology機能"を使用して、Kubernetes クラスタ内のノードにボリュームを選択的に作成して接続することができます。

概要

CSI トポロジ機能を使用すると、リージョンとアベイラビリティーゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウド プロバイダーは Kubernetes 管理者がゾーン ベースのノードを生成できるようにしています。ノードは、リージョン内の異なるアベイラビリティーゾーン、または複数のリージョンに配置できます。マルチゾーンアーキテクチャにおけるワークロードのボリュームのプロビジョニングを容易にするために、Trident は CSI トポロジを使用します。

ヒント	CSI トポロジー機能の詳細 "ここをクリックしてください。"

Kubernetes は、2 つの独自のボリューム バインディング モードを提供します：

- `VolumeBindingMode`を `Immediate`に設定すると、Tridentはトポロジを意識せずにボリュームを作成します。ボリュームのバインディングと動的プロビジョニングは、PVCの作成時に処理されます。これはデフォルトの `VolumeBindingMode`であり、トポロジ制約を強制しないクラスターに適しています。永続ボリュームは、要求元のポッドのスケジュール要件に依存せずに作成されます。
- `VolumeBindingMode`を `WaitForFirstConsumer`に設定すると、PVC の永続ボリュームの作成とバインドは、PVC を使用するポッドがスケジュールされて作成されるまで遅延されます。このようにして、トポロジ要件によって適用されるスケジュール制約を満たすボリュームが作成されます。

メモ	`WaitForFirstConsumer`バインディング モードではトポロジ ラベルは必要ありません。これは、CSI トポロジ機能とは独立して使用できます。

要件

CSI トポロジを利用するには、次のものが必要です：

- "サポートされている Kubernetes バージョン"を実行しているKubernetesクラスタ

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードにはトポロジー認識を導入するラベルが必要です（(`topology.kubernetes.io/region`および `topology.kubernetes.io/zone`）。これらのラベルは、Tridentがトポロジを認識できるようにするために、Tridentをインストールする前に*クラスタ内のノードに存在している必要があります*。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

ステップ**1**：トポロジー対応のバックエンドを作成する

Tridentストレージ バックエンドは、可用性ゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドには、サポートされているゾーンとリージョンのリストを表すオプションの `supportedTopologies`ブロックを含めることができます。このようなバックエンドを使用す

るStorageClassesの場合、ボリュームは、サポートされているリージョン/ゾーンでスケジュールされているアプリケーションによって要求された場合にのみ作成されます。

バックエンドの定義の例を次に示します：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

メモ	`supportedTopologies`は、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、StorageClassで指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むStorageClassesの場合、Tridentはバックエンドにボリュームを作成します。

`supportedTopologies`をストレージプールごとに定義することもできます。次の例を参照してください：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

この例では、 `region`および `zone`ラベルはストレージ プールの場所を表します。 `topology.kubernetes.io/region`および `topology.kubernetes.io/zone`は、ストレージ プールをどこから使用できるかを指定します。

ステップ**2**：トポロジーを認識する**StorageClasses**を定義する

クラスタ内のノードに提供されるトポロジラベルに基づいて、StorageClassesをトポロジ情報を含めるように定義できます。これにより、PVC要求の候補となるストレージプールと、Tridentによってプロビジョニングされたボリュームを利用できるノードのサブセットが決定されます。

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

上記のStorageClass定義では、 `volumeBindingMode`が `WaitForFirstConsumer`に設定されています。このStorageClassで要求されたPVCは、ポッド内で参照されるまで処理されません。また、 `allowedTopologies`は使用するゾーンとリージョンを提供します。 `netapp-san-us-east1` StorageClassは、上記で定義された `san-backend-us-east1`バックエンドにPVCを作成します。

ステップ**3**：**PVC**を作成して使用する

StorageClassが作成され、バックエンドにマップされたら、PVC を作成できるようになります。

以下の例を参照してください `spec`：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

このマニフェストを使用して PVC を作成すると、次のようになります：

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

Tridentでボリュームを作成してPVCにバインドするには、ポッド内のPVCを使用します。次の例を参照してください：

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

このpodSpecは、Kubernetesに対して、`us-east1`リージョンに存在するノードにポッドをスケジュールし、`us-east1-a`または`us-east1-b`ゾーンに存在する任意のノードから選択するように指示します。

次の出力を参照してください：

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

バックエンドを更新して `**supportedTopologies**`を含める

既存のバックエンドは、 `supportedTopologies`のリストを含めるように `tridentctl backend update`を使用して更新できます。これはすでにプロビジョニングされたボリュームには影響せず、以降のPVCにのみ使用されます。

詳細情報の参照

- "コンテナのリソースを管理する"
- "nodeSelector"
- "親和性と反親和性"
- "TaintsとTolerations"

スナップショットの操作

Kubernetes の Persistent Volume （ PV ）のボリュームスナップショットにより、ポイントインタイムのボリュームコピーが可能になります。Trident を使用して作成されたボリュームのスナップショットを作成したり、 Trident 外部で作成されたスナップショットをインポートしたり、既存のスナップショットから新しいボリュームを作成したり、スナップショットからボリュームデータをリカバリしたりできます。

概要

ボリュームスナップショットは `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `azure-netapp-files`、および `google-cloud-netapp-volumes`ドライバでサポートされています。

開始する前に

スナップショットを操作するには、外部スナップショットコントローラーとカスタムリソース定義（CRD）が必要です。これは Kubernetes オーケストレーター（例： Kubeadm 、 GKE 、OpenShift）の責任です。

Kubernetesディストリビューションにスナップショットコントローラとcrdが含まれていない場合は、ボリュームSnapshotコントローラを導入するを参照してください。

メモ	GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEは組み込みの隠しスナップショットコントローラを使用します。

ボリューム **Snapshot** を作成する

手順

1. `VolumeSnapshotClass`を作成します。詳細については、"VolumeSnapshotClass"を参照してください。
   - `driver`はTrident CSIドライバを指しています。
   - `deletionPolicy`は `Delete`または `Retain`に設定できます。 `Retain`に設定すると、`VolumeSnapshot`オブジェクトが削除された場合でも、ストレージクラスター上の基礎となる物理Snapshotは保持されます。

   例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   deletionPolicy: Delete
   ```

2. 既存の PVC のスナップショットを作成します。

   例

   - この例では、既存の PVC のスナップショットを作成します。

   ```
   cat snap.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: pvc1-snap
   spec:
     volumeSnapshotClassName: csi-snapclass
     source:
       persistentVolumeClaimName: pvc1
   ```

   - この例では、 `pvc1`という名前のPVCのボリュームスナップショットオブジェクトを作成し、スナップショットの名前は `pvc1-snap`に設定されます。VolumeSnapshotはPVCに類似しており、実際のスナップショットを表す `VolumeSnapshotContent`オブジェクトに関連付けられています。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 説明することで、 `VolumeSnapshotContent` オブジェクトを `pvc1-snap` VolumeSnapshot用に特定できます。 `Snapshot Content Name`は、このスナップショットに対応するVolumeSnapshotContentオブジェクトを識別します。 `Ready To Use`パラメータは、このスナップショットを使用して新しいPVCを作成できることを示します。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

ボリューム**Snapshot**から**PVC**を作成する

`dataSource`を使用して、`VolumeSnapshot`という名前の `<pvc-name>`をデータのソースとして使用する`PVC`を作成できます。`PVC`が作成されると、ポッドに接続して他の`PVC`と同じように使用できるようになります。

警告 PVC はソース ボリュームと同じバックエンドに作成されます。"KB：Trident PVC スナップショットからの PVC の作成は代替バックエンドでは実行できません"を参照してください。

次の例では、 `pvc1-snap`をデータソースとして使用してPVCを作成します。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

ボリューム **Snapshot** をインポートする

Tridentは"Kubernetesの事前プロビジョニングされたスナップショットプロセス"をサポートしており、クラスタ管理者が `VolumeSnapshotContent`オブジェクトを作成し、Trident外部で作成されたスナップショットをインポートできるようにします。

開始する前に

Tridentでスナップショットの親ボリュームを作成またはインポートしておく必要があります。

手順

1. *クラスタ管理者：*バックエンドスナップショットを参照する `VolumeSnapshotContent`オブジェクトを作成します。これにより、Tridentでスナップショットワークフローが開始されます。
   - `annotations`のバックエンドスナップショットの名前を `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`として指定します。
   - `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`を `snapshotHandle`で指定します。これは、外部スナップショッターが `ListSnapshots`コールでTridentに提供する唯一の情報です。

メモ	`<volumeSnapshotContentName>`は、CR の命名制約により、バックエンドのスナップショット名と常に一致するとは限りません。

     例

     次の例では、バックエンドスナップショット `snap-01`を参照する `VolumeSnapshotContent`オブジェクトを作成します。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. クラスタ管理者： `VolumeSnapshot`オブジェクトを参照するCRを作成します。 `VolumeSnapshotContent`これにより、指定されたネームスペースで `VolumeSnapshot`を使用するためのアクセスが要求されます。

   例

   次の例では、 `VolumeSnapshot`という名前のCRを作成し、 `VolumeSnapshotContent`という名前の `import-snap-content`を参照します。 `import-snap`

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. *内部処理（操作は不要）：*外部スナップショットツールは新しく作成された `VolumeSnapshotContent`を認識し、 `ListSnapshots`呼び出しを実行します。Tridentは `TridentSnapshot`を作成します。
   - 外部スナップショットは、 `VolumeSnapshotContent`を `readyToUse`に設定し、 `VolumeSnapshot`を `true`に設定します。
   - Tridentが返します `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim`を作成して新しい `VolumeSnapshot`を参照します。ここで、 `spec.dataSource`（または `spec.dataSourceRef`）名は `VolumeSnapshot`名です。

例

次の例では、 `VolumeSnapshot`という名前の `import-snap`を参照するPVCを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

スナップショットを使用してボリュームデータを回復する

`ontap-nas`および `ontap-nas-economy`ドライバーを使用してプロビジョニングされたボリュームの互換性を最大限に高めるために、スナップショットディレクトリはデフォルトでは非表示になっています。 `.snapshot`ディレクトリを有効にして、スナップショットから直接データをリカバリします。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

メモ	Snapshotコピーをリストアすると、既存のボリューム構成が上書きされます。Snapshotコピーの作成後にボリュームデータに加えられた変更は失われます。

Snapshotからのインプレースボリュームリストア

Tridentは、 `TridentActionSnapshotRestore`（TASR）CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

Tridentは、 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `google-cloud-netapp-volumes`、および `solidfire-san`ドライバでのスナップショットのリストアをサポートしています。

開始する前に

バインドされた PVC と使用可能なボリューム スナップショットが必要です。

- PVC ステータスがバインドされていることを確認します。

```
kubectl get pvc
```

- ボリューム スナップショットが使用できる状態であることを確認します。

```
kubectl get vs
```

手順

1. TASR CRを作成します。この例では、PVC `pvc1`とボリュームSnapshot `pvc1-snapshot`のCRを作成します。

   メモ　TASR CR は、PVC と VS が存在する名前空間に存在する必要があります。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. スナップショットから復元するには CR を適用します。この例では snapshot `pvc1`から復元します。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

結果

Tridentはスナップショットからデータを復元します。スナップショットの復元ステータスを確認できます：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

メモ

- ほとんどの場合、Tridentは失敗した場合に操作を自動的に再試行しません。再度操作を実行する必要があります。
- 管理者アクセス権を持たない Kubernetes ユーザーには、アプリケーション名前空間で TASR CR を作成するために、管理者からの権限付与が必要になる場合があります。

関連するスナップショットを含む **PV** を削除する

関連するスナップショットを持つ永続ボリュームを削除すると、対応するTridentボリュームは「削除中」状態に更新されます。Tridentボリュームを削除するには、ボリュームスナップショットを削除します。

ボリューム**Snapshot**コントローラを導入する

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. Snapshot Controller を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

メモ	必要に応じて `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`を開き、 `namespace`をネームスペースに更新します。

関連リンク

- "ボリュームスナップショット"
- "VolumeSnapshotClass"

ボリュームグループのスナップショットを操作する

永続ボリューム（PV）の Kubernetes ボリューム グループ スナップショット NetApp Trident は、複数のボリュームのスナップショット（ボリューム スナップショットのグループ）を作成する機能を提供します。このボリューム グループ スナップショットは、同じポイントインタイムで作成された複数のボリュームからのコピーを表します。

メモ	VolumeGroupSnapshotは、ベータ API を備えた Kubernetes のベータ機能です。Kubernetes 1.32は、VolumeGroupSnapshotに必要な最小バージョンです。

ボリュームグループの**Snapshot**を作成する

ボリュームグループスナップショットは、次のストレージドライバでサポートされます：

- `ontap-san`ドライバ - iSCSI および FC プロトコルのみで、NVMe/TCP プロトコルには使用できません。
- `ontap-san-economy` - iSCSI プロトコルのみ。
- `ontap-nas`

> メモ　ボリュームグループのスナップショットは、NetApp ASA r2 または AFX ストレージシステムではサポートされていません。

開始する前に

- Kubernetes のバージョンが K8s 1.32 以上であることを確認してください。
- スナップショットを操作するには、外部スナップショットコントローラーとカスタムリソース定義（CRD）が必要です。これは Kubernetes オーケストレーター（例： Kubeadm 、 GKE 、OpenShift）の責任です。

  Kubernetesディストリビューションに外部スナップショットコントローラとCRDが含まれていない場合は、ボリュームSnapshotコントローラを導入するを参照してください。

  > メモ　GKE 環境でオンデマンドボリュームグループスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEは組み込みの隠しスナップショットコントローラを使用します。

- スナップショットコントローラYAMLで、`CSIVolumeGroupSnapshot`機能ゲートを'true'に設定して、ボリュームグループのスナップショットが有効になっていることを確認します。
- ボリュームグループスナップショットを作成する前に、必要なボリュームグループスナップショットクラスを作成します。
- すべてのPVC/ボリュームが同じSVM上にあることを確認して、VolumeGroupSnapshotを作成できるようにします。

手順

- VolumeGroupSnapshot を作成する前に、VolumeGroupSnapshotClass を作成してください。詳細については、"VolumeGroupSnapshotClass"を参照してください。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- 既存のストレージクラスを使用して必要なラベルを持つ PVC を作成するか、これらのラベルを既存の PVC に追加します。

次の例では、 `pvc1-group-snap`をデータソースとして、 `consistentGroupSnapshot: groupA`をラベルとして使用してPVCを作成します。要件に応じてラベルのキーと値を定義します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- PVC で指定されたラベル（(`consistentGroupSnapshot: groupA`）と同じラベルを持つ VolumeGroupSnapshot を作成します。

  この例では、ボリュームグループのスナップショットを作成します：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

グループ**Snapshot**を使用してボリュームデータをリカバリする

ボリューム グループ スナップショットの一部として作成された個々のスナップショットを使用して、個々の永続ボリュームをリストアできます。ボリューム グループ スナップショットをユニットとしてリカバリすることはできません。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

メモ	Snapshotコピーをリストアすると、既存のボリューム構成が上書きされます。Snapshotコピーの作成後にボリュームデータに加えられた変更は失われます。

Snapshotからのインプレースボリュームリストア

Tridentは、 `TridentActionSnapshotRestore`（TASR）CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

詳細については、"Snapshotからのインプレースボリュームリストア"を参照してください。

関連するグループスナップショットを含む **PV** を削除する

グループボリュームのスナップショットを削除する場合：

- VolumeGroupSnapshotsは、グループ内の個々のスナップショットではなく、グループ全体として削除できます。
- PersistentVolumesにスナップショットが存在する状態で削除された場合、Tridentはそのボリュームを「削除中」状態に移動します。これは、ボリュームを安全に削除する前にスナップショットを削除する必要があるためです。
- グループ化されたスナップショットを使用してクローンが作成され、その後グループを削除する場合、split-on-clone処理が開始され、分割が完了するまでグループを削除することはできません。

ボリューム**Snapshot**コントローラを導入する

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.
yaml
```

2. Snapshot Controller を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

メモ	必要に応じて `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`を開き、 `namespace`をネームスペースに更新します。

関連リンク

- "VolumeGroupSnapshotClass"
- "ボリュームスナップショット"