Trident 24.10ドキュメント

Trident

NetApp
November 22, 2024

目次

Trident 24.10ドキュメント

リリースノート

新機能

リリースノートには、最新バージョンのTridentの新機能、拡張機能、バグ修正に関する情報が記載されています。

> インストーラ zip ファイルに含まれている Linux の tridentctl バイナリは ' テスト済みでサポートされているバージョンですzip ファイルの「 /extras 」部分に含まれている「 macos 」バイナリはテストされておらず、サポートされていないことに注意してください。

24.10の新機能

拡張機能

- Google Cloud NetApp VolumesドライバがNFSボリュームに対して一般提供されるようになり、ゾーン対応のプロビジョニングがサポートされるようになりました。
- GCPワークロードIDは、GKEを使用するGoogle Cloud NetApp VolumeのCloud Identityとして使用されます。
- LUN-SAN ONTAPドライバおよびLUN-SAN-Economyドライバに設定パラメータが追加され、ユーザがONTAP形式オプションを指定できるようになりました `formatOptions`。
- Azure NetApp Filesの最小ボリュームサイズを50GiBに縮小Azureの新しい最小サイズは、11月に一般提供される予定です。
- ONTAP NAS-EconomyドライバとONTAP SAN-Economyドライバを既存のFlexVolプールに制限する設定パラメータが追加されました `denyNewVolumePools`。
- すべてのONTAPドライバで、SVMでアグリゲートの追加、削除、名前変更が検出されるようになりました。
- 報告されたPVCサイズを使用可能にするために、LUKS LUNに18MiBのオーバーヘッドを追加。
- ONTAP - SANおよびONTAP - SAN -エコノミーノードステージとアンステージエラー処理が改善され、ステージが失敗した後にアンステージでデバイスを削除できるようになりました。
- カスタムロールジェネレータを追加しました。これにより、お客様はONTAPでTridentの最小限のロールを作成できます。
- トラブルシューティング用のロギングを追加 `lsscsi`（"問題#792"）。

Kubernetes

- Kubernetesネイティブワークフロー向けのTridentの新機能を追加：
  - データ保護
  - データ移行
  - ディザスタリカバリ
  - アプリケーションのモビリティ

    "Trident protectの詳細"です。

- TridentがKubernetes APIサーバと通信するために使用するQPS値を設定するための新しいフラグをインストーラに追加しました `--k8s_api_qps`。
- Kubernetesクラスタノード上のストレージプロトコルの依存関係を自動管理するためのフラグをインストーラに追加 `--node-prep`。Amazon Linux 2023 iSCSIストレージプロトコルとの互換性をテストおよび検証済み
- ノードの正常でないシャットダウンシナリオでのONTAP - NAS -エコノミーボリュームの強制切断のサポートが追加されました。
- 新しいnfs-nas-エコノミーONTAPボリュームでは、バックエンドオプションの使用時にqtree単位のエクスポートポリシーが使用されます `autoExportPolicy`。qtreeは、アクセス制御とセキュリティを向上させるために、公開時にノード制限のエクスポートポリシーにのみマッピングされます。アクティブなワークロードに影響を与えることなく、Tridentがすべてのノードからボリュームの公開を解除すると、既存のqtreeが新しいエクスポートポリシーモデルに切り替えられます。
- Kubernetes 1.31のサポートを追加。

実験的な機能強化

- ONTAP SANドライバでのファイバチャネルサポートのテクニカルプレビューを追加。を参照してください "ファイバチャネルのサポート"。

の修正

- * Kubernetes *：
  - Trident Helmのインストールを妨げるRancherアドミッションWebhookを修正しました（"問題#839"）。
  - Helmチャート値のアフィン変換キー()を修正しました"問題#898"。
  - 固定tridentControllerPluginNodeSelector/tridentNodePluginNodeSelectorは"true" value()では動作しません"問題#899"。
  - クローニング中に作成された一時スナップショットを削除しました（"問題#901"）。
- Windows Server 2019のサポートが追加されました。
- Trident repo()の「go mod tidy」を修正しました"問題#767"。

非推奨

- * Kubernetes：*
  - サポートされるKubernetesの最小要件を1.25に更新。
  - PODセキュリティポリシーのサポートが削除されました。

製品のブランド変更

24.10リリース以降、Astra TridentはTrident（NetApp Trident）に名称が変更されます。このブランド変更は、Tridentの機能、サポートされるプラットフォーム、相互運用性には影響しません。

24.06の変更点

拡張機能

- 重要： `limitVolumeSize` ONTAPエコノミードライバでqtree / LUNのサイズが制限されるようになりました。これらのドライバのFlexVolサイズを制御するには、新しいパラメータを使用し `limitVolumePoolSize` ます。"問題#341"()。
- 廃止されたigroupを使用している場合に、iSCSIの自己修復機能で正確なLUN IDでSCSIスキャンを開始できるようになりました（"問題#883"）。
- バックエンドが中断モードの場合でもボリュームのクローン処理とサイズ変更処理を実行できるようになりました。
- Tridentコントローラのユーザ設定のログ設定をTridentノードポッドに伝播する機能が追加されました。
- Trident for ONTAPバージョン9.15.1以降ではなく、デフォルトでRESTを使用するためのサポートが追加されました。
- 新しい永続ボリュームのONTAPストレージバックエンドでのカスタムボリューム名とメタデータのサポートが追加されました。
- NFSマウントオプションがNFSバージョン4.xを使用するように設定されている場合に、（ANF）ドライバがデフォルトでSnapshotディレクトリが自動的に有効になるように拡張されました `azure-netapp-files` 。
- NFSボリュームに対するBottlerocketのサポートが追加されました。
- Google Cloud NetApp Volumeのテクニカルプレビューのサポートを追加。

Kubernetes

- Kubernetes 1.30のサポートを追加。
- Trident DaemonSetが起動時にゾンビマウントと残留トラッキングファイルをクリーンアップする機能を追加"問題#883"()。
- LUKSボリュームを動的にインポートするためのPVCアノテーションが追加されました `trident.netapp.io/luksEncryption` （"問題#849"）。
- ANFドライバにトポロジ対応を追加。
- Windows Server 2022ノードのサポートが追加されました。

の修正

- 古いトランザクションによるTridentのインストールエラーを修正しました。
- kutes()からの警告メッセージを無視するtridentctlを修正しました"問題#892"。
- Tridentコントローラの優先度が（"問題#887"）に `0` 変更されました `SecurityContextConstraint`。
- ONTAPドライバでは、20MiB未満のボリュームサイズを使用できるようになりました（"問題[#885"）。
- ONTAP SANドライバのサイズ変更処理中にFlexVolが縮小されないようにするための固定Trident。
- NFS v4.1でのANFボリュームのインポートエラーを修正。

非推奨

- EOLのWindows Server 2019のサポートが削除されました。

24.02の変更点

拡張機能

- Cloud Identityのサポートが追加されました。
  - ANF-AzureワークロードIDを持つAKは、クラウドIDとして使用されます。
  - FSxN-AWS IAMロールを持つEKSがクラウドIDとして使用されます。
- EKSコンソールからEKSクラスタにアドオンとしてTridentをインストールするサポートが追加されました。
- iSCSIの自己修復を設定および無効にする機能（"問題#864"）。
- AWS IAMおよびSecretsManagerとの統合を可能にし、Tridentがバックアップを含むFSxボリュームを削除できるように、ONTAPドライバにFSxパーソナリティを追加（"問題#453"）。

Kubernetes

- Kubernetes 1.29のサポートを追加。

の修正

- ACPが有効になっていない場合、ACPの警告メッセージが修正されました（"問題#866"）。
- クローンがスナップショットに関連付けられている場合、ONTAPドライバのスナップショット削除中にクローンスプリットを実行する前に10秒の遅延が追加されました。

非推奨

- マルチプラットフォームイメージマニフェストからIn-Tooアテステーションフレームワークを削除しました。

23.10の変更点

の修正

- 要求された新しいサイズがontap-nasおよびontap-nas-flexgroupストレージドライバの合計ボリュームサイズよりも小さい場合、ボリュームの拡張が修正されました（"問題#834"）。
- ontap-nasおよびontap-nas-flexgroupストレージドライバのインポート時にボリュームの使用可能なサイズのみを表示するための固定ボリュームサイズ（"問題#722"）。
- ONTAP-NAS-EconomyのFlexVol名変換が修正されました。
- ノードのリブート時のWindowsノードでのTrident初期化の問題が修正されました。

拡張機能

Kubernetes

Kubernetes 1.28のサポートを追加。

Trident

- azure-netapp-filesストレージドライバでAzure Managed Identities（AMI）を使用するためのサポートが追加されました。
- ONTAP-SANドライバでNVMe over TCPのサポートが追加されました。
- ユーザによってバックエンドがSuspended状態に設定されている場合に、ボリュームのプロビジョニングを一時停止する機能が追加されました（"問題#558"）。

23.07.1の変更点

- Kubernetes：*ダウンタイムゼロのアップグレードをサポートするためのデーモンセットの削除を修正（"問題#740"）。

23.07の変更点

の修正

Kubernetes

- Tridentのアップグレードを修正し、古いポッドが終了状態で停止（"問題#740"）。
- 「transient-trident-version-pod」の定義に公差を追加（"問題#795"）。

Trident

- ノードステージング操作中にゴーストiSCSIデバイスを識別して修正するためのLUN属性を取得するときに、LUNシリアル番号が照会されるようにするためのONTAP ZAPI要求を修正しました。
- ストレージドライバコード（"問題#816"）。
- use-rest = trueを指定してONTAPドライバを使用すると、クォータのサイズが修正されました。
- ONTAP-SAN-EconomyでLUNクローンを固定作成
- パブリッシュ情報フィールドを元に戻す `rawDevicePath` 終了： `devicePath`;データの取り込みとリカバリのためのロジックを追加(場合によっては) `devicePath` フィールド。

拡張機能

Kubernetes

- 事前プロビジョニングされたSnapshotのインポートのサポートが追加されました。
- 最小限の導入とデーモン設定のLinux権限（"問題#817"）。

Trident

- 「online」ボリュームおよびSnapshotの状態フィールドが報告されなくなりました。
- ONTAPバックエンドがオフラインの場合は、バックエンドの状態を更新します（"問題#801"、 "#543"）。
- LUNシリアル番号は、ControllerVolumePublishワークフロー中に常に取得および公開されます。
- iSCSIマルチパスデバイスのシリアル番号とサイズを確認するロジックが追加されました。

- 正しいマルチパスデバイスがステージングされていないことを確認するための、iSCSIボリュームの追加検証。

実験的強化

ONTAP-SANドライバでのNVMe over TCPのテクニカルプレビューのサポートを追加。

ドキュメント

組織とフォーマットの多くの改善が行われました。

非推奨

Kubernetes

- v1beta1スナップショットのサポートが削除されました。
- CSI以前のボリュームとストレージクラスのサポートが削除されました。
- サポートされるKubernetesの最小要件を1.22に更新。

23.04の変更点

> ONTAP-SAN-*ボリュームの強制的なボリューム接続解除は、非グレースフルノードシャットダウン機能のゲートが有効になっているKubernetesバージョンでのみサポートされます。[Force detach]は、インストール時にを使用して有効にする必要があります `--enable-force-detach` Tridentインストーラのフラグ。

の修正

- Tridentのオペレータが、仕様で指定されている場合にインストールにIPv6 localhostを使用するように修正しました。
- Trident Operatorクラスタロールの権限が固定され、バンドルの権限（"問題 #799"）。
- RWXモードで複数のノードにrawブロックボリュームを接続することで問題 を修正。
- SMBボリュームのFlexGroup クローニングのサポートとボリュームインポートが修正されました。
- Tridentコントローラがすぐにシャットダウンできない問題を修正問題 しました（"問題 #811"）。
- ONTAP-SAN-*ドライバでプロビジョニングされた指定したLUNに関連付けられているすべてのigroup名を一覧表示する修正を追加しました。
- 外部プロセスを完了まで実行できるようにする修正を追加しました。
- s390アーキテクチャ（"問題 #537"）。
- ボリュームマウント処理中の誤ったログレベルを修正しました（"問題 #781"）。
- 固定電位タイプアサーションエラー（"問題 #802"）。

拡張機能

- Kubernetes：
  - Kubernetes 1.27のサポートを追加。

- LUKSボリュームのインポートのサポートが追加されました。
- ReadWriteOncePod PVCアクセスモードのサポートが追加されました。
- ノードの正常でないシャットダウン時にONTAP-SAN-*ボリュームで強制的に接続解除がサポートされるようになりました。
- すべてのontap-san-*ボリュームでノード単位のigroupを使用するようになりました。LUNはigroupにマッピングされるだけで、それらのノードにアクティブにパブリッシュされるため、セキュリティ体制が強化されます。アクティブなワークロードに影響を与えることなく既存のボリュームを安全であるとTridentが判断した場合、必要に応じて新しいigroupスキームに切り替えます（"問題 #758"）。
- Tridentで管理されていないigroupをONTAP-SAN-*バックエンドからクリーンアップし、Tridentのセキュリティを強化

- ストレージドライバontap-nas-economyとontap-nas-flexgroupに、Amazon FSxによるSMBボリュームのサポートが追加されました。
- ontap-nas、ontap-nas-economy、ontap-nas-flexgroupストレージドライバでSMB共有のサポートが追加されました。
- arm64ノードのサポートを追加しました"問題 #732"）。
- 最初にAPIサーバを非アクティブ化することで、Tridentが手順 をシャットダウンできるようになりました"問題 #811"）。
- Windowsおよびarm64ホストのクロスプラットフォームビルドサポートをMakefileに追加しました。build.mdを参照してください。

非推奨

- Kubernetes：** ONTAP-SANおよびONTAP-SAN-economyドライバ（"問題 #758"）。

23.01.1の変更点

の修正

- Tridentのオペレータが、仕様で指定されている場合にインストールにIPv6 localhostを使用するように修正しました。
- Trident Operatorクラスタロールの権限が、バンドルの権限と同期されるように修正されました "問題 #799"。
- 外部プロセスを完了まで実行できるようにする修正を追加しました。
- RWXモードで複数のノードにrawブロックボリュームを接続することで問題 を修正。
- SMBボリュームのFlexGroup クローニングのサポートとボリュームインポートが修正されました。

23.01の変更点

TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレードしてください。

の修正

- Kubernetes：Helm（"問題#783、#794"）。

拡張機能

Kubernetes

- Kubernetes 1.26のサポートを追加。
- Trident RBACのリソース利用率が全般的に向上（"問題 番号757"）。
- ホストノードで解除されたiSCSIセッションや古いiSCSIセッションを自動で検出して修正できるようになりました。
- LUKS暗号化ボリュームの拡張のサポートが追加されました。
- Kubernetes：LUKS暗号化ボリュームのクレデンシャルローテーションのサポートを追加しました。

Trident

- ONTAP 対応のAmazon FSXを使用したSMBボリュームのONTAP NASストレージドライバへのサポートが追加されました。
- SMBボリュームの使用時のNTFS権限のサポートが追加されました。
- CVSサービスレベルを使用したGCPボリュームのストレージプールのサポートが追加されました。
- FlexGroupをONTAP-NAS-flexgroupストレージドライバで作成する際のflexgroupAggregateListのオプション使用がサポートされるようになりました。
- 複数のFlexVolを管理する場合の、ONTAPとNASの両方に対応したストレージドライバのパフォーマンスが向上しました。
- すべてのONTAP NASストレージドライバに対してデータLIFの更新を有効にしました。
- Trident DeploymentとDemonSetの命名規則を更新し、ホストノードOSを反映させました。

非推奨

- Kubernetes：サポートされる最小Kubernetes数を1.21に更新
- 設定時にデータLIFを指定しないようにしてください `ontap-san` または `ontap-san-economy` ドライバ。

22.10の変更

- Trident 22.10にアップグレードする前に、次の重要な情報をお読みください。*

> **Trident 22.10 に関するの重要な情報**
>
> - TridentでKubernetes 1.25がサポートされるようになりました。Kubernetes 1.25にアップグレードする前に、Tridentを22.10にアップグレードする必要があります。
> - SAN環境では、Tridentでマルチパス構成の使用が厳密に適用されるようになりました。multipath.confファイルの推奨値はです `find_multipaths: no`。
>
> 非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

の修正

- を使用して作成されたONTAP バックエンドに固有の修正済み問題 `credentials` 22.07.0アップグレード時にフィールドがオンラインにならない（"問題 #759"）。
- **Docker**：一部の環境でDockerボリュームプラグインが起動しないという問題 が修正されました（"問題 #548" および "問題 #760"）。
- レポートノードに属するデータLIFのサブセットのみが公開されるように、ONTAP SANバックエンド固有の修正されたSLM問題 。
- ボリュームの接続時にiSCSI LUNの不要なスキャンが発生するというパフォーマンス問題 の問題が修正されました。
- Trident iSCSIワークフロー内の細分化された再試行が削除され、迅速に失敗して外部の再試行間隔が短縮されました。
- 対応するマルチパスデバイスがすでにフラッシュされている場合にiSCSIデバイスのフラッシュ時にエラーが返される修正問題 。

拡張機能

- Kubernetes：
  - Kubernetes 1.25のサポートを追加。Kubernetes 1.25にアップグレードする前に、Tridentを22.10にアップグレードする必要があります。
  - Trident Deployment and DemonSet用に別々のServiceAccount、ClusterRole、ClusterRoleBindingを追加して、今後の権限の強化を可能にしました。
  - のサポートが追加されました "ネームスペース間ボリューム共有"。
- すべてTrident `ontap-*` ストレージドライバがONTAP REST APIで機能するようになりました。
- 新しい演算子YAMLを追加しました (`bundle_post_1_25.yaml`）を使用しない場合 `PodSecurityPolicy` Kubernetes 1.25をサポートするため。
- を追加しました "LUKS暗号化ボリュームをサポートします" の場合 `ontap-san` および `ontap-san-economy` ストレージドライバ。
- Windows Server 2019ノードのサポートが追加されました。
- を追加しました "WindowsノードでのSMBボリュームのサポート" を使用する `azure-netapp-files` ストレージドライバ。
- ONTAP ドライバの自動MetroCluster スイッチオーバー検出機能が一般提供されるようになりました。

非推奨

- **Kubernetes**：サポートされている最小Kubernetesを1.20に更新。
- Astraデータストア(Aads)ドライバを削除
- のサポートが削除されました `yes` および `smart` のオプション `find_multipaths` iSCSI用にワーカーノードのマルチパスを設定する場合。

2007年22月の変更

の修正

- Kubernetes **
  - HelmまたはTrident OperatorでTridentを設定する際に、ノードセレクタのブール値と数値を処理するように問題 を修正しました。（"GitHub問題 #700"）
  - 非CHAPパスのエラーを処理する問題 を修正したため、失敗した場合kubeletが再試行されるようになりました。 "GitHub問題 #736"）

拡張機能

- CSIイメージのデフォルトレジストリとして、k8s .gcr.ioからregistry.k8s .ioに移行します
- ONTAP SANボリュームでは、ノード単位のigroupが使用され、LUNがigroupにマッピングされると同時に、これらのノードにアクティブに公開されてセキュリティ体制が強化されます。Tridentがアクティブなワークロードに影響を与えずに安全であると判断した場合、既存のボリュームは新しいigroupスキームに適宜切り替えられます。
- TridentのインストールにResourceQuotaが含まれ、PriorityClassの消費がデフォルトで制限されたときにTrident DemonSetがスケジュールされるようになりました。
- Azure NetApp Filesドライバにネットワーク機能のサポートが追加されました。（"GitHub問題 #717"）
- ONTAP ドライバにTech Previewの自動MetroCluster スイッチオーバー検出機能を追加。（"GitHub問題 #228"）

非推奨

- **Kubernetes**：サポートされる最小Kubernetes数が1.19に更新されました。
- バックエンド構成では、単一の構成で複数の認証タイプを使用できなくなりました。

削除します

- AWS CVSドライバ（22.04以降で廃止）が削除されました。
- Kubernetes
  - ノードのポッドから不要なSYS_Admin機能を削除。
  - nodeprepを単純なホスト情報とアクティブなサービス検出に減らし、作業者ノードでNFS / iSCSIサービスが利用可能になったことをベストエフォートで確認します。

ドキュメント

新しい"PODセキュリティ標準"（PSS）セクションが追加され、インストール時にTridentで有効になった権限の詳細が追加されました。

2004年10月22日の変更

ネットアップは、製品やサービスの改善と強化を継続的に行っています。ここでは、Tridentの最新機能の一部を紹介します。以前のリリースについては、を参照してください "以前のバージョンのドキュメント"。

以前のリリースの Trident からアップグレードして Azure NetApp Files を使用する場合 ' 現在 'location`config パラメータは ' 必須のシングルトンフィールドになっています

の修正

- iSCSI イニシエータ名の解析が改善されました。（"GitHub問題 #681"）
- CSI ストレージクラスのパラメータが許可されていない問題 を修正しました。（"GitHub問題 #598"）
- Trident CRD での重複キー宣言が修正されました。（"GitHub問題 #671"）
- 不正確な CSI スナップショットログを修正しました。（"GitHub問題 #629"）を選択します
- 削除したノードでボリュームを非公開にする問題 を修正しました。（"GitHub 問題 #691"）
- ブロックデバイスでのファイルシステムの不整合の処理が追加されました。（"GitHub問題 #656"）
- インストール時に「 imageRegistry 」フラグを設定するときに、自動サポートイメージをプルする問題 を修正しました。（"GitHub問題 #715"）
- Azure NetApp Filesドライバが複数のエクスポートルールを含むボリュームのクローンを作成できない問題を修正しました問題。

拡張機能

- Trident のセキュアエンドポイントへのインバウンド接続には、 TLS 1.3 以上が必要です。（"GitHub問題 #698"）
- Trident では、セキュアなエンドポイントからの応答に HSTS ヘッダーが追加されました。
- Trident では、 Azure NetApp Files の UNIX 権限機能が自動的に有効化されるようになりました。
- * Kubernetes * ： Trident のデプロイ機能は、システムノードに不可欠な優先度クラスで実行されるようになりました。（"GitHub問題 #694"）

削除します

E シリーズドライバ（ 20.07 以降無効）が削除されました。

22.01.1 の変更

の修正

- 削除したノードでボリュームを非公開にする問題 を修正しました。（"GitHub 問題 #691"）
- ONTAP API 応答でアグリゲートスペースを確保するために nil フィールドにアクセスすると、パニックが修正されました。

22.01.0 の変更

の修正

- * Kubernetes ：大規模なクラスタのノード登録バックオフ再試行時間を延長します。
- azure-NetApp-files ドライバが、同じ名前の複数のリソースによって混乱することがあるという解決済みの問題 。
- ONTAP SAN IPv6 データ LIF が角かっこで指定した場合に機能するようになりました。
- すでにインポートされているボリュームをインポートしようとすると、 EOF 問題 が返され、 PVC は保留状態になります。（"GitHub 問題 #489"）

- SolidFireボリュームでSnapshotが32個を超える場合にTridentのパフォーマンスが低下する問題が修正されました。
- SSL 証明書の作成時に SHA-1 を SHA-256 に置き換えました。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。

拡張機能

- Kubernetes の機能拡張：
  - Kubernetes 1.23 のサポートが追加されました。
  - Trident Operator または Helm 経由でインストールした場合、 Trident ポッドのスケジュールオプションを追加します。（"GitHub 問題 #651"）
- GCP ドライバでリージョン間のボリュームを許可します。（"GitHub 問題 #633"）
- Azure NetApp Filesボリュームに「unixPermissions」オプションがサポートされるようになりました。（"GitHub 問題 #666"）

非推奨

Trident REST インターフェイスは、 127.0.0.1 または [::1] アドレスでのみリスンおよびサービスを提供できます

21.10.1 の変更点

> v21.10.0 リリースには、ノードが削除されてから Kubernetes クラスタに再度追加されたときに、 Trident コントローラを CrashLoopBackOff 状態にすることができる問題があります。この問題は、 v21.10.1 (GitHub 問題 669) で修正されています。

の修正

- GCP CVS バックエンドでボリュームをインポートする際の競合状態が修正され、インポートに失敗することがありました。
- ノードを削除してから Kubernetes クラスタ（ GitHub 問題 669 ）に再度追加するときに、 Trident コントローラを CrashLoopBackOff 状態にする問題を修正しました。
- SVM 名を指定しなかった場合に問題が検出されないという問題を修正しました（ GitHub 問題 612 ）。

21.10.0 の変更点

の修正

- XFS ボリュームのクローンをソースボリュームと同じノードにマウントできない固定問題（ GitHub 問題 514 ）
- Tridentがシャットダウン時に致命的なエラーを記録する問題を修正(GitHub Issue 597)。
- Kubernetes 関連の修正：

- スナップショットを作成するときに 'ONTAP-NAS' および 'ONTAP-NAS-flexgroup ドライバ（ GitHub 問題 645 ）を使用して ' ボリュームの使用済み領域を最小リストアサイズとして返します
- ボリュームのサイズ変更後に 'Failed to expand filesystem エラーがログに記録された問題を修正しました (GitHub 問題 560)
- POD が「 Terminating 」状態で停止する可能性がある固定問題（ GitHub 問題 572 ）。
- 「 ONTAP-SAN-エコノミー 」問題がスナップショット FlexVol （ GitHub 533 ）でいっぱいになる場合があるという問題を修正しました。
- 異なるイメージを持つ固定カスタム YAML インストーラ問題（ GitHub 問題 613 ）。
- Snapshot サイズの計算方法を固定（ GitHub 問題 611 ）。
- すべてのTridentインストーラがプレーンなKubernetesをOpenShiftと識別できる問題を修正(GitHub Issue 639)。
- Kubernetes API サーバにアクセスできない場合に、 Trident オペレータが更新を停止するよう修正しました（ GitHub 問題 599 ）。

拡張機能

- GCP - CVS パフォーマンスボリュームに対する「 unixPermissions 」オプションのサポートが追加されました。
- GCP でのスケール最適化 CVS ボリュームのサポートが 600GiB から 1TiB に追加されました。
- Kubernetes 関連の機能拡張：
  - Kubernetes 1.22 のサポートが追加されました。
  - Trident の operator と Helm チャートを Kubernetes 1.22 （ GitHub 問題 628 ）と連携させるように設定
  - tridentctl images コマンドに演算子イメージを追加 (GitHub 問題 570)

実験的な機能強化

- 「 ONTAP SAN 」ドライバでのボリューム・レプリケーションのサポートを追加しました。
- 'ONTAP-NAS-flexgroup 'ONTAP-SAN' および 'ONTAP-NAS-エコノミー ' ドライバの 'tech preview* REST サポートを追加

既知の問題

ここでは、本製品の正常な使用を妨げる可能性のある既知の問題について記載します。

- TridentがインストールされているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は `helm upgrade`、クラスタをアップグレードする前に、values.yamlをに `true`設定するかコマンドに追加する `--set excludePodSecurityPolicy=true`ように更新する必要があります。 `excludePodSecurityPolicy`
- StorageClassで指定した(`fsType=""`が含まれていないボリュームには、`Trident`によって空白が適用されるように `fsType`なりました `fsType`。Tridentでは、Kubernetes 1.17以降を使用する場合、NFSボリュームに空のを指定できます `fsType`。iSCSIボリュームの場合、セキュリティコンテキストの使用を適用するときは、StorageClassで `fsGroup`を設定する必要があります `fsType`。
- 複数のTridentインスタンスでバックエンドを使用する場合は、各バックエンド構成ファイルの値

がONTAPバックエンドに対して異なるか、SolidFireバックエンドに対して異なる値を使用する `TenantName`必要があります `storagePrefix`。Tridentは、Tridentの他のインスタンスで作成されたボリュームを検出できません。ONTAPまたはSolidFireバックエンドに既存のボリュームを作成しようとすると成功します。これは、Tridentではボリューム作成が優先的な処理として処理されるためです。 `storagePrefix` `TenantName`同じバックエンドに作成されたボリュームで名前の競合が発生する可能性があります。

- Tridentをインストールし（またはTridentオペレータを使用）、を使用して `tridentctl`Tridentを管理する場合は `tridentctl`、環境変数が設定されていることを確認する必要があります `KUBECONFIG`。これは、対象となるKubernetesクラスタを指定するために必要 `tridentctl`です。複数のKubernetes環境を使用する場合は、ファイルが正確にソースされていることを確認する必要があり `KUBECONFIG`ます。
- iSCSI PVS のオンラインスペース再生を実行するには、作業者ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これは、「 discard 」を必要とする RHEL/RedHat CoreOS インスタンスに当てはまります "マウントオプション"; discard mountOption がに含まれていることを確認します "d4b9b9554fd820f43eae492d33e41167" オンラインブロックの破棄をサポートするため。
- 各KubernetesクラスタにTridentのインスタンスが複数あると、Tridentは他のインスタンスと通信できず、そのインスタンスが作成した他のボリュームを検出できません。そのため、クラスタ内で複数のインスタンスを実行すると、予期しない誤った動作が発生します。KubernetesクラスタごとにTridentのインスタンスを1つだけ配置する必要があります。
- TridentがオフラインのときにTridentベースのオブジェクトがKubernetesから削除された場合、`StorageClass`Tridentはオンラインに戻っても対応するストレージクラスをデータベースから削除しません。これらのストレージクラスは、またはREST APIを使用して削除して `tridentctl`ください。
- ユーザが、対応するPVCを削除する前にTridentでプロビジョニングされたPVを削除しても、Tridentはバッキングボリュームを自動的に削除しません。またはREST APIを使用してボリュームを削除してください `tridentctl`。
- FlexGroup では、プロビジョニング要求ごとに一意のアグリゲートセットがないかぎり、同時に複数の ONTAP をプロビジョニングすることはできません。
- IPv6経由のTridentを使用する場合は、バックエンド定義でとを `dataLIF`角かっこで指定する必要があります `managementLIF`。たとえば、 `[fd20:8b1e:b258:2000:f816:3eff:feec:0]`です。

> ONTAP SANバックエンドでは指定できません `dataLIF`。Tridentは、使用可能なすべてのiSCSI LIFを検出し、それらを使用してマルチパスセッションを確立します。

- を使用する場合 `solidfire-san` OpenShift 4.5を搭載したドライバ。基になるワーカーノードがMD5をCHAP認証アルゴリズムとして使用するようにします。Element 12.7では、FIPS準拠のセキュアなCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256が提供されています。

詳細については、こちらをご覧ください

- "Trident GitHub"
- "Tridentブログ"

以前のバージョンのドキュメント

Trident 24.10を実行していない場合は、以前のリリースのドキュメントがに基づいて提供されて"Tridentのサポートライフサイクル"います。

- "Trident 24.06"

- "Trident 24.02"
- "Trident 23.10"
- "Trident 23.07"
- "Trident 23.04"
- "Trident 23.01"
- "Trident 22.10"
- "Trident 22.07"
- "Trident 22.04"
- "Trident 22.01"

はじめに

Tridentの詳細

Tridentの詳細

Trident は、ネットアップが管理する、完全にサポートされているオープンソースプロジェクトです。Container Storage Interface（CSI）などの業界標準のインターフェイスを使用して、コンテナ化されたアプリケーションの永続性要求を満たすように設計されています。

Tridentとは

NetApp Tridentでは、ONTAP（AFF、SolidFire、Select、Cloud、Amazon FSx for NetApp ONTAP ）、Elementソフトウェア（NetApp HCI、FAS）、Azure NetApp Filesサービス、Cloud Volumes Service on Google Cloudなど、パブリッククラウドやオンプレミスで一般的なすべてのNetAppストレージプラットフォームのストレージリソースの消費と管理を実現できます。

Tridentは、コンテナストレージインターフェイス（CSI）に準拠した動的ストレージオーケストレーションツールで、とネイティブに統合され"Kubernetes"ます。Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして動作します。詳細については、を参照してください "Tridentのアーキテクチャ" 。

Tridentは、NetAppストレージプラットフォーム向けのDockerエコシステムと直接統合することもできます。NetApp Docker Volume Plugin（nDVP）は、ストレージプラットフォームからDockerホストへのストレージリソースのプロビジョニングと管理をサポートします。詳細については、を参照してください "Trident for Dockerの導入" 。

Kubernetesを初めて使用する場合は、 "Kubernetesの概念とツール"。

TridentのTest Driveプログラム

Test Driveプログラムを利用するには、すぐに使用できるラボイメージを使用して、「コンテナ化されたワークロード向けの永続的ストレージの簡単な導入とクローニング」へのアクセスをリクエストしてください"ネットアップのテスト用ドライブ"。このテストドライブでは、3ノードのKubernetesクラスタとTridentがインストールおよび設定されたサンドボックス環境を提供します。これは、Tridentに慣れ親しんでその機能を調べるための素晴らしい方法です。

もう1つのオプションは、 "kubeadm インストールガイド" Kubernetes が提供します。

これらの手順を使用して構築したKubernetesクラスタは、本番環境では使用しないでください。本番環境向けクラスタ向けに、ディストリビューションから提供されている本番環境導入ガイドを使用します。

KubernetesとNetApp製品の統合

NetAppのストレージ製品ポートフォリオは、Kubernetesクラスタのさまざまな要素と統合されているため、高度なデータ管理機能が提供され、Kubernetes環境の機能、機能、パフォーマンス、可用性が強化されます。

NetApp ONTAP 対応の Amazon FSX

"NetApp ONTAP 対応の Amazon FSX" 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起動して実行できる、フルマネージドのAWSサービスです。

Azure NetApp Files の特長

"Azure NetApp Files の特長" は、ネットアップが提供するエンタープライズクラスの Azure ファイル共有サービスです。要件がきわめて厳しいファイルベースのワークロードも、ネットアップが提供するパフォーマンスと充実のデータ管理機能を使用して、 Azure でネイティブに実行できます。

Cloud Volumes ONTAP

"Cloud Volumes ONTAP" は、クラウドで ONTAP データ管理ソフトウェアを実行するソフトウェア型ストレージアプライアンスです。

Google Cloud NetAppボリューム

"Google Cloud NetAppボリューム" Google Cloudのフルマネージドファイルストレージサービスで、ハイパフォーマンスなエンタープライズクラスのファイルストレージを提供します。

Element ソフトウェア

"要素（ Element ）" ストレージ管理者は、パフォーマンスを保証し、ストレージの設置面積を合理化することで、ワークロードを統合できます。

NetApp HCI

"NetApp HCI" 日常業務を自動化し、インフラ管理者がより重要な業務に集中できるようにすることで、データセンターの管理と拡張を簡易化します。

Trident では、コンテナ化されたアプリケーション用のストレージデバイスを、基盤となる NetApp HCI ストレージプラットフォームに直接プロビジョニングして管理できます。

NetApp ONTAP

"NetApp ONTAP" は、NetAppのマルチプロトコルユニファイドストレージオペレーティングシステムで、あらゆるアプリケーションに高度なデータ管理機能を提供します。

ONTAP システムには、オールフラッシュ、ハイブリッド、オール HDD のいずれかの構成が採用されており、自社開発のハードウェア（ FAS と AFF ）、ノーブランド製品（ ONTAP Select ）、クラウドのみ（ Cloud Volumes ONTAP ）など、さまざまな導入モデルが用意されています。Tridentは、次のONTAP導入モデルをサポートしています。

Tridentのアーキテクチャ

Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして動作します。Tridentボリュームをマウントする可能性があるホストでノードポッドが実行されている必要があります。

コントローラポッドとノードポッドについて

Tridentは、Kubernetesクラスタに1つ以上の単一または複数Tridentノードポッドとして導入されTridentコントローラポッド、標準のKUBSI_CSI Sidecar Containers_を使用してCSIプラグインの導入を簡素化します。"Kubernetes CSIサイドカーコンテナ"Kubernetes Storageコミュニティが管理しています。

Kubernetes"ノードセレクタ"を使用して、"寛容さと汚れ"ポッドを特定のノードまたは優先ノードで実行するように制限します。Tridentのインストール時に、コントローラポッドとノードポッドのノードセレクタと許容範囲を設定できます。

- コントローラプラグインは、Snapshotやサイズ変更などのボリュームのプロビジョニングと管理を処理します。
- ノードプラグインによって、ノードへのストレージの接続が処理されます。

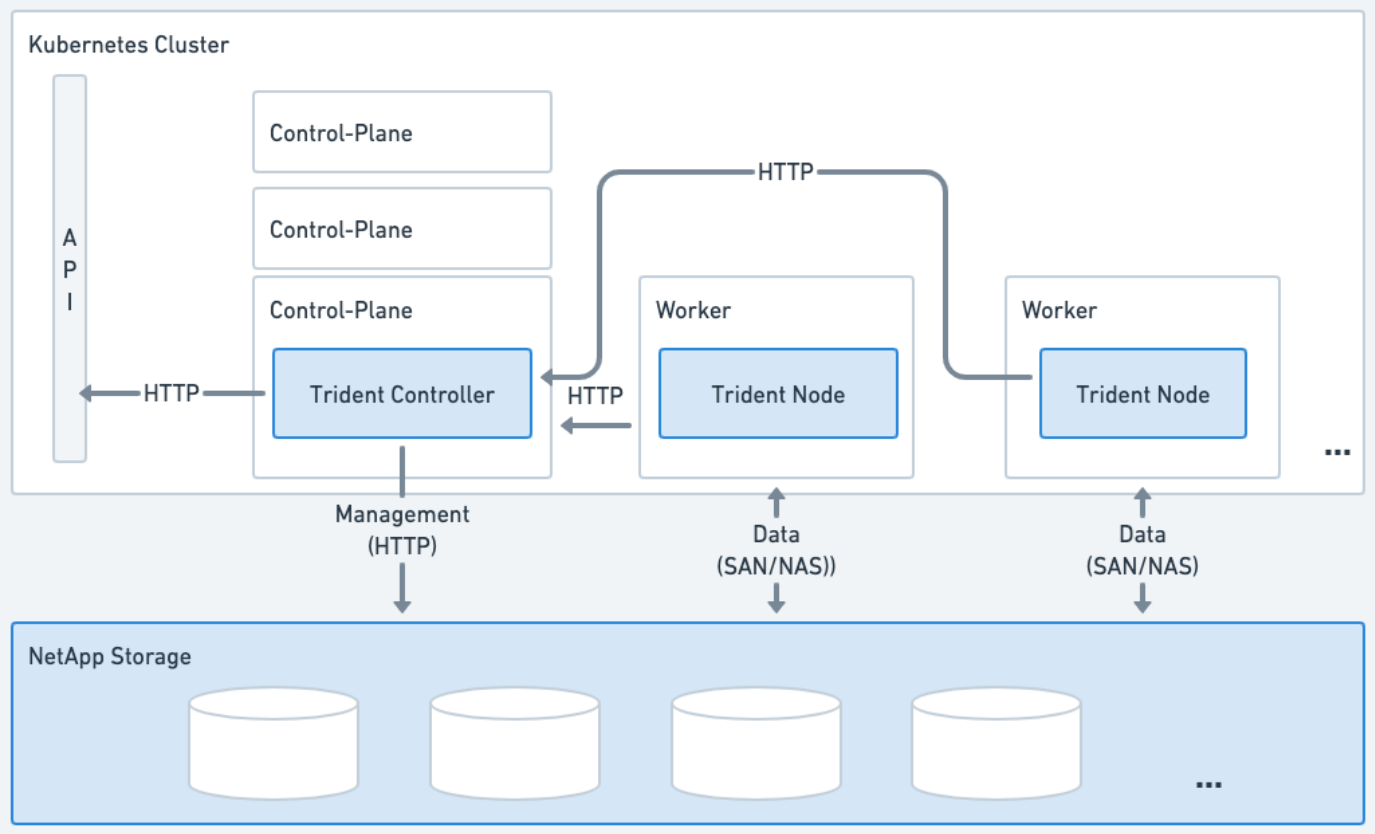


図 1. Kubernetesクラスタに導入されたTrident

Tridentコントローラポッド

Tridentコントローラポッドは、CSIコントローラプラグインを実行する単一のポッドです。

- NetAppストレージ内のボリュームのプロビジョニングと管理を担当

- Kubernetes環境で管理
- インストールパラメータに応じて、コントロールプレーンノードまたはワーカーノードで実行できます。

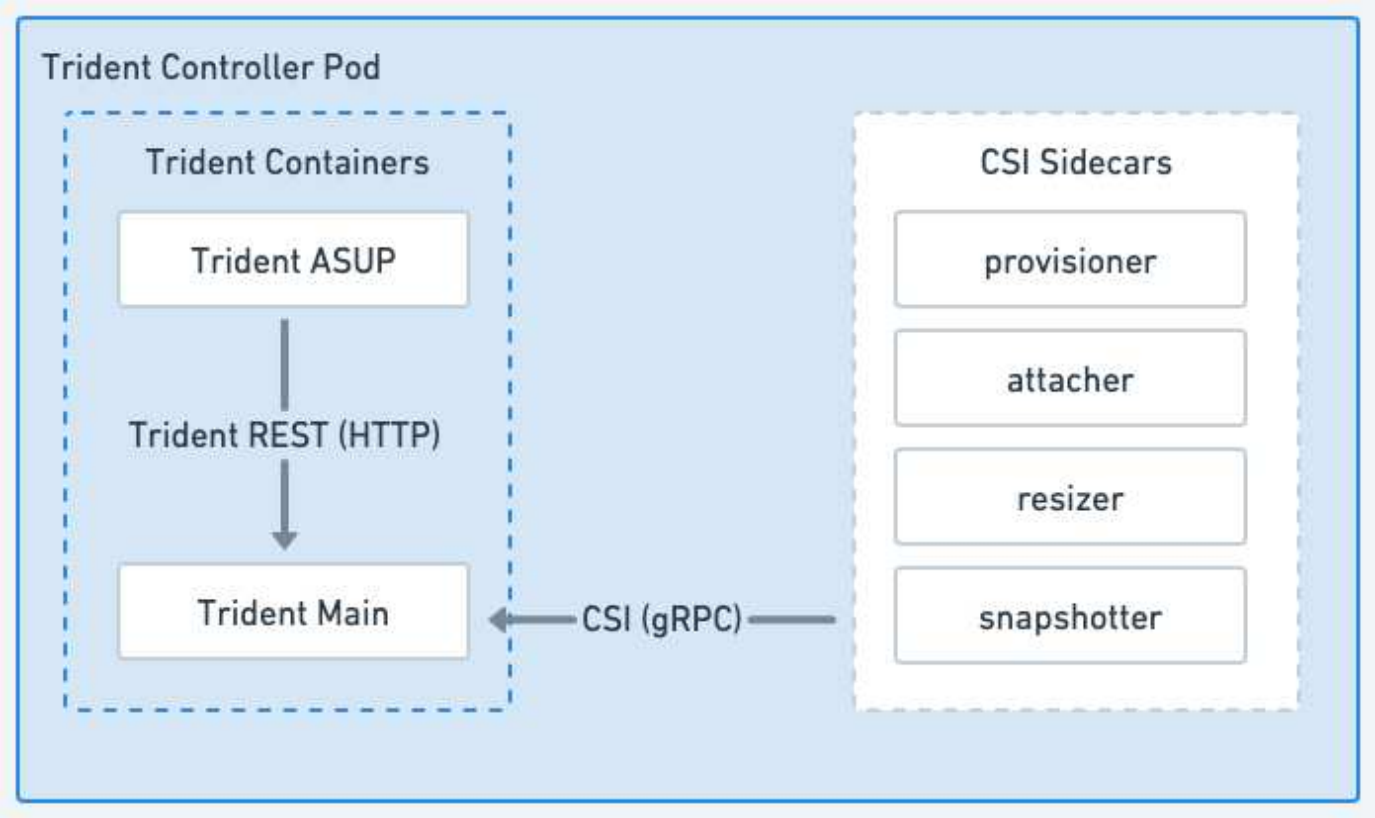


図 **2. Trident**コントローラポッドの図

Tridentノードポッド

Tridentノードポッドは、CSIノードプラグインを実行する特権ポッドです。

- ホストで実行されているPodのストレージのマウントとアンマウントを担当します。
- Kubernetesデーモンセットで管理
- NetAppストレージをマウントするすべてのノードで実行する必要がある

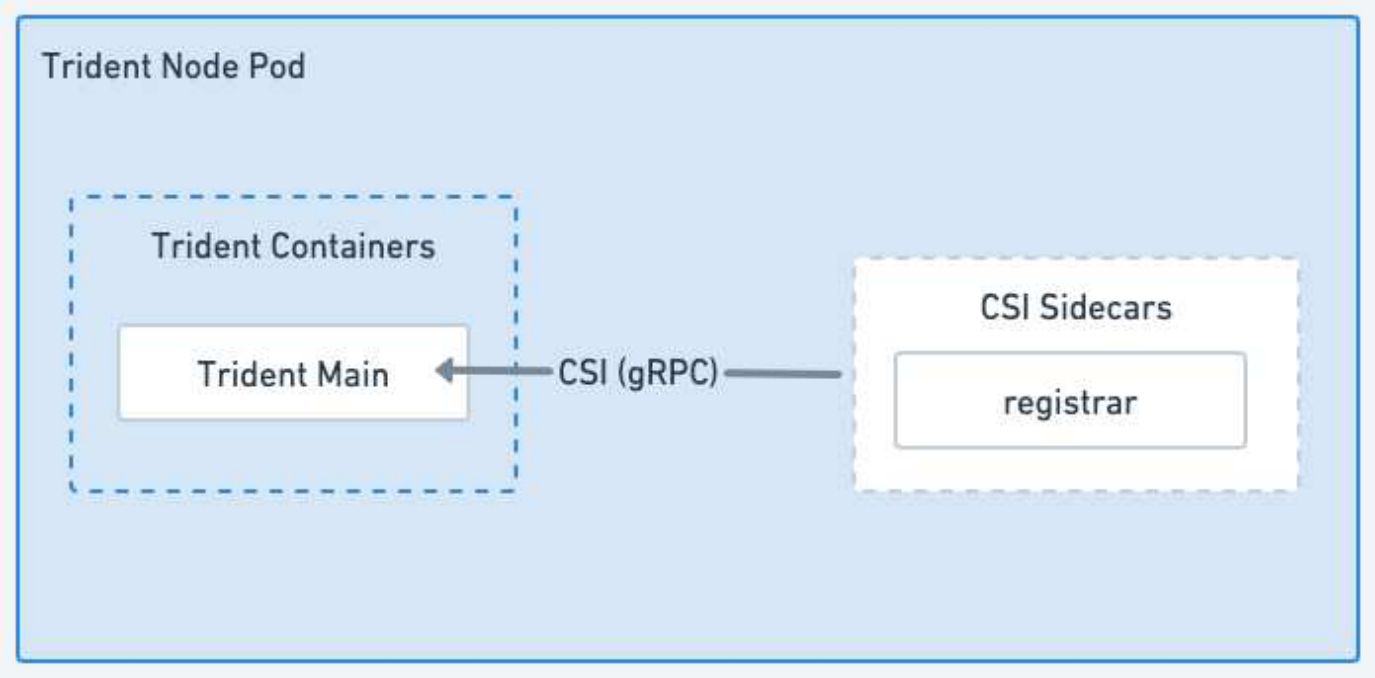


図 **3. Trident**ノードのポッド図

サポートされる **Kubernetes** クラスタアーキテクチャ

Tridentは、次のKubernetesアーキテクチャでサポートされます。

Kubernetes クラスタアーキテクチャ	サポートされます	デフォルトのインストールです
単一マスター、コンピューティング	はい。	はい。
複数のマスター、コンピューティング	はい。	はい。
マスター、「 etcd' 」、コンピューティング	はい。	はい。
マスター、インフラ、コンピューティング	はい。	はい。

概念

プロビジョニング

Tridentでのプロビジョニングには2つのフェーズがあります。最初のフェーズでは、ストレージクラスを適切なバックエンドストレージプールのセットに関連付け、プロビジョニング前の必要な準備として実行します。2番目のフェーズではボリュームの作成自体が行われ、保留状態のボリュームのストレージクラスに関連付けられたストレージプールの中からストレージプールを選択する必要があります。

ストレージクラスの関連付け

バックエンドストレージプールをストレージクラスに関連付けるには、ストレージクラスの要求された属性と、 additionalStoragePools`の `excludeStoragePools`リストの両方が `storagePools`必要です。ストレージクラスを作成すると、 Trident はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。ストレージプールの属性および名前が要求されたすべての属性およびプール名と一致すると、Tridentはそのストレージクラスに適した一連のストレージプールにそのストレージプールを追加します。さらに、Tridentは、リストに表示されているすべてのストレージプールをそのセットに追加します `additionalStoragePools。これは、ストレージクラスの要求された属性のすべてまたはいずれかを属性が満たさない場合でも同様です。このリストを使用して、ストレージクラスでのストレージプールの使用を無効にしたり削除したりする必要があり `excludeStoragePools`ます。Tridentでは、新しいバックエンドを追加するたびに同様のプロセスが実行され、そのストレージプールが既存のストレージクラスのストレージクラスを満たしているかどうかがチェックされ、除外としてマークされているものは削除されます。

ボリュームの作成

Tridentでは、ストレージクラスとストレージプールの関連付けを使用して、ボリュームのプロビジョニング先を決定します。ボリュームを作成すると、Tridentはまずそのボリュームのストレージクラスに対応する一連のストレージプールを取得します。ボリュームにプロトコルを指定すると、要求されたプロトコルを提供できないストレージプールはTridentによって削除されます（たとえば、NetApp HCI / SolidFireバックエンドではファイルベースのボリュームを提供できず、ONTAP NASバックエンドではブロックベースのボリュームを提供できません）。Tridentは、この結果セットの順序をランダム化してボリュームを均等に分散し、その順序を繰り返して各ストレージプールでボリュームのプロビジョニングを試みます。成功した場合は正常に返され、プロセスで発生したエラーが記録されます。Tridentは、要求されたストレージクラスとプロトコルで使用可能な*すべての*ストレージプールでのプロビジョニングに失敗した場合にのみ、エラー*を返します。

ボリューム **Snapshot**

Tridentがドライバのボリュームスナップショットを作成する方法の詳細については、こちらを参照してください。

ボリューム**Snapshot**の作成方法について説明します

- をクリックします ontap-nas、 ontap-san、 gcp-cvs`および `azure-netapp-files ドライバ、各永続ボリューム（PV）はFlexVol にマッピングされます。その結果、ボリューム Snapshot はネットアップ Snapshot として作成されます。NetAppのスナップショット・テクノロジーは'競合するスナップショット・テクノロジーよりも安定性'拡張性'リカバリ性'パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- をクリックします ontap-nas-flexgroup ドライバ、各永続ボリューム（PV）はFlexGroup にマッピングされます。その結果、ボリューム Snapshot は NetApp FlexGroup Snapshot として作成されます。NetAppのスナップショット・テクノロジーは'競合するスナップショット・テクノロジーよりも安定性'拡張性'リカバリ性'パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- をクリックします ontap-san-economy ドライバとPVSは、共有FlexVol上に作成されたLUNにマッピングされます。PVS のボリューム Snapshot は、関連付けられた LUN の FlexClone を実行することで実現されます。ONTAP FlexCloneテクノロジを使用すると、大規模なデータセットのコピーをほぼ瞬時に作成できます。コピーと親でデータブロックが共有されるため、メタデータに必要な分しかストレージは消費されません。
- 「 olidfire-SAN' ドライバの場合、各 PV は NetApp Element ソフトウェア / NetApp HCI クラスタ上に作成された LUN にマッピングされます。ボリューム Snapshot は、基盤となる LUN の Element Snapshot で表されます。これらの Snapshot はポイントインタイムコピーであり、消費するシステムリソースとスペ

ースはごくわずかです。

- ドライバと `ontap-san`ドライバを使用する場合、 `ontap-nas`ONTAPスナップショットはFlexVolのポイントインタイムコピーであり、FlexVol自体のスペースを消費します。その結果、ボリューム内の書き込み可能なスペースが、 Snapshot の作成やスケジュール設定にかかる時間を短縮できます。この問題に対処する簡単な方法の 1 つは、 Kubernetes を使用してサイズを変更することでボリュームを拡張することです。もう 1 つの方法は、不要になった Snapshot を削除することです。Kubernetesで作成されたボリュームSnapshotを削除すると、Tridentは関連付けられているONTAP Snapshotを削除します。Kubernetes で作成されていない ONTAP スナップショットも削除できます。

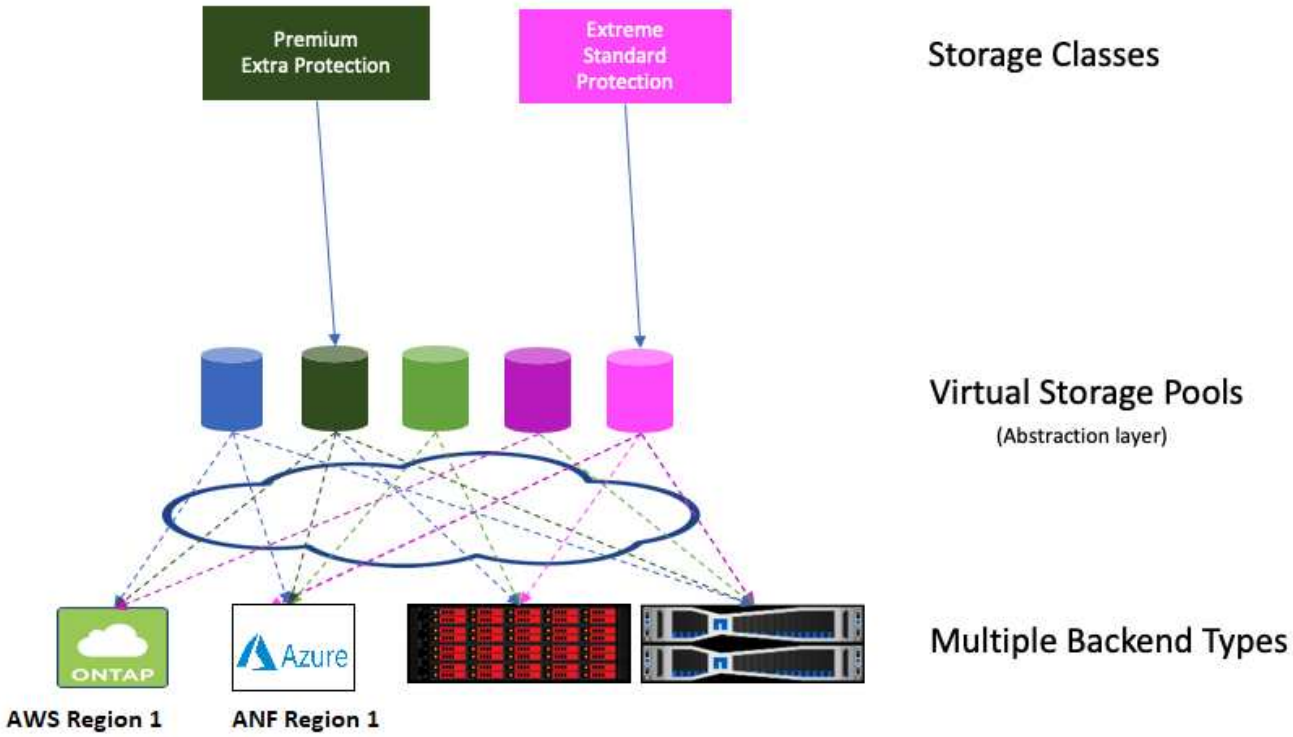
Tridentでは、ボリュームSnapshotを使用して新しいPVSを作成できます。これらの Snapshot から PVS を作成するには、サポート対象の ONTAP および CVS バックエンドに対して FlexClone テクノロジを使用します。SnapshotからPVを作成する場合、元のボリュームはSnapshotの親ボリュームのFlexCloneになります。`solidfire-san`ドライバは、Elementソフトウェアのボリュームクローンを使用してSnapshotからPVSを作成します。ここで、 Element Snapshot からクローンを作成します。

仮想プール

仮想プールは、TridentストレージバックエンドとKubernetesの間の抽象化レイヤを提供します `StorageClasses`。管理者は、必要な基準を満たすために使用する物理バックエンド、バックエンドプール、またはバックエンドタイプを指定することなく、バックエンドに依存しない共通の方法で、各バックエンドの場所、パフォーマンス、保護などの側面を定義でき `StorageClass`ます。

仮想プールについて説明します

ストレージ管理者は、任意のTridentバックエンド上の仮想プールをJSONまたはYAML定義ファイルで定義できます。



仮想プールリストの外部で指定されたすべての要素はバックエンドにグローバルであり、すべての仮想プールに適用されます。一方、各仮想プールは、 1 つまたは複数の要素を個別に指定できます（バックエンドグローバルな要素を上書きします）。

- 仮想プールを定義する場合は、バックエンド定義内の既存の仮想プールの順序を変更しないでください。
- 既存の仮想プールの属性を変更しないことをお勧めします。変更を行うには、新しい仮想プールを定義する必要があります。

ほとんどの項目はバックエンド固有の用語で指定されます。アスペクト値は、バックエンドのドライバの外部には表示されず、での照合には使用できません `StorageClasses`。代わりに、管理者が各仮想プールに 1 つ以上のラベルを定義します。各ラベルはキー：値のペアで、ラベルは一意のバックエンド間で共通です。側面と同様に、ラベルはプールごとに指定することも、バックエンドに対してグローバルに指定することもできます。名前と値があらかじめ定義されている側面とは異なり、管理者は必要に応じてラベルキーと値を定義する完全な裁量を持っています。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

A `StorageClass` セレクタパラメータ内のラベルを参照して、使用する仮想プールを指定します。仮想プールセレクタでは、次の演算子がサポートされます。

演算子	例	プールのラベル値は次のとおりです。
=	パフォーマンス = プレミアム	一致
`!= `	パフォーマンス！ = 非常に優れています	一致しません
「 in`	場所（東部、西部）	値のセットに含まれています

演算子	例	プールのラベル値は次のとおりです。
「 notin 」	パフォーマンス記名（シルバー、ブロンズ）	値のセットに含まれていません
`<key>`	保護	任意の値で存在します
`!<key>`	!保護	存在しません

ボリュームアクセスグループ

Tridentの使用方法の詳細については、こちらをご覧 "ボリュームアクセスグループ"ください。

> CHAP を使用する場合は、このセクションを無視してください。 CHAP では、管理を簡易化し、以下に説明する拡張の制限を回避することが推奨されます。また、TridentをCSIモードで使用している場合は、このセクションを無視してかまいません。Tridentは、拡張CSIプロビジョニングツールとしてインストールされている場合にCHAPを使用します。

ボリュームアクセスグループについて学習する

Tridentでは、ボリュームアクセスグループを使用して、プロビジョニングするボリュームへのアクセスを制御できます。CHAPが無効な場合は、構成で1つ以上のアクセスグループIDを指定しないかぎり、というアクセスグループが検索され `trident`ます。

Tridentは、新しいボリュームを設定済みのアクセスグループに関連付けますが、アクセスグループ自体の作成や管理は行いません。アクセスグループは、ストレージバックエンドをTridentに追加する前に存在している必要があります。また、そのバックエンドでプロビジョニングされるボリュームをマウントできる可能性があるKubernetesクラスタ内のすべてのノードのiSCSI IQNが含まれている必要があります。ほとんどのインストール環境では、クラスタ内のすべてのワーカーノードがこれに含まれます。

Kubernetes クラスタに 64 個を超えるノードがある場合は、複数のアクセスグループを使用する必要があります。各アクセスグループには最大 64 個の IQN を含めることができ、各ボリュームは 4 つのアクセスグループに属することができます。最大 4 つのアクセスグループを設定すると、クラスタ内の任意のノードから最大 256 ノードのサイズのすべてのボリュームにアクセスできるようになります。ボリュームアクセスグループの最新の制限については、を参照してください。 "こちらをご覧ください"。

デフォルト設定を使用している設定から変更する場合 `trident` 他のユーザも使用するアクセスグループには、のIDを追加します `trident` リスト内のアクセスグループ。

クイックスタートガイド（Trident）

Tridentをインストールしてストレージリソースの管理を開始するには、いくつかの手順を実行します。作業を開始する前に、を参照してください"Tridentの要件"。

> Dockerについては、を参照して"Trident for Docker"ください。

1 Tridentのインストール

Tridentには、さまざまな環境や組織に最適化されたいくつかのインストール方法とモードが用意されていま

す。

"Trident をインストール"

2 ワーカーノードを準備します

Kubernetesクラスタ内のすべてのワーカーノードが、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します"

3 バックエンドを作成します

バックエンドは、Tridentとストレージシステムの間の関係を定義します。Tridentは、そのストレージシステムとの通信方法や、Tridentがそのシステムからボリュームをプロビジョニングする方法を解説します。

"バックエンドの設定" ストレージシステム

4 **Kubernetes**ストレージクラスの作成

Kubernetes StorageClassオブジェクトでは、プロビジョニングツールとしてTridentが指定されており、カスタマイズ可能な属性を使用してボリュームをプロビジョニングするためのストレージクラスを作成できます。Tridentは、Tridentプロビジョニングツールを指定するKubernetesオブジェクト用に一致するストレージクラスを作成します。

"ストレージクラスを作成する。"

5 ボリュームをプロビジョニングする

A_PersistentVolume_（PV）は、Kubernetesクラスタ上のクラスタ管理者がプロビジョニングする物理ストレージリソースです。*PersistentVolumeClaim*（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

設定したKubernetes StorageClassを使用してPVへのアクセスを要求するPersistentVolume（PV）と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。

"ボリュームをプロビジョニングする"

次の手順

バックエンドの追加、ストレージクラスの管理、バックエンドの管理、ボリューム処理の実行が可能になりました。

要件

Tridentをインストールする前に、これらの一般的なシステム要件を確認してください。個 々 のバックエンドには追加の要件がある場合があり

Tridentに関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

サポートされるフロントエンド（オーケストレーションツール）

Tridentは、次のような複数のコンテナエンジンとオーケストレーションツールをサポートしています。

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）
- Kubernetes 1.25~1.31
- OpenShift 4.10-4.17
- Rancher Kubernetes Engine 2（RKE2）v1.28.5 + rke2r1

Trident オペレータは、次のリリースでサポートされています。

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）
- Kubernetes 1.25~1.31
- OpenShift 4.10-4.17
- Rancher Kubernetes Engine 2（RKE2）v1.28.5 + rke2r1

Tridentは、Google Kubernetes Engine（GKE）、Amazon Elastic Kubernetes Services（EKS）、Azure Kubernetes Service（AKS）、Mirantis Kubernetes Engine（MKE）、VMware Tanzu Portfolioなど、他のフルマネージド/自己管理型Kubernetesソリューションとも連携します。

TridentとONTAPは、のストレージプロバイダとして使用できます"KubeVirt"。

Tridentがインストールされ ているKubernetesクラスタを1.24から1.25以降にアップグレードする前に、を参照してください"Helmインストールのアップグレード"。

サポートされるバックエンド（ストレージ）

Tridentを使用するには、次のサポートされているバックエンドが1つ以上必要です。

- NetApp ONTAP 対応の Amazon FSX
- Azure NetApp Files の特長
- Cloud Volumes ONTAP
- Google Cloud NetAppボリューム
- FAS/AFF / Select 9.5以降
- ネットアップオール SAN アレイ（ ASA ）
- NetApp HCI / Elementソフトウェア11以降

機能の要件

次の表は、このリリースのTridentで使用できる機能と、このリリースでサポートされるKubernetesのバージョンをまとめたものです。

フィーチャー（ **Feature** ）	**Kubernetes** のバージョン	フィーチャーゲートが必要ですか？
Trident	1.25 ~ 1.31	いいえ
ボリューム Snapshot	1.25 ~ 1.31	いいえ
ボリューム Snapshot からの PVC	1.25 ~ 1.31	いいえ
iSCSI PV のサイズ変更	1.25 ~ 1.31	いいえ
ONTAP 双方向 CHAP	1.25 ~ 1.31	いいえ
動的エクスポートポリシー	1.25 ~ 1.31	いいえ
Trident のオペレータ	1.25 ~ 1.31	いいえ
CSI トポロジ	1.25 ~ 1.31	いいえ

テスト済みのホストオペレーティングシステム

Tridentは特定のオペレーティングシステムを正式にサポートしていませんが、次の機能が動作することがわかっています。

- OpenShift Container Platform（AMD64およびARM64）でサポートされているRed Hat CoreOS（RHCOS）のバージョン
- RHEL 8+（AMD64およびARM64）

  NVMe/TCPにはRHEL 9以降が必要です。

- Ubuntu 22.04以降（AMD64およびARM64）

- Windows Server 2022

デフォルトでは、Tridentはコンテナ内で実行されるため、どのLinuxワーカーでも実行されます。ただし、使用しているバックエンドに応じて、Tridentが提供するボリュームを、標準のNFSクライアントまたはiSCSIイニシエータを使用してマウントできる必要があります。

tridentctl ユーティリティーは ' これらの Linux ディストリビューションでも動作します

ホストの設定

Kubernetesクラスタ内のすべてのワーカーノードが、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、ドライバの選択に基づいてNFS、iSCSI、またはNVMeのツールをインストールする必要があります。

"ワーカーノードを準備します"

ストレージシステムの構成：

バックエンド構成でTridentを使用するには、ストレージシステムの変更が必要になる場合があります。

"バックエンドを設定"

Tridentポート

Tridentでは、通信のために特定のポートにアクセスする必要があります。

"Tridentポート"

コンテナイメージと対応する Kubernetes バージョン

エアギャップを使用したインストールでは、Tridentのインストールに必要なコンテナイメージの参照先を以下に示します。コマンドを使用し `tridentctl images`て、必要なコンテナイメージのリストを確認します。

Kubernetesのバージョン	コンテナイメージ
v1.25.0、v1.26.0、v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0	• Docker .io / NetApp / Trident：24.10.0 • docker.io / netapp/trident-autosupport：24.10 • registry.k8s.io/sig-storage/csi-provisioner：v5.1.0 • registry.k8s.io/sig-storage/csi-attacher：v4.7.0 • registry.k8s.io/sig-storage/csi-resizer：v1.12.0 • registry.k8s.io/sig-storage/csi-snapshotter：v8.1.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.12.0 • docker.io/netapp/trident-operator：24.10.0（オプション）

Trident をインストール

Tridentのインストールについて

Tridentをさまざまな環境や組織にインストールできるように、NetAppには複数のインストールオプションが用意されています。Tridentは、Tridentオペレータ（手動またはHelmを使用）またはを使用してインストールできます `tridentctl`。このトピックでは、適切なインストールプロセスを選択するための重要な情報を提供します。

Trident 24.06に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

作業を開始する前に

インストールパスに関係なく、次のものが必要です。

- サポートされているバージョンのKubernetesと機能の要件を有効にして実行されている、サポートされるKubernetesクラスタに対するすべての権限。を確認します "要件" を参照してください。
- サポートされているネットアップストレージシステムへのアクセス。
- Kubernetesワーカーノードすべてからボリュームをマウントできます。
- を搭載したLinuxホスト `kubectl` （または `oc`OpenShiftを使用している場合）Kubernetesクラスタを管理するようにインストールおよび設定します。
- 。 `KUBECONFIG` Kubernetesクラスタ構成を参照するように設定された環境変数。
- Kubernetes と Docker Enterprise を併用する場合は、 "CLI へのアクセスを有効にする手順は、ユーザが行ってください"。

に慣れていない場合は "基本概念"今こそ、そのための絶好の機会です。

インストール方法を選択します

適切なインストール方法を選択します。また、に関する考慮事項についても確認しておく必要があります "メソッド間を移動しています" 決定する前に。

Trident演算子を使用する

手動で導入する場合でも、Helmを使用する場合でも、Tridentオペレータはインストールを簡素化し、Tridentリソースを動的に管理するための優れた方法です。カスタムリソース（CR）の属性を使用する`TridentOrchestrator`こともできます"Tridentのオペレータ環境をカスタマイズ"。

Tridentオペレータには次のようなメリットがあります。

** Trident**オブジェクトの作成****

Tridentオペレータが、Kubernetesのバージョンに応じて次のオブジェクトを自動的に作成します。

- オペレータのサービスアカウント
- ClusterRoleおよびClusterRoleBindingをサービスアカウントにバインドする
- 専用のPodSecurityPolicy（Kubernetes 1.25以前用）
- 演算子自体

****リソースアカウンタビリティ****

クラスタを対象としたTridentオペレータは、Tridentインストールに関連付けられたリソースをクラスタレベルで管理します。これにより、ネームスペースを対象とした演算子を使用してクラスタを対象としたリソースを管理する際に発生する可能性のあるエラーを軽減できます。これは、自己修復とパッチ適用に不可欠です。

**** 自己回復機能****

オペレータはTridentのインストールを監視し、展開が削除された場合や誤って変更された場合などの問題に積極的に対処します。 `trident-operator-<generated-id>`CRをTridentインストールに関連付けるポッドが作成され`TridentOrchestrator`ます。これにより、クラスタ内にTridentのインスタンスが1つだけ存在し、そのセットアップを制御して、インストールが強力であることを確認できます。インストールに変更が加えられると（展開またはノードのデミスタなど）、オペレータはそれらを識別し、個別に修正します。

**** は、インストール済みの既存の**** を簡単に更新できます

既存の展開をオペレータと簡単に更新できます。を編集するだけで済みます `TridentOrchestrator` CRを使用してインストールを更新します。

たとえば、デバッグログを生成するためにTridentを有効にする必要があるシナリオを考えてみましょう。これを行うには、を `TridentOrchestrator` `true`次のように設定し `spec.debug`ます。

```
kubectl patch torc <trident-orchestrator-name> -n trident --type=merge
-p '{"spec":{"debug":true}}'
```

実行後 `TridentOrchestrator` が更新され、オペレータが既存のインストールの更新とパッチを処理します。これにより、新しいポッドが作成され、それに応じてインストールが変更される可能性があります。

****クリーン再インストール****

クラスタを対象としたTridentオペレータを使用すると、クラスタを対象としたリソースを完全に削除できます。ユーザーはTridentを完全にアンインストールして簡単に再インストールできます。

** Kubernetes**の自動アップグレード処理****

Kubernetesバージョンのクラスタをサポート対象バージョンにアップグレードすると、オペレータは既存のTridentインストールを自動的に更新し、Kubernetesバージョンの要件を満たすように変更します。

> クラスタがサポート対象外のバージョンにアップグレードされた場合、オペレータがTridentをインストールできません。Tridentがオペレータとともにすでにインストールされている場合は、サポートされていないKubernetesバージョンにTridentがインストールされていることを示す警告が表示されます。

を使用します `tridentctl`

アップグレードが必要な既存の導入環境がある場合、または導入環境を高度にカスタマイズする場合は、を検討する必要があります。これは、従来のTridentの導入方法です。

Tridentリソースのマニフェストを生成できます。これには、導入、デーモンセット、サービスアカウント、Tridentのインストール時に作成されるクラスタロールが含まれます。

> 22.04リリース以降では、Tridentをインストールするたびにに`AES`キーが再生成されなくなりました。このリリースでは、Tridentは新しいシークレットオブジェクトをインストールします。このオブジェクトは複数のインストールにまたがって保持されます。つまり、 `tridentctl`22.04では以前のバージョンのTridentをアンインストールできますが、以前のバージョンでは22.04のインストールをアンインストールできません。適切なインストール方法_を選択します。

インストールモードを選択します

組織で必要な_インストールモード_（標準、オフライン、またはリモート）に基づいて導入プロセスを決定します。

標準インストール

これはTridentをインストールする最も簡単な方法であり、ネットワーク制限が適用されないほとんどの環境で機能します。標準インストールモードでは、デフォルトのレジストリを使用して(`docker.io`、必要なTrident（およびCSI(`registry.k8s.io`）イメージを格納します。

標準モードを使用する場合、Tridentインストーラは次の処理を実行します。

- インターネット経由でコンテナイメージを取得します
- デプロイメントまたはノードデーモンセットを作成します。これにより、Kubernetesクラスタ内の対象となるすべてのノードでTridentポッドがスピンアップされます。

オフラインインストール

オフラインインストールモードは、エアギャップまたは安全な場所で必要になる場合があります。このシナリオでは、必要なTridentイメージとCSIイメージを格納するために、1つのプライベートなミラーリングされたレジストリ、または2つのミラーリングされたレジストリを作成できます。

> CSIイメージは、レジストリ設定に関係なく、1つのレジストリに存在する必要があります。

リモートインストール

次に、リモートインストールプロセスの概要を示します。

- Tridentを導入するリモートマシンに、適切なバージョンのを導入し `kubectl`ます。
- Kubernetes クラスタから構成ファイルをコピーし、リモートマシンで「 KUBECONFIG 」環境変数を設定します。
- 「 kubectl get nodes 」コマンドを開始して、必要な Kubernetes クラスタに接続できることを確認します。
- 標準のインストール手順を使用して、リモートマシンからの導入を完了します。

メソッドとモードに基づいてプロセスを選択します

決定が終わったら、適切なプロセスを選択します。

メソッド	インストールモード
Tridentのオペレータ（手動）	"標準インストール" "オフラインインストール"

メソッド	インストールモード
Tridentオペレータ（Helm）	"標準インストール" "オフラインインストール"
`tridentctl`	"標準インストールまたはオフラインインストール"

インストール方法を切り替える

インストール方法を変更することもできます。その前に、次の点を考慮してください。

- Tridentのインストールとアンインストールには、常に同じ方法を使用してください。を使用してを展開した場合は `tridentctl`、適切なバージョンのバイナリを使用してTridentをアンインストールする必要があります `tridentctl`。同様に、オペレータを使用して展開する場合は、CRを編集し、Tridentをアンインストールするように設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。
- オペレータベースの導入環境を削除してTridentの導入に使用する場合 `tridentctl`は、まず`Trident`を編集してからアンインストールするように設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。次に、とオペレータの配置を削除し `TridentOrchestrator`ます。その後、を使用してをインストールできます `tridentctl`。
- オペレータベースの手動導入環境で、HelmベースのTridentオペレータ環境を使用する場合は、最初に手動でオペレータをアンインストールしてからHelmインストールを実行する必要があります。これにより、Helm は必要なラベルとアノテーションを使用して Trident オペレータを導入できます。これを行わないと、 Helm ベースの Trident オペレータの導入が失敗し、ラベル検証エラーとアノテーション検証エラーが表示されます。を使用する場合は `tridentctl`- Helmベースの展開を使用すると、問題を発生させずに導入できます。

その他の既知の設定オプション

VMware Tanzuポートフォリオ製品にTridentをインストールする場合：

- クラスタが特権ワークロードをサポートしている必要があります。
- --kubbelet-dir` フラグは kubelet ディレクトリの場所に設定する必要があります。デフォルトでは、これは /var/vcap/data/kubelet` です。

  --kubbelet-dir' を使用して kubelet の場所を指定することは、 Trident Operator 、 Helm 、および tridentctl の展開で動作することが知られています。

Tridentオペレータを使用してインストール

Tridentオペレータを手動で導入（標準モード）

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

Trident 24.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

Tridentオペレータを手動で導入し、Tridentをインストール

レビュー "インストールの概要" インストールの前提条件を満たし、環境に適したインストールオプションを選択していることを確認します。

作業を開始する前に

インストールを開始する前に、Linuxホストにログインして、管理が機能していることを確認します。 "サポートされる Kubernetes クラスタ" 必要な権限があることを確認します。

> OpenShift では、以降のすべての例で「 kubectl 」ではなく「 OC 」を使用し、「 OC login-u SYSTEM ： admin 」または「 OC login-u kube-admin 」を実行して最初に「 *system:admin 」としてログインします。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

3. Docker Hubのイメージを使用してポッドを起動し、ポッドネットワーク経由でストレージシステムにアクセスできることを確認します。

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

手順**1**：**Trident**のインストーラパッケージをダウンロード

Tridentインストーラパッケージには、Tridentオペレータの導入とTridentのインストールに必要なすべてのものが含まれています。からTridentインストーラの最新バージョンをダウンロードして展開し"GitHubの_Assets_sectionを参照してください"ます。

```
wget https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

手順**2**：を作成します `TridentOrchestrator` **CRD**

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

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

手順**3**：**Trident**のオペレータを導入する

Tridentインストーラには、オペレータのインストールと関連オブジェクトの作成に使用できるバンドルファイルが用意されています。バンドルファイルは、デフォルト設定を使用してオペレータを導入し、Tridentをインストールするための簡単な方法です。

- クラスタでKubernetes 1.24を実行している場合は、を使用し `bundle_pre_1_25.yaml`ます。

- クラスタでKubernetes 1.25以降を実行している場合は、を使用します `bundle_post_1_25.yaml`。

作業を開始する前に

- デフォルトでは、Tridentのインストーラによって `trident` ネームスペース：状況に応じて `trident` ネームスペースが存在しません。次を使用して作成してください：

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

- オペレータを以外のネームスペースに配置する場合 `trident` 名前空間、更新 `serviceaccount.yaml`、 `clusterrolebinding.yaml` および `operator.yaml` を使用してバンドルファイルを生成します `kustomization.yaml`。

  a. を作成します `kustomization.yaml` 次のコマンドを使用して、*<bundle.yaml>* is `bundle_pre_1_25.yaml` または `bundle_post_1_25.yaml` 使用しているKubernetesのバージョンに基づきます。

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

2. operator、deployment、およびReplicaSetsが作成されたことを確認します。

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

Kubernetes クラスタには、オペレータのインスタンスが * 1 つしか存在しないようにしてください。Trident のオペレータが複数の環境を構築することは避けてください。

手順**4**：を作成します `TridentOrchestrator` **Trident**をインストール

これで、を作成してTridentをインストールできます `TridentOrchestrator`。必要に応じて、仕様内の属性を使用 `TridentOrchestrator`できます"Tridentのインストールをカスタマイズ"。

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
  nodePrep:
  - iscsi
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:24.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:24.10.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v24.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します `TridentOrchestrator` ステータス

のステータス `TridentOrchestrator` インストールが正常に完了したかどうかを示し、インストールされているTridentのバージョンが表示されます。インストール中、のステータス `TridentOrchestrator` からの変更 `Installing` 終了： `Installed`。を確認した場合は `Failed` ステータスとオペレータは単独で回復できません。 "ログをチェックしてください"。

ステータス	説明
インストール中です	オペレータはこのCRを使用してTridentをインストールしています `TridentOrchestrator`。
インストール済み	Tridentは正常にインストールされました。
アンインストール中です	オペレータはTridentをアンインストールしています。 `spec.uninstall=true`
アンインストール済み	Tridentがアンインストールされます。
失敗しました	オペレータはTridentをインストール、パッチ適用、アップデート、またはアンインストールできませんでした。オペレータは自動的にこの状態から回復しようとします。この状態が解消されない場合は、トラブルシューティングが必要です。
更新中です	オペレータが既存のインストールを更新しています。
エラー	「 TridentOrchestrator 」は使用されません。別のファイルがすでに存在します。

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.10.0        | 24.10.0        |
+----------------+----------------+
```

Tridentオペレータを手動で導入（オフラインモード）

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

Trident 24.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

Tridentオペレータを手動で導入し、Tridentをインストール

レビュー "インストールの概要" インストールの前提条件を満たし、環境に適したインストールオプションを選択していることを確認します。

作業を開始する前に

Linuxホストにログインして、管理が機能していることとを確認します "サポートされる Kubernetes クラスタ" 必要な権限があることを確認します。

OpenShift では、以降のすべての例で「 kubectl 」ではなく「 OC 」を使用し、「 OC login-u SYSTEM ： admin 」または「 OC login-u kube-admin 」を実行して最初に「 *system:admin 」としてログインします。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

3. Docker Hubのイメージを使用してポッドを起動し、ポッドネットワーク経由でストレージシステムにアクセスできることを確認します。

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

手順**1**：**Trident**のインストーラパッケージをダウンロード

Tridentインストーラパッケージには、Tridentオペレータの導入とTridentのインストールに必要なすべてのものが含まれています。からTridentインストーラの最新バージョンをダウンロードして展開し"GitHubの_Assets_sectionを参照してください"ます。

```
wget https://github.com/NetApp/trident/releases/download/v24.10.0/trident-installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

手順**2**：を作成します `TridentOrchestrator` **CRD**

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

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

手順**3**：オペレータのレジストリの場所を更新します

で `/deploy/operator.yaml`、イメージレジストリの場所を反映するように更新し `image: docker.io/netapp/trident-operator:24.10.0` ます。は "TridentとCSIの画像" 1つのレジストリまた

は別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例：

- `image: <your-registry>/trident-operator:24.10.0` イメージがすべて1つのレジストリに格納されている場合。
- `image: <your-registry>/netapp/trident-operator:24.10.0` TridentイメージがCSIイメージとは別のレジストリにある場合。

ステップ4：Tridentオペレータを導入

Tridentインストーラには、オペレータのインストールと関連オブジェクトの作成に使用できるバンドルファイルが用意されています。バンドルファイルは、デフォルト設定を使用してオペレータを導入し、Tridentをインストールするための簡単な方法です。

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

作業を開始する前に

- デフォルトでは、Tridentのインストーラによって `trident` ネームスペース：状況に応じて `trident` ネームスペースが存在しません。次を使用して作成してください：

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

- オペレータを以外のネームスペースに配置する場合 `trident` 名前空間、更新 `serviceaccount.yaml`、 `clusterrolebinding.yaml` および `operator.yaml` を使用してバンドルファイルを生成します `kustomization.yaml`。
  a. を作成します `kustomization.yaml` 次のコマンドを使用して、*<bundle.yaml>* is `bundle_pre_1_25.yaml` または `bundle_post_1_25.yaml` 使用しているKubernetesのバージョンに基づきます。

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

2. operator、deployment、およびReplicaSetsが作成されたことを確認します。

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

Kubernetes クラスタには、オペレータのインスタンスが * 1 つしか存在しないようにしてください。Trident のオペレータが複数の環境を構築することは避けてください。

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

。 "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。更新 `deploy/crds/tridentorchestrator_cr.yaml` レジストリ設定に基づいて追加の場所の仕様を追加します。

1つのレジストリ内のイメージ

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:24.10"
tridentImage: "<your-registry>/trident:24.10.0"
```

異なるレジストリ内の画像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:24.10"
tridentImage: "<your-registry>/trident:24.10.0"
```

手順**6**：を作成します `TridentOrchestrator` **Trident**をインストール

これで、を作成してTridentをインストールできます `TridentOrchestrator`。必要に応じて、仕様内の属性をさらに使用 `TridentOrchestrator`できます"Tridentのインストールをカスタマイズ"。次の例は、TridentイメージとCSIイメージが異なるレジストリにあるインストールを示しています。

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

kubectl describe torc trident

Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/trident-autosupport:24.10
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:24.10.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/trident-autosupport:24.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Http Request Timeout:       90s
    Image Pull Secrets:
    Image Registry:       <your-registry>
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
    Trident Image:        <your-registry>/trident:24.10.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v24.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します `TridentOrchestrator` ステータス

のステータス `TridentOrchestrator` インストールが正常に完了したかどうかを示し、インストールされているTridentのバージョンが表示されます。インストール中、のステータス `TridentOrchestrator` からの変更 `Installing` 終了： `Installed`。を確認した場合は `Failed` ステータスとオペレータは単独で回復できません。 "ログをチェックしてください"。

ステータス	説明
インストール中です	オペレータはこのCRを使用してTridentをインストールしています `TridentOrchestrator`。
インストール済み	Tridentは正常にインストールされました。
アンインストール中です	オペレータはTridentをアンインストールしています。 `spec.uninstall=true`
アンインストール済み	Tridentがアンインストールされます。
失敗しました	オペレータはTridentをインストール、パッチ適用、アップデート、またはアンインストールできませんでした。オペレータは自動的にこの状態から回復しようとします。この状態が解消されない場合は、トラブルシューティングが必要です。
更新中です	オペレータが既存のインストールを更新しています。
エラー	「 TridentOrchestrator 」は使用されません。別のファイルがすでに存在します。

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.10.0        | 24.10.0        |
+----------------+----------------+
```

Helm（標準モード）を使用してTridentを導入

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

Trident 24.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

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

Tridentの使用 "Helmチャート" Tridentオペレータを導入し、Tridentを一度にインストールできます。

レビュー "インストールの概要" インストールの前提条件を満たし、環境に適したインストールオプションを選択していることを確認します。

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

手順

1. Trident Helmリポジトリを追加します。

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

2. を使用し `helm install`、次の例のように導入環境の名前を指定します。 `100.2404.0`は、インストールするTridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 100.2410.0
--create-namespace --namespace <trident-namespace>
```

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

を使用できます `helm list` 名前、ネームスペース、グラフ、ステータス、アプリケーションバージョンなどのインストールの詳細を確認するには、次の手順を実行します。とリビジョン番号。

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

オプション	説明
`--values` （または `-f`)	オーバーライドを使用してYAMLファイルを指定します。これは複数回指定でき、右端のファイルが優先されます。
`--set`	コマンドラインでオーバーライドを指定します。

たとえば、のデフォルト値を変更するには `debug`、次のコマンドを実行します。は、インストールするTridentのバージョンです。 `100.2410.0`

```
helm install <name> netapp-trident/trident-operator --version 100.2410.0
--create-namespace --namespace trident --set tridentDebug=true
```

設定オプション

このテーブルと `values.yaml` Helmチャートの一部であるファイルには、キーとそのデフォルト値のリストが表示されます。

オプション	説明	デフォルト
`nodeSelector`	ポッド割り当てのノードラベル	
`podAnnotations`	ポッドの注釈	
`deploymentAnnotations`	配置のアノテーション	
`tolerations`	ポッド割り当ての許容値	

オプション	説明	デフォルト

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

`tridentControllerPluginNodeSelector`	ポッド用の追加のノードセレクタ。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentNodePluginNodeSelector`	ポッド用の追加のノードセレクタ。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	
`tridentNodePluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください [コントローラポッドとノードポッドについて] を参照してください。	

オプション	説明	デフォルト
「 imageRegistry 」と入力します	、、 `trident`およびその他のイメージのレジストリを指定します `trident-operator`。デフォルトをそのまま使用する場合は、空のままにします。重要：プライベートリポジトリにTridentをインストールする場合、スイッチを使用してリポジトリの場所を指定する場合は `imageRegistry`、リポジトリパスにはを使用しないで`/netapp/`ください。	""
`imagePullPolicy`	のイメージプルポリシーを設定します `trident-operator`。	`IfNotPresent`
「 imagePullSecrets 」	のイメージプルシークレットを設定します `trident-operator`、`trident`、およびその他の画像。	
「 kubeletDir 」を参照してください	kubeletの内部状態のホスト位置を上書きできます。	`"/var/lib/kubelet"`
`operatorLogLevel`	Tridentオペレータのログレベルを次のように設定できます。`trace`、 `debug`、 `info`、 `warn`、`error`または `fatal`。	`"info"`
`operatorDebug`	Tridentオペレータのログレベルをdebugに設定できます。	「真」
`operatorImage`	のイメージを完全に上書きできます `trident-operator`。	""
`operatorImageTag`	のタグを上書きできます `trident-operator` イメージ（Image）：	""
`tridentIPv6`	IPv6クラスタでのTridentの動作を有効にできます。	「偽」
`tridentK8sTimeout`	ほとんどのKubernetes API処理でデフォルトの30秒タイムアウトを上書きします（0以外の場合は秒単位）。	`0`
`tridentHttpRequestTimeout`	HTTP要求のデフォルトの90秒タイムアウトをで上書きします `0s` タイムアウトの期間は無限です。負の値は使用できません。	`"90s"`
`tridentSilenceAutosupport`	Trident定期AutoSupportレポートをディセーブルにできます。	「偽」

オプション	説明	デフォルト
tridentAutosupportImageTag	Trident AutoSupportコンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Trident AutoSupportコンテナがHTTPプロキシ経由で自宅に電話できるようにします。	""
tridentLogFormat	Tridentロギング形式を設定し(text`ます。または `json)	"text"
tridentDisableAuditLog	Trident監査ロガーをディセーブルにします。	「真」
tridentLogLevel	Tridentのログレベルを、、 debug info、、 warn、 `error`または `fatal`に設定 `trace`できます。	"info"
tridentDebug	Tridentのログレベルをに設定できます debug。	「偽」
tridentLogWorkflows	特定のTridentワークフローのトレースロギングまたはログ抑制を有効にできます。	""
tridentLogLayers	トレースロギングまたはログ抑制に対して特定のTridentレイヤをイネーブルにできます。	""
「 tridentImage 」のように入力します	Tridentのイメージを完全に上書きできます。	""
tridentImageTag	Tridentのイメージのタグを上書きできます。	""
tridentProbePort	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
windows	TridentをWindowsワーカーノードにインストールできるようにします。	「偽」
enableForceDetach	強制切り離し機能を有効にできます。	「偽」
excludePodSecurityPolicy	オペレータポッドのセキュリティポリシーを作成から除外します。	「偽」
cloudProvider	をに設定します "Azure" AKSクラスタで管理対象IDまたはクラウドIDを使用する場合。EKSクラスタでクラウドIDを使用する場合は、「aws」に設定します。	""

オプション	説明	デフォルト
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（「azure.workload.identity/client-id：xxxxxxxxxx-xxxx-xxxxxxx」）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（「eks.amazonaws.com/role-arn: arn：aws：iam：：123456：role / Trident -role」）に設定されます。	""
`iscsiSelfHealingInterval`	iSCSIの自己修復が実行される間隔。	`5m0s`
`iscsiSelfHealingWaitTime`	iSCSIの自己修復が、ログアウトとその後のログインを実行して古いセッションの解決を開始するまでの時間。	`7m0s`
`nodePrep`	指定したデータストレージプロトコルを使用してボリュームを管理できるように、Trident でKubernetesクラスタのノードを準備できるようにします。現在 `**iscsi**`サポートされている値は、のみです。	

コントローラポッドとノードポッドについて

Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして動作します。Tridentボリュームをマウントする可能性があるホストでノードポッドが実行されている必要があります。

Kubernetes "ノードセレクタ" および "寛容さと汚れ" は、特定のノードまたは優先ノードで実行されるようにポッドを制限するために使用されます。「ControllerPlugin」およびを使用します `NodePlugin`を使用すると、拘束とオーバーライドを指定できます。

- コントローラプラグインは、Snapshotやサイズ変更などのボリュームのプロビジョニングと管理を処理します。
- ノードプラグインによって、ノードへのストレージの接続が処理されます。

Helm（オフラインモード）を使用したTridentのオペレータの導入

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

Trident 24.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident に関するの重要な情報

- TridentでKubernetes 1.31がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

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

Tridentの使用 "Helmチャート" Tridentオペレータを導入し、Tridentを一度にインストールできます。

レビュー "インストールの概要" インストールの前提条件を満たし、環境に適したインストールオプションを選択していることを確認します。

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

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

手順

1. Trident Helmリポジトリを追加します。

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

2. を使用し `helm install` 、展開とイメージレジストリの場所の名前を指定します。は "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。この例では、 `100.2410.0`はインストールするTridentのバージョンです。

1つのレジストリ内のイメージ

```
helm install <name> netapp-trident/trident-operator --version
100.2410.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

異なるレジストリ内の画像

```
helm install <name> netapp-trident/trident-operator --version
100.2410.0 --set imageRegistry=<your-registry> --set
operatorImage=<your-registry>/trident-operator:24.10.0 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:24.06
--set tridentImage=<your-registry>/trident:24.10.0 --create
-namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```

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

を使用できます `helm list` 名前、ネームスペース、グラフ、ステータス、アプリケーションバージョンなどのインストールの詳細を確認するには、次の手順を実行します。とリビジョン番号。

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

オプション	説明
`--values` （または `-f`)	オーバーライドを使用してYAMLファイルを指定します。これは複数回指定でき、右端のファイルが優先されます。
`--set`	コマンドラインでオーバーライドを指定します。

たとえば、のデフォルト値を変更するには `debug`、次のコマンドを実行します。は、インストールするTridentのバージョンです。 `100.2410.0`

```
helm install <name> netapp-trident/trident-operator --version 100.2410.0
--create-namespace --namespace trident --set tridentDebug=true
```

nodePrep値を追加するには、次のコマンドを実行します。

```
helm install <name> netapp-trident/trident-operator --version 100.2406.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

設定オプション

このテーブルと `values.yaml` Helmチャートの一部であるファイルには、キーとそのデフォルト値のリストが表示されます。

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

オプション	説明	デフォルト
`nodeSelector`	ポッド割り当てのノードラベル	
`podAnnotations`	ポッドの注釈	
`deploymentAnnotations`	配置のアノテーション	
`tolerations`	ポッド割り当ての許容値	

オプション	説明	デフォルト

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

`tridentControllerPluginNodeSelector`	ポッド用の追加のノードセレクタ。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
`tridentControllerPluginTolerations`	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	

オプション	説明	デフォルト
tridentNodePluginNodeSelector	ポッド用の追加のノードセレクタ。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
tridentNodePluginTolerations	ポッドに対するKubernetesの許容範囲を上書きします。を参照してください "コントローラポッドとノードポッドについて" を参照してください。	
「 imageRegistry 」と入力します	、、 trident`およびその他のイメージのレジストリを指定します `trident-operator。デフォルトをそのまま使用する場合は、空のままにします。重要：プライベートリポジトリにTridentをインストールする場合、スイッチを使用してリポジトリの場所を指定する場合は imageRegistry、リポジトリパスにはを使用しないで `/netapp/`ください。	""
imagePullPolicy	のイメージプルポリシーを設定します trident-operator。	IfNotPresent
「 imagePullSecrets 」	のイメージプルシークレットを設定します trident-operator、trident、およびその他の画像。	
「 kubeletDir 」を参照してください	kubeletの内部状態のホスト位置を上書きできます。	"/var/lib/kubelet"
operatorLogLevel	Tridentオペレータのログレベルを次のように設定できます。trace、 debug、 info、 warn、error`または `fatal。	"info"
operatorDebug	Tridentオペレータのログレベルをdebugに設定できます。	「真」
operatorImage	のイメージを完全に上書きできます trident-operator。	""
operatorImageTag	のタグを上書きできます trident-operator イメージ（Image）：	""
tridentIPv6	IPv6クラスタでのTridentの動作を有効にできます。	「偽」
tridentK8sTimeout	ほとんどのKubernetes API処理でデフォルトの30秒タイムアウトを上書きします（0以外の場合は秒単位）。	0

オプション	説明	デフォルト
tridentHttpRequestTimeout	HTTP要求のデフォルトの90秒タイムアウトをで上書きします 0s タイムアウトの期間は無限です。負の値は使用できません。	"90s"
tridentSilenceAutosupport	Trident定期AutoSupportレポートをディセーブルにできます。	「偽」
tridentAutosupportImageTag	Trident AutoSupportコンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Trident AutoSupportコンテナがHTTPプロキシ経由で自宅に電話できるようにします。	""
tridentLogFormat	Tridentロギング形式を設定し(text`ます。または `json)	"text"
tridentDisableAuditLog	Trident監査ロガーをディセーブルにします。	「真」
tridentLogLevel	Tridentのログレベルを、、 debug info、、 warn、 `error`または `fatal`に設定 `trace`できます。	"info"
tridentDebug	Tridentのログレベルをに設定できます debug。	「偽」
tridentLogWorkflows	特定のTridentワークフローのトレースロギングまたはログ抑制を有効にできます。	""
tridentLogLayers	トレースロギングまたはログ抑制に対して特定のTridentレイヤをイネーブルにできます。	""
「 tridentImage 」のように入力します	Tridentのイメージを完全に上書きできます。	""
tridentImageTag	Tridentのイメージのタグを上書きできます。	""
tridentProbePort	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
windows	TridentをWindowsワーカーノードにインストールできるようにします。	「偽」
enableForceDetach	強制切り離し機能を有効にできます。	「偽」
excludePodSecurityPolicy	オペレータポッドのセキュリティポリシーを作成から除外します。	「偽」

オプション	説明	デフォルト
`nodePrep`	指定したデータストレージプロトコルを使用してボリュームを管理できるように、Trident でKubernetesクラスタのノードを準備できるようにします。現在 `**iscsi**`サポートされている値は、のみです。	

Tridentオペレータのインストールをカスタマイズ

Trident演算子を使用すると、仕様の属性を使用してTridentのインストールをカスタマイズでき `TridentOrchestrator`ます。引数で許可される範囲を超えてインストールをカスタマイズする場合 `TridentOrchestrator`は、を使用してカスタムYAMLマニフェストを生成し、必要に応じて変更することを検討して `tridentctl`ください。

コントローラポッドとノードポッドについて

Tridentは、単一のコントローラポッドと、クラスタ内の各ワーカーノード上のノードポッドとして動作します。Tridentボリュームをマウントする可能性があるホストでノードポッドが実行されている必要があります。

Kubernetes "ノードセレクタ" および "寛容さと汚れ" は、特定のノードまたは優先ノードで実行されるようにポッドを制限するために使用されます。「ControllerPlugin」およびを使用します `NodePlugin`を使用すると、拘束とオーバーライドを指定できます。

- コントローラプラグインは、Snapshotやサイズ変更などのボリュームのプロビジョニングと管理を処理します。
- ノードプラグインによって、ノードへのストレージの接続が処理されます。

設定オプション

> `spec.namespace`は、Tridentがインストールされているネームスペースを示すためににに指定されてい `TridentOrchestrator`ます。このパラメータ*は、Tridentのインストール後は更新できません*。これを実行しようとすると、 `TridentOrchestrator`ステータスがに変わり `Failed`ます。Tridentをネームスペース間で移行することは想定されていません。

このテーブルの詳細 `TridentOrchestrator` 属性。

パラメータ	説明	デフォルト
`namespace`	Tridentをインストールするネームスペース	`"default"`
「バグ」	Tridentのデバッグを有効にする	「偽」
`enableForceDetach`	`ontap-san`、 `ontap-san-economy`および `ontap-nas-economy`のみ。KubernetesのNon-Graceful Node Shutdown（NGN）と連携して、ノードに障害が発生した場合に、マウントされたボリュームを含むワークロードを新しいノードに安全に移行する機能をクラスタ管理者に提供します。	「偽」

パラメータ	説明	デフォルト
`windows`	をに設定します `true` Windowsワーカーノードへのインストールを有効にします。	「偽」
`cloudProvider`	をに設定します `"Azure"` AKSクラスタで管理対象IDまたはクラウドIDを使用する場合。EKSクラスタでクラウドIDを使用する場合は、「aws」に設定します。	`""`
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（「azure.workload.identity/client-id：xxxxxxxxx-xxxx-xxxxxxx」）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（「eks.amazonaws.com/role-arn: arn：aws：iam：：123456：role / Trident -role」）に設定されます。	`""`
`IPv6`	IPv6経由のTridentのインストール	いいえ
`k8sTimeout`	Kubernetes 処理のタイムアウト	`30sec`
`silenceAutosupport`	AutoSupportバンドルをNetAppに送信しない自動	「偽」
「 autosupportImage 」を参照してください	AutoSupport テレメトリのコンテナイメージ	`"netapp/trident-autosupport:24.10"`
「 autosupportProxy 」と入力します	AutoSupportを送信するためのプロキシのアドレス/ポート テレメータ	`"http://proxy.example.com:8888"`
uninstall	Tridentのアンインストールに使用するフラグ	「偽」
`logFormat`	使用するTridentログ形式[text、json]	`"text"`
「 tridentImage 」のように入力します	インストールするTridentイメージ	`"netapp/trident:24.10"`
「 imageRegistry 」と入力します	形式の内部レジストリへのパス `<registry FQDN>[:port][/subpath]`	`"k8s.gcr.io"`（Kubernetes 1.19以降）または `"quay.io/k8scsi"`
「 kubeletDir 」を参照してください	ホスト上の kubelet ディレクトリへのパス	`"/var/lib/kubelet"`
`wipeout`	Tridentを完全に削除するために削除するリソースのリスト	
「 imagePullSecrets 」	内部レジストリからイメージをプルするシークレット	
`imagePullPolicy`	Tridentオペレータのイメージプルポリシーを設定します。有効な値は次のとおりです。 `Always` 常にイメージをプルする。 `IfNotPresent` ノード上にイメージが存在しない場合にのみ取得します。 `Never` 画像を絶対に引き出さないでください。	`IfNotPresent`
controller’PluginNodeSelector	ポッド用の追加のノードセレクタ。 の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです

パラメータ	説明	デフォルト
controllerPluginTolerations	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです
「 nodePluginNodeSelector 」	ポッド用の追加のノードセレクタ。の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです
「 nodePluginTolerations` 」	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです
`nodePrep`	指定したデータストレージプロトコルを使用してボリュームを管理できるように、TridentでKubernetesクラスタのノードを準備できるようにします。現在 **`iscsi`** サポートされている値は、のみです。	

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

フォースデタッチの詳細

[強制切り離し（Force detach）]は、 `ontap-san-economy`および `onatp-nas-economy`でのみ使用でき `ontap-san`ます。強制接続解除を有効にする前に、Kubernetesクラスタで非グレースフルノードシャットダウン（NGN）を有効にする必要があります。詳細については、を参照してください "Kubernetes：正常なノードシャットダウンではありません"。

ドライバを使用する場合 `ontap-nas-economy`は、管理対象のエクスポートポリシーを使用してtaintが適用されたKubernetesノードからのアクセスをTridentが制限できるように、バックエンド構成のパラメータをに `true`設定する必要 `autoExportPolicy`があります。

TridentはKubernetes NGNに依存しているため、許容できないすべてのワークロードのスケジュールを再設定するまで、正常でないノードからテイントを削除しないで `out-of-service`ください。汚染を無謀に適用または削除すると、バックエンドのデータ保護が危険にさらされる可能性があります。

Kubernetesクラスタ管理者がtaintをノードに適用し、 `enableForceDetach`をに設定する `true`と `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute`、Tridentはノードのステータスを確認し、次の処理を行います。

1. そのノードにマウントされたボリュームのバックエンドI/Oアクセスを停止します。
2. Tridentノードオブジェクトを（新しいパブリケーションに対しては安全ではない）としてマークします `dirty`。

Tridentコントローラは、Tridentノードポッドによって（とマークされた後で）ノードが再修飾されるまで、新しいパブリッシュボリューム要求を拒否し `dirty`ます。マウントされたPVCを使用してスケジュールされたワークロード（クラスタノードが正常で準備が完了したあとも）は、Tridentがそのノードを検証できるようになるまで受け入れられません `clean`（新しいパブリケーションに対して安全）。

ノードの健常性が回復してtaintが削除されると、Tridentは次の処理を実行します。

1. ノード上の古い公開パスを特定してクリーンアップします。
2. ノードが状態（アウトオブサービス状態が削除され、ノードが `Ready`状態）で、古い公開パスがすべてクリーンである場合、 `cleanable`Tridentはノードをとして再登録し、新しい公開ボリュームをそのノードに許可します `clean。

構成例

次の属性を使用できます： [設定オプション] テイギスルバアイ `TridentOrchestrator` をクリックして、インストールをカスタマイズします。

基本的なカスタム設定

これは、基本的なカスタムインストールの例です。

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

ノードセレクタ

この例では、ノードセレクタを使用してTridentをインストールします。

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

Windowsワーカーノード

この例では、WindowsワーカーノードにTridentをインストールします。

```
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true
```

AKSクラスタ上の管理対象**ID**

この例では、AKSクラスタで管理対象IDを有効にするためにTridentをインストールします。

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

AKSクラスタ上のクラウド**ID**

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

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

EKSクラスタ上のクラウド**ID**

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

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/trident-role'"
```

GKEのクラウド**ID**

この例では、GKEクラスタにクラウドIDで使用するTridentをインストールします。

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

tridentctlを使用してインストールします

tridentctlを使用してインストールします

を使用してTridentをインストールでき `tridentctl`ます。このプロセスは、`Trident`に必要なコンテナイメージがプライベートレジストリに保存されているかどうかにかかわらず、インストールに適用されます。配置をカスタマイズするには `tridentctl`、を参照してください"tridentctl 展開をカスタマイズします"。

Trident 24.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

Trident **に関する**の重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart` multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

を使用して**Trident**をインストール `tridentctl`

レビュー "インストールの概要" インストールの前提条件を満たし、環境に適したインストールオプションを選択していることを確認します。

作業を開始する前に

インストールを開始する前に、Linuxホストにログインして、管理が機能していることを確認します。 "サポートされる Kubernetes クラスタ" 必要な権限があることを確認します。

> OpenShift では、以降のすべての例で「 kubectl 」ではなく「 OC 」を使用し、「 OC login-u SYSTEM ： admin 」または「 OC login-u kube-admin 」を実行して最初に「 *system:admin 」としてログインします。

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

3. Docker Hubのイメージを使用してポッドを起動し、ポッドネットワーク経由でストレージシステムにアクセスできることを確認します。

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

手順**1**：**Trident**のインストーラパッケージをダウンロード

Tridentインストーラパッケージは、Tridentポッドを作成し、その状態を維持するために使用されるCRDオブジェクトを設定し、CSIサイドカーを初期化して、ボリュームのプロビジョニングやクラスタホストへの接続などのアクションを実行します。からTridentインストーラの最新バージョンをダウンロードして展開し"GitHubの_Assets_sectionを参照してください"ます。この例では、選択したTridentバージョンで_< Tridentインストーラ-XX.X.X.tar.gz>_を更新します。

```
wget https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

手順**2**：**Trident**をインストールする

コマンドを実行して、目的のネームスペースにTridentをインストールし `tridentctl install`ます。追加の引数を追加して、イメージのレジストリの場所を指定できます。

標準モード

```
./tridentctl install -n trident
```

1つのレジストリ内のイメージ

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:24.10 --trident
-image <your-registry>/trident:24.10.0
```

異なるレジストリ内の画像

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:24.10 --trident
-image <your-registry>/trident:24.10.0
```

インストールステータスは次のようになります。

```
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=24.10.0
INFO Trident installation succeeded.
....
```

インストールを確認します。

ポッドの作成ステータスまたはを使用して、インストールを確認できます `tridentctl`。

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-679648bd45-cv2mx   6/6     Running   0          5m29s
trident-node-linux-vgc8n              2/2     Running   0          5m29s
```

インストーラが正常に完了しない場合、または `trident-controller-<generated id>` (`trident-csi-<generated id>` 23.01より前のバージョンでは、* RUNNING *ステータスがありません。プラットフォームはインストールされませんでした。使用 `-d` 終了： "デバッグモードをオンにします" および問題 のトラブルシューティングを行います。

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 24.10.0        | 24.10.0        |
+----------------+----------------+
```

構成例

次の例は、を使用してTridentをインストールする場合の設定例 `tridentctl`です。

Windowsノオト

WindowsノードでTridentを実行するには、次の手順を実行します。

```
tridentctl install --windows -n trident
```

強制的に切り離し

強制切り離しの詳細については、を参照してください "Tridentオペレータのインストールをカスタマイズ"。

```
tridentctl install --enable-force-detach=true -n trident
```

tridentctlのインストールをカスタマイズします

Tridentインストーラを使用して、インストールをカスタマイズできます。

インストーラの詳細を確認してください

Tridentインストーラでは、属性をカスタマイズできます。たとえば、Tridentイメージをプライベートリポジトリにコピーした場合は、を使用してイメージ名を指定できます `--trident-image`。Tridentイメージと必要なCSIサイドカーイメージをプライベートリポジトリにコピーした場合は、次の形式のスイッチを使用してリポジトリの場所を指定することをお勧めし `--image-registry`ます。 `<registry FQDN>[:port]`

> プライベートリポジトリにTridentをインストールするときに、スイッチを使用してリポジトリの場所を指定する場合は `--image-registry`、リポジトリパスにを使用しないで `/netapp/`ください。例： `./tridentctl install --image-registry <image-registry> -n <namespace>`

通常の「 /var/lib/kubelet 」以外のパスに「 kubelet 」がデータを保持している Kubernetes の配布を使用する場合は、「 --kubelet -dir 」を使用して代替パスを指定できます。

インストーラの引数で許可される範囲を超えてインストールをカスタマイズする必要がある場合は、配置ファイルをカスタマイズすることもできます。--generate-custom-yaml パラメータを使用して、インストーラの「 etup 」ディレクトリに次の YAML ファイルを作成します。

- trident-clusterrolebinding.yaml
- trident-deployment.yaml
- trident-CRDs .YAML
- trident-clusterrolment.yaml
- trident-demimonimon.yamml`
- trident-service.yaml
- trident-namespac.yaml
- trident-ServiceAccount.yaml
- trident-resourcesequota.yaml

これらのファイルを生成したら、必要に応じて変更し、「 --use-custom-yaml 」を使用してカスタム展開をインストールできます。

```
./tridentctl install -n trident --use-custom-yaml
```

Tridentを使用

ワーカーノードを準備します

Kubernetesクラスタ内のすべてのワーカーノードが、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、ドライバの選択に基づいて、NFS、iSCSI、NVMe/TCP、またはFCの各ツールをインストールする必要があります。

適切なツールを選択する

ドライバを組み合わせて使用している場合は、ドライバ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。最新バージョンのRedHat CoreOSには、デフォルトでツールがインストールされています。

NFSツール

"NFSツールのインストール" を使用している場合： `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `gcp-cvs`。

iSCSIツール

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

NVMeツール

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

NVMe/TCPにはONTAP 9.12以降を推奨します。

SCSI over FCツール

- SCSI over Fibre Channel（FC）は、Trident 24.10リリースの技術プレビュー機能です。*

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

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

ノードサービスの検出

Tridentは、ノードでiSCSIサービスまたは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検出されたサービスが特定されますが、サービスが適切に設定されていることは保証されませ逆に、検出されたサービスがない場合も、ボリュームのマウントが失敗する保証はありません。

イベントを確認します

Tridentは、検出されたサービスを識別するためのイベントをノードに対して作成します。次のイベントを確認するには、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

Tridentは、TridentノードCR上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、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用のコマンドを使用して、NFSツールをインストールします。ブート時にNFSサービスが開始されてい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
```

NFSツールをインストールしたあとにワーカーノードをリブートして、コンテナにボリュームを接続する際の障害を回避します。

iSCSI ボリューム

Tridentでは、iSCSIセッションの確立、LUNのスキャン、マルチパスデバイスの検出、フォーマット、ポッドへのマウントを自動的に実行できます。

iSCSIの自己回復機能

ONTAPシステムの場合、Tridentは5分ごとにiSCSIの自己修復を実行し、次のことを実現します。

1. *希望するiSCSIセッションの状態と現在のiSCSIセッションの状態を識別します
2. *希望する状態と現在の状態を比較して、必要な修理を特定します。Tridentは、修理の優先順位と、修理をいつプリエンプトするかを決定します。
3. *現在のiSCSIセッションの状態を希望するiSCSIセッションの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、それぞれのデーモンセットポッドのコンテナにあり `trident-main` ます。ログを表示するには、Tridentのインストール時にを「true」に設定しておく必要があります `debug`。

Trident iSCSIの自己修復機能を使用すると、次のことを防止できます。

- ネットワーク接続問題 後に発生する可能性がある古いiSCSIセッションまたは正常でないiSCSIセッション。セッションが古くなった場合、Tridentは7分間待機してからログアウトし、ポータルとの接続を再確立します。

  たとえば、ストレージコントローラでCHAPシークレットがローテーションされた場合にネットワークが接続を失うと、古い（*stale*）CHAPシークレットが保持されることがあります。自己修復では、これを認識し、自動的にセッションを再確立して、更新されたCHAPシークレットを適用できます。

- iSCSIセッションがありません
- LUNが見つかりません
- Tridentをアップグレードする前に考慮すべきポイント*
- ノード単位のigroup（23.04以降で導入）のみを使用している場合、iSCSIの自己修復によってSCSIバス内のすべてのデバイスに対してSCSI再スキャンが開始されます。
- バックエンドを対象としたigroup（23.04で廃止）のみを使用している場合、iSCSIの自己修復によってSCSIバス内の正確なLUN IDのSCSI再スキャンが開始されます。
- ノード単位のigroupとバックエンドを対象としたigroupが混在している場合、iSCSIの自己修復によってSCSIバス内の正確なLUN IDのSCSI再スキャンが開始されます。

iSCSIツールをインストール

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

作業を開始する前に

- Kubernetes クラスタ内の各ノードには一意の IQN を割り当てる必要があります。* 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RHEL互換のその他のLinuxディストリビューションをで使用している場合は、を使用します `solidfire-san` DriverおよびElement OS 12.5以前。CHAP認証アルゴリズムがMD5 inに設定されていることを確認します `/etc/iscsi/iscsid.conf`。Element 12.7では、FIPS準拠のセキュアなCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256が提供されています。

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

- iSCSI PVSを搭載したRHEL / RedHat CoreOSを実行するワーカーノードを使用する場合は、を指定します `discard` StorageClassのmountOptionを使用して、インラインのスペース再生を実行します。を参照してください "Red Hat のドキュメント"。

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 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. マルチパスを有効化：

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

「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

4. 「 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
```

5. 'iSCSI' を有効にして開始します

```
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 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：

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

4. マルチパスを有効化：

```
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「 defVaults 」に含まれていることを確認します。

5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されてい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 では 'iSCSI デーモンを起動するために 'open-iscsi' を起動する前に 'iscsiadm を持つターゲット・ポートを検出する必要がありますまたは 'iscsid' サービスを 'iscsid' を自動的に開始するように変更することもできます

iSCSI自己回復の設定または無効化

次の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の自己修復間隔*：iSCSIの自己修復を実行する頻度を指定します（デフォルト：5分）。小さい数値を設定することで実行頻度を高めるか、大きい数値を設定することで実行頻度を下げることができます。

iSCSIの自己修復間隔を0に設定すると、iSCSIの自己修復が完全に停止します。iSCSIの自己修復を無効にすることは推奨しません。iSCSIの自己修復が意図したとおりに機能しない、またはデバッグ目的で機能しない特定のシナリオでのみ無効にする必要があります。

- * iSCSI自己回復待機時間*：正常でないセッションからログアウトして再ログインを試みるまでのiSCSI自己回復の待機時間を決定します（デフォルト：7分）。健全でないと識別されたセッションがログアウトされてから再度ログインしようとするまでの待機時間を長くするか、またはログアウトしてログインしてからログインするまでの時間を短くするように設定できます。

Helm

iSCSIの自己修復設定を構成または変更するには、 `iscsiSelfHealingInterval` および `iscsiSelfHealingWaitTime` helmのインストール中またはhelmの更新中のパラメータ。

次の例では、iSCSIの自己修復間隔を3分、自己修復の待機時間を6分に設定しています。

```
helm install trident trident-operator-100.2410.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

Tridentctl

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

次の例では、iSCSIの自己修復間隔を3分、自己修復の待機時間を6分に設定しています。

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

NVMe/TCPホリユウム

オペレーティングシステムに対応した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や、使用しているカーネルバージョンに対応するNVMeパッケージがない場合は、ノードのカーネルバージョンをNVMeパッケージで更新しなければならないことがあります。

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
```

インストールを確認します

インストールが完了したら、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が割り当てられていることを確認します。

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

Tridentでは、NVMeがダウンしてもパスがあきらめないように値が変更され `ctrl_device_tmo` ます。この設定は変更しないでください。

ファイバチャネル（FC）のサポート

Fibre Channel（FC；ファイバチャネル）プロトコルをTridentで使用して、ONTAPシステムで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。

- SCSI over Fibre Channel（FC）は、Trident 24.10リリースの技術プレビュー機能です。*

ファイバチャネルは、その高いパフォーマンス、信頼性、拡張性から、エンタープライズストレージ環境で広く採用されているプロトコルです。ストレージデバイスに堅牢で効率的な通信チャネルを提供し、高速で安全なデータ転送を可能にします。SCSI over Fibre Channelを使用すると、既存のSCSIベースのストレージインフラストラクチャを活用しながら、ファイバチャネルの高パフォーマンスと長距離機能を活用できます。これにより、ストレージリソースを統合し、低レイテンシで大量のデータを処理できる、拡張性と効率性に優れたストレージエリアネットワーク（SAN）を構築できます。

TridentでFC機能を使用すると、次のことが可能になります。

- 配置仕様を使用してPVCを動的にプロビジョニングします。
- ボリュームのSnapshotを作成し、そのSnapshotから新しいボリュームを作成します。
- 既存のFC-PVCのクローンを作成します。
- 導入済みのボリュームのサイズを変更します。

前提条件

FCに必要なネットワークとノードを設定します。

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、を参照してください "network interface show" 。
2. イニシエータ（ホスト）のインターフェイスの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スイッチにゾーニングを設定します。

   詳細については、各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。

- "ファイバチャネルとFCoEのゾーニングの概要"
- "FCおよびFC-NVMe SANホストの構成方法"

ワーカーノードを準備します

Kubernetesクラスタ内のすべてのワーカーノードが、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FC用に準備するには、必要なツールをインストールする必要があります。

FCツールのインストール

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

- iSCSI PVSを搭載したRHEL / RedHat CoreOSを実行するワーカーノードを使用する場合は、を指定します `discard` StorageClassのmountOptionを使用して、インラインのスペース再生を実行します。を参照してください "Red Hat のドキュメント"。

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 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. マルチパスを有効化：

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

「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

4. 「 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
```

5. 'iSCSI' を有効にして開始します

```
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 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：

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

4. マルチパスを有効化：

```
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「 defVaults 」に含まれていることを確認します。

5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されてい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 では 'iSCSI デーモンを起動するために 'open-iscsi' を起動する前に 'iscsiadm を持つターゲット・ポートを検出する必要がありますまたは 'iscsid' サービスを 'iscsid' を自動的に開始するように変更することもできます

バックエンド構成の作成

ドライバおよび `fcp`sanType`としてTridentバックエンドを作成します `ontap-san`。

参照先：

- "バックエンドにONTAP SANドライバを設定する準備をします"
- "ONTAP のSAN構成オプションと例"

FCヲシヨウシタバックエンドコウセイノレイ

```
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
  sanType: fcp
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

ストレージクラスを作成する。

詳細については、以下を参照してください。

- "ストレージ構成オプション"

ストレージクラスの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fcp-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  protocol: "fcp"
  storagePool: "aggr1"
allowVolumeExpansion: True
```

バックエンドの構成と管理

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の間の関係を定義します。Tridentは、そのストレージシステムとの通信方法や、Tridentがそのシステムからボリュームをプロビジョニングする方法を解説します。

Tridentは、ストレージクラスで定義された要件に一致するストレージプールをバックエンドから自動的に提供します。ストレージシステムにバックエンドを設定する方法について説明します。

- "Azure NetApp Files バックエンドを設定します"
- "Cloud Volumes Service for Google Cloud Platform バックエンドを設定します"
- "NetApp HCI または SolidFire バックエンドを設定します"
- "バックエンドに ONTAP または Cloud Volumes ONTAP NAS ドライバを設定します"
- "バックエンドに ONTAP または Cloud Volumes ONTAP SAN ドライバを設定します"
- "Amazon FSx for NetApp ONTAPでTridentを使用"

Azure NetApp Files の特長

Azure NetApp Files バックエンドを設定します

Azure NetApp FilesをTridentのバックエンドとして設定できます。Azure NetApp Filesバックエンドを使用してNFSボリュームとSMBボリュームを接続できます。Tridentは、Azure Kubernetes Services（AKS）クラスタの管理対象IDを使用したクレデンシャル管理もサポートしています。

Azure NetApp Filesドライバの詳細

Tridentには、クラスタと通信するための次のAzure NetApp Filesストレージドライバが用意されています。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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 azure-NetApp-files 」と入力します	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	`nfs`、 `smb`

考慮事項

- Azure NetApp Filesサービスでは、50GiB未満のボリュームはサポートされません。より小さいボリュームを要求すると、Tridentは50GiBのボリュームを自動的に作成します。
- Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。

AKSの管理対象ID

Tridentでは、Azure Kubernetes Servicesクラスタがサポートされます"管理対象ID"。管理されたアイデンティティによって提供される合理的なクレデンシャル管理を利用するには、次のものが必要です。

- AKSを使用して導入されるKubernetesクラスタ
- AKS Kubernetesクラスタに設定された管理対象ID
- 指定する `"Azure"` を含む `Trident` がインストールされています。 `cloudProvider`

Trident オペレータ

Trident演算子を使用してTridentをインストールするには、を `tridentorchestrator_cr.yaml` に `"Azure"`設定します `cloudProvider`。例：

```
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、環境変数を使用してTridentセットをAzureに `$CP`インストールし `cloudProvider`ます。

```
helm install trident trident-operator-100.2410.0.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、フラグをに `Azure`設定し `cloudProvider`ます。

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

AKSのクラウドID

クラウドIDを使用すると、Kubernetesポッドは、明示的なAzureクレデンシャルを指定するのではなく、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ます。

Azureでクラウドを活用するには、以下が必要です。

- AKSを使用して導入されるKubernetesクラスタ
- AKS Kubernetesクラスタに設定されたワークロードIDとoidc-issuer
- ワークロードIDを指定 `"Azure"`および `cloudIdentity`指定するを含むTridentがインストールされている `cloudProvider`

Trident オペレータ

Trident演算子を使用してTridentをインストールするには、をに設定し、を `tridentorchestrator_cr.yaml`に `"Azure"`設定 `cloudProvider`し `cloudIdentity`azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`ます。

例：

```
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'*
```

Helm

次の環境変数を使用して、* cloud-provider（CP）フラグと 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-
xxxxxxxxxx'"
```

次の例では、環境変数を使用してTridentをインストールし `cloudProvider`、をAzureに `$CP`設定し、をUSING THE環境変数 `$CI`に設定し `cloudIdentity`ます。

```
helm install trident trident-operator-100.2410.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$CI"
```

<code>tridentctl</code>

次の環境変数を使用して、* cloud provider フラグと cloud identity *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-
xxxxxxxxxx"
```

次の例では、Tridentをインストールし、フラグをに設定し、 `cloud-identity`を `$CI`に `$CP`設定し `cloud-provider`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準備をします

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

NFSボリュームと**SMB**ボリュームの前提条件

Azure NetApp Files を初めて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にいくつかの初期設定が必要です。を参照してください "Azure：Azure NetApp Files をセットアップし、NFSボリュームを作成します"。

を設定して使用します "Azure NetApp Files の特長" バックエンドには次のものが必要です。

- `subscriptionID`、 `tenantID`、 `clientID`、 `location`および `clientSecret` AKSクラスタで管理対象IDを使用する場合はオプションです。
- `tenantID`、 `clientID`および `clientSecret` は、AKSクラスタでクラウドIDを使用する場合はオプションです。

- 容量プール。を参照してください "Microsoft：Azure NetApp Files 用の容量プールを作成します"。
- Azure NetApp Files に委任されたサブネット。を参照してください "Microsoft：サブネットをAzure NetApp Files に委任します"。
- Azure NetApp Files が有効な Azure サブスクリプションのスクリプト ID 。
- `tenantID`、 `clientID`および `clientSecret` から "アプリケーション登録" Azure Active Directory で、 Azure NetApp Files サービスに対する十分な権限がある。アプリケーション登録では、次のいずれかを使用します。
  - オーナーまたは寄与者のロール "Azureで事前定義"。
  - "カスタム投稿者ロール"(`assignableScopes`（サブスクリプションレベル）。次の権限がTridentで必要な権限のみに制限されています。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ます"を参照してください。

カスタムコントリビュータロール

```
{
    "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がサポートされます `location` を1つ以上含むデータセンターを展開します "委任されたサブネット"。Trident 22.01の時点では `location` パラメータは、バックエンド構成ファイルの最上位にある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。
- を使用してください `Cloud Identity`、 `client ID` Aから "ユーザーが割り当てた管理ID" そのIDを `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

SMBボリュームに関するその他の要件

SMBボリュームを作成するには、以下が必要です。

- Active Directoryが設定され、Azure NetApp Files に接続されています。を参照してください "Microsoft：Azure NetApp Files のActive Directory接続を作成および管理します"。
- LinuxコントローラノードとWindowsワーカーノードでWindows Server 2022を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Azure NetApp FilesがActive Directoryに対して認証できるように、Active Directory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サービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub: CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

Azure NetApp Files バックエンド構成のオプションと例

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説明し、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、場所）を使用して、要求された場所で使用可能な容量プール上に、要求されたサービスレベルとサブネットに一致するAzure NetApp Filesボリュームを作成します。

Tridentでは、手動QoS容量プールはサポートされません。

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	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「 azure-NetApp-files 」
backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + ランダムな文字
' スクリプト ID' 。	Azure サブスクリプションのサブスクリプション ID AKSクラスタで管理IDが有効になっている場合はオプションです。	
「 tenantID 」。	アプリケーション登録からのテナント ID AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	
「 clientID 」。	アプリケーション登録からのクライアント ID AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	
「 clientSecret 」を入力します。	アプリケーション登録からのクライアントシークレット AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	

パラメータ	説明	デフォルト
「サービスレベル」	「標準」、「プレミアム」、「ウルトラ」のいずれかです	"" （ランダム）
「ロケーション」	新しいボリュームを作成する Azure の場所の名前 AKSクラスタで管理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 」のように入力します	検出されたリソースをフィルタリングするためのネットアップアカウントのリスト	"[]" （フィルタなし）
「 capacityPools	検出されたリソースをフィルタリングする容量プールのリスト	"[]" （フィルタなし、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	""
「サブネット」	「 icrosoft.Netapp/volumes` 」に委任されたサブネットの名前	""
「ネットワーク機能」	ボリューム用のVNet機能のセットです。の場合もあります `Basic` または `Standard`。ネットワーク機能は一部の地域では使用できず、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。を指定します `networkFeatures` この機能を有効にしないと、ボリュームのプロビジョニングが失敗します。	""
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。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、を参照してください `nfsvers=4` カンマで区切って複数のマウントオプションリストを指定し、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よりも優先されます。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）

パラメータ	説明	デフォルト
「バグトレースフラグ」	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," メソッド ":true," 検出 ":true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`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、を参照してください "CSI トポロジを使用します"。	

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

必要な権限とリソース

PVCの作成時に「No capacity pools found」エラーが表示される場合は、アプリケーション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。デバッグを有効にすると、バックエンドの作成時に検出されたAzureリソースがTridentによってログに記録されます。適切なロールが使用されていることを確認します。

の値 `resourceGroups`、 `netappAccounts`、 `capacityPools`、 `virtualNetwork`および `subnet` 短縮名または完全修飾名を使用して指定できます。ほとんどの場合、短縮名は同じ名前の複数のリソースに一致する可能性があるため、完全修飾名を使用することを推奨します。

。 `resourceGroups`、 `netappAccounts`および `capacityPools` 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リソース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。完全修飾名の形式は次のとおりです。

を入力します	の形式で入力し
リソースグループ	< リソースグループ >
ネットアップアカウント	< リソースグループ >/< ネットアップアカウント >
容量プール	< リソースグループ >/< ネットアップアカウント >/< 容量プール >
仮想ネットワーク	< リソースグループ >/< 仮想ネットワーク >
サブネット	<resource group>/< 仮想ネットワーク >/< サブネット >

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で、デフォルトのボリュームプロビジョニングを制御できます。を参照してください [構成例] を参照してください。

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームに対するエクスポートルール `exportRule` CIDR表記のIPv4アドレスまたはIPv4サブネットの任意の組み合わせをカンマで区切って指定する必要があります。SMBボリュームでは無視されます。	"0.0.0.0/0 "
「スナップショット方向」	.snapshot ディレクトリの表示を制御します	NFSv4の場合は「true」NFSv3の場合は「false」
「 size 」	新しいボリュームのデフォルトサイズ	" 100G "
「 unixPermissions 」	新しいボリュームのUNIX権限（8進数の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的な設定を示しています。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、バックエンドの絶対的な最小構成です。この構成では、Tridentは設定された場所でAzure NetApp Filesに委譲されたすべてのNetAppアカウント、容量プール、およびサブネットを検出し、それらのプールおよびサブネットの1つに新しいボリュームをランダムに配置します。は省略されているため、 `nasType nfs` デフォルトが適用され、バックエンドでNFSボリュームがプロビジョニングされます。

この構成は、Azure NetApp Filesの使用を開始して試している段階で、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範囲を設定することが必要な場合に適しています。

```
---
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の管理対象**ID**

このバックエンド構成では、 `subscriptionID`、 `tenantID`、 `clientID`および `clientSecret`は、管理対象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
```

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`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構成

このバックエンド構成では、容量プール内のAzureの場所 `Ultra`にボリュームが配置され `eastus`ます。Tridentは、その場所のAzure NetApp Filesに委譲されたすべてのサブネットを自動的に検出し、そのいずれかに新しいボリュームをランダムに配置します。

```
---
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
```

高度な設定

このバックエンド構成は、ボリュームの配置を単一のサブネットにまで適用する手間をさらに削減し、一部のボリュームプロビジョニングのデフォルト設定も変更します。

```
---
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
virtualNetwork: my-virtual-network
subnet: 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'
```

仮想プール構成

このバックエンド構成では、 1 つのファイルに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。プールを区別するために、仮想プールのラベルを使用しました `performance`。

```
---
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:
  - ultra-1
  - ultra-2
  networkFeatures: Standard
- labels:
    performance: silver
  serviceLevel: Premium
  capacityPools:
  - premium-1
- labels:
    performance: bronze
  serviceLevel: Standard
  capacityPools:
  - standard-1
  - standard-2
```

サポートされるトポロジ構成

Tridentを使用すると、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用のボリュームを簡単にプロビジョニングできます。 `supportedTopologies`このバックエンド構成のブロックは、バックエンドごとに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クラスタノードのラベルのリージョンとゾーンの値と一致している必要があります。これらのリージョンとゾーンは、ストレージクラスで指定できる許容値のリストです。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、を参照してください "CSI トポロジを使用します"。

```
---
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
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

次のようになります `StorageClass` 定義は、上記のストレージプールを参照してください。

を使用した定義の例 `parameter.selector` フィールド

を使用します `parameter.selector` を指定できます `StorageClass` ボリュームをホストするために使用される仮想プール。ボリュームには、選択したプールで定義された要素があります。

```
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

を使用します `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クレデンシャルを指定できます。

デフォルトネームスペースの基本設定

```
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"
```

ネームスペース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でフィルタリングします。 `nasType: nfs` または `nasType: null` NFSプールに対してフィルタを適用します。

バックエンドを作成します

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>
```

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。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
```

構成ファイルで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ボリューム

Google Cloud NetApp Volumeバックエンドの設定

Google Cloud NetApp VolumesをTridentのバックエンドとして設定できるようになりました。Google Cloud NetApp Volumeバックエンドを使用してNFSボリュームを接続できます。

Google Cloud NetApp Volumesドライバの詳細

Tridentは、クラスタと通信するためのドライバを提供します `google-cloud-netapp-volumes`。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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
`google-cloud-netapp-volumes`	NFS	ファイルシステム	RWO、ROX、RWX、RWOP	`nfs`

GKEのクラウドID

クラウドIDを使用すると、Kubernetesポッドは、明示的なGoogle Cloudクレデンシャルを指定するのではなく、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ます。

Google Cloudでクラウドアイデンティティを活用するには、以下が必要です。

- GKEを使用して導入されるKubernetesクラスタ。
- GKEクラスタに設定されたワークロードIDおよびoidc-issuer。
- ワークロードIDを指定 `"GCP"`および `cloudIdentity`指定するを含むTridentがインストールされ

ています `cloudProvider。

Trident オペレータ

Trident演算子を使用してTridentをインストールするには、をに設定し、を `tridentorchestrator_cr.yaml`に `"GCP"`設定 `cloudProvider`し `cloudIdentity` `iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com`ます。

例：

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

Helm

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

```
export CP="GCP"
export ANNOTATION="iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com"
```

次の例では、環境変数を使用してTridentをインストールし、をGCPに設定し `cloudProvider`、を環境変数を使用 `$ANNOTATION`して `$CP`を設定し `cloudIdentity`ます。

```
helm install trident trident-operator-100.2406.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$ANNOTATION"
```

<code>tridentctl</code>

次の環境変数を使用して、* cloud provider フラグと cloud identity *フラグの値を設定します。

```
export CP="GCP"
export ANNOTATION="iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com"
```

次の例では、Tridentをインストールし、フラグをに設定し、 `cloud-identity`を `$ANNOTATION`に `$CP`設定し `cloud-provider`ます。

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

Google Cloud NetApp Volumeバックエンドを設定する準備

Google Cloud NetApp Volume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ボリュームノゼンテイジョウケン

Google Cloud NetApp Volumeを初めてまたは新しい場所で使用している場合は、Google Cloud NetApp VolumeをセットアップしてNFSボリュームを作成するために、いくつかの初期設定が必要です。を参照してください "作業を開始する前に"。

Google Cloud NetApp Volumeバックエンドを設定する前に、次の条件を満たしていることを確認してください。

- Google Cloud NetApp Volumes Serviceで設定されたGoogle Cloudアカウント。を参照してください "Google Cloud NetAppボリューム"。
- Google Cloudアカウントのプロジェクト番号。を参照してください "プロジェクトの特定"。
- NetApp Volume Admin）ロールが割り当てられたGoogle Cloudサービスアカウント (`netappcloudvolumes.admin`。を参照してください "IDおよびアクセス管理のロールと権限"。
- GCNVアカウントのAPIキーファイル。を参照して "サービスアカウントキーを作成します"
- ストレージプール。を参照してください "ストレージプールの概要"。

Google Cloud NetApp Volumeへのアクセスの設定方法の詳細については、を参照してください "Google Cloud NetApp Volumeへのアクセスをセットアップする"。

Google Cloud NetApp Volumeのバックエンド構成オプションと例

Google Cloud NetApp VolumeのNFSバックエンド構成オプションについて説明し、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、 1 つの 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場合は、バックエンドを追加で定義します。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	の値は `storageDriverName` 「google-cloud-netapp-volumes」と指定する必要があります。

パラメータ	説明	デフォルト
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。この値は、Google Cloudポータルのホームページにあります。	
「ロケーション」	TridentがGCNVボリュームを作成するGoogle Cloudの場所。リージョン間Kubernetesクラスタを作成する場合、で作成したボリュームは `location` 、複数のGoogle Cloudリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストを発生させます。	
「 apiKey 」と入力します	ロールが割り当てられたGoogle Cloudサービスアカウントの`APIキー` `netappcloudvolumes.admin` 。このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。には `apiKey` 、、、、、の各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`。	
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）
「サービスレベル」	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。	
「ネットワーク」	GCNVボリュームに使用されるGoogle Cloudネットワーク。	
「バグトレースフラグ」	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詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
`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、を参照してください "CSI トポロジを使用します"。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

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームのエクスポートルール。IPv4アドレスの任意の組み合わせをカンマで区切って指定する必要があります。	"0.0.0.0/0 "
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスします	NFSv4の場合は「true」NFSv3の場合は「false」
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	""（デフォルトの0を使用）
「 unixPermissions 」	新しいボリュームのUNIX権限（8進数の4桁）。	""

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的な設定を示しています。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、バックエンドの絶対的な最小構成です。この構成では、Tridentは設定された場所でGoogle Cloud NetApp Volumeに委譲されたすべてのストレージプールを検出し、それらのプールの1つに新しいボリュームをランダムに配置します。は省略されているため、 `nasType` `nfs` デフォルトが適用され、バックエンドでNFSボリュームがプロビジョニングされます。

この構成は、Google Cloud NetApp Volumeの使用を開始して試用す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範囲設定が必要になることがよくあります。

```
---

apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    XsYg6gyxy4zq7OlwWgLwGa==\n
```

```
    -----END PRIVATE KEY-----\n

---

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
```

StoragePoolsフイルタヲシヨウシタセツテイ

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----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
```

仮想プール構成

このバックエンド構成では、1つのファイルに複数の仮想プールが定義されます。仮想プールは、セクションで定義し storage ます。さまざまな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、それらを表すストレージクラスをKubernetesで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、次の例では performance 、仮想プールを区別するためにラベルと serviceLevel タイプが使用されています。

また、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たり、個 々 の仮想プールのデフォルト値を上書きしたりすることもできます。次の例では、 snapshotReserve exportRule 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、を参照してください "仮想プール"。

```
---

apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 'f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec'
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
```

```
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----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
```

サポートされるトポロジ構成

Tridentを使用すると、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用のボリュームを簡単にプロビジョニングできます。 `supportedTopologies`このバックエンド構成のブロックは、バックエンドごとに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クラスタノードのラベルのリージョンとゾーンの値と一致している必要があります。これらのリージョンとゾーンは、ストレージクラスで指定できる許容値のリストです。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、を参照してください "CSI トポロジを使用します"。

```
---
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、バックエンドの設定に何か問題があります。バックエンドについては、コマンドを使用して説明するか、次のコマンドを実行してログを表示して原因を特定できます `kubectl get tridentbackendconfig <backend-name>` 。

```
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。

その他の例

ストレージクラスの定義の例

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

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

フィールドを使用した定義例 **`parameter.selector`** ：

を使用する `parameter.selector` と、ボリュームのホストに使用される各に対してを指定できます `StorageClass` "仮想プール" 。ボリュームには、選択したプールで定義された要素があります。

```
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、を参照してください "ストレージクラスを作成する。"。

PVC定義の例**PVC**テイギノレイ

```
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バックエンド用にCloud Volumes Service を設定します

提供されている構成例を使用して、Tridentインストールのバックエンドとして NetApp Cloud Volumes Service for Google Cloudを構成する方法を説明します。

Google Cloudドライバの詳細

Tridentは、クラスタと通信するためのドライバを提供します `gcp-cvs`。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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 gcp-cvs 」	NFS	ファイルシステム	RWO、ROX、RWX、RWOP	`nfs`

TridentによるCloud Volumes Service for Google Cloudのサポートの詳細

Tridentでは"サービスタイプ"、次の2つのいずれかにCloud Volumes Serviceボリュームを作成できます。

- * CVS-Performance *：デフォルトのTridentサービスタイプ。パフォーマンスが最適化されたこのサービスタイプは、パフォーマンスを重視する本番環境のワークロードに最適です。CVS -パフォーマンスサービスタイプは、サイズが100GiB以上のボリュームをサポートするハードウェアオプションです。"3つのサービスレベル"次のいずれかを選択できます。
  - `standard`
  - `premium`
  - `extreme`
- * CVS *：CVSサービスタイプは、中程度のパフォーマンスレベルに制限された高レベルの可用性を提供します。CVSサービスタイプは、ストレージプールを使用して1GiB未満のボリュームをサポートするソフトウェアオプションです。ストレージプールには最大50個のボリュームを含めることができ、すべてのボリュームでプールの容量とパフォーマンスを共有できます。のいずれかを選択できます "2つのサービスレベル"：
  - `standardsw`
  - `zoneredundantstandardsw`

必要なもの

を設定して使用します "Cloud Volumes Service for Google Cloud" バックエンドには次のものが必要です。

- NetApp Cloud Volumes Service で設定されたGoogle Cloudアカウント
- Google Cloud アカウントのプロジェクト番号
- 「 netappcloudvolumes .admin 」ロールを持つ Google Cloud サービスアカウント
- Cloud Volumes Service アカウントのAPIキーファイル

バックエンド構成オプション

各バックエンドは、 1 つの 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場合は、バックエンドを追加で定義します。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	"GCP-cvs"
backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + API キーの一部
'torageClass'	CVSサービスタイプを指定するためのオプションのパラメータ。CVSサービスタイプを選択するために使用し `software`ます。それ以外の場合、Tridentはサービスタイプがcvs-Performanceとみなされ(`hardware`ます）。	
`storagePools`	CVSサービスタイプのみ。ボリューム作成用のストレージプールを指定す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。この値は、Google Cloudポータルのホームページにあります。	
「 hostProjectNumber 」	共有VPCネットワークを使用する場合は必須です。このシナリオでは、 `projectNumber` は、サービスプロジェクトです `hostProjectNumber` は、ホストプロジェクトです。	
「 apiRegion 」と入力します	TridentがCloud Volumes Serviceボリュームを作成するGoogle Cloudリージョン。リージョン間Kubernetesクラスタを作成する場合、で作成したボリュームは `apiRegion`、複数のGoogle Cloudリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストを発生させます。	
「 apiKey 」と入力します	を使用したGoogle CloudサービスアカウントのAPIキー `netappcloudvolumes.admin` ロール。このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。	

パラメータ	説明	デフォルト
「 ProxyURL 」と入力します	CVSアカウントへの接続にプロキシサーバが必要な場合は、プロキシURLを指定します。プロキシサーバには、 HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、プロキシサーバで自己署名証明書を使用するために証明書の検証はスキップされます。認証が有効になっているプロキシサーバはサポートされていません。	
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）
「サービスレベル」	新しいボリュームのCVS -パフォーマンスレベルまたはCVSサービスレベル。CVS -パフォーマンスの値はです `standard`、 `premium`または `extreme` 。CVSの値はです `standardsw` または `zoneredundantstandardsw`。	CVS -パフォーマンスのデフォルトは「Standard」です。CVSのデフォルトは"standardsw"です。
「ネットワーク」	Cloud Volumes Service ボリュームに使用するGoogle Cloudネットワーク。	デフォルト
「バグトレースフラグ」	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詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

`allowedTopologies`	クロスリージョンアクセスを有効にするには、のStorageClass定義を使用します `allowedTopologies` すべてのリージョンを含める必要があります。例：
	`- key: topology.kubernetes.io/region`
	`values:`
	`- us-east1`
	`- europe-west1`	

ボリュームのプロビジョニングオプション

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームのエクスポートルール。CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切って指定する必要があります。	"0.0.0.0/0 "
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスします	いいえ
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	"" （ CVS のデフォルト値をそのまま使用）

パラメータ	説明	デフォルト
「 size 」	新しいボリュームのサイズ。CVS - パフォーマンス最小値は100GiBです。CVS最小値は1GiBです。	CVS -パフォーマンスサービスのタイプはデフォルトで「100GiB」です。CVSサービスのタイプではデフォルトが設定されません が、1GiB以上が必要です。

CVS -パフォーマンスサービスの種類の例

次の例は、CVS -パフォーマンスサービスタイプの設定例を示しています。

例 **1** ：最小限の構成

これは、デフォルトの「標準」サービスレベルでデフォルトのCVSパフォーマンスサービスタイプを使用する最小バックエンド構成です。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```

例**2**：サービスレベルの設定

この例は、サービスレベルやボリュームのデフォルトなど、バックエンド構成オプションを示しています。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

例3：仮想プールの構成

この例では、を使用します `storage` 仮想プールおよびを設定します `StorageClasses` それはそれらを再度参照する。を参照してください [ストレージクラスの定義] をクリックして、ストレージクラスの定義方法を確認します。

ここでは、すべての仮想プールに対して特定のデフォルトが設定され、すべての仮想プールに対してが設定されます `snapshotReserve` 5%およびである `exportRule` を0.0.0.0/0に設定します。仮想プールは、で定義されます `storage` セクション。個 々 の仮想プールにはそれぞれ独自の定義があります `serviceLevel`をクリックすると、一部のプールでデフォルト値が上書きされます。プールを区別するために、仮想プールのラベルを使用しました `performance` および `protection`。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
    performance: extreme
    protection: extra
  serviceLevel: extreme
```

```
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
- labels:
    performance: extreme
    protection: standard
  serviceLevel: extreme
- labels:
    performance: premium
    protection: extra
  serviceLevel: premium
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
- labels:
    performance: premium
    protection: standard
  serviceLevel: premium
- labels:
    performance: standard
  serviceLevel: standard
```

ストレージクラスの定義

次のStorageClass定義は、仮想プールの構成例に適用されます。を使用します `parameters.selector`では、ボリュームのホストに使用する仮想プールをストレージクラスごとに指定できます。ボリュームには、選択したプールで定義された要素があります。

ストレージクラスの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-extra-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=extreme; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-standard-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium-extra-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=premium; protection=extra"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=premium; protection=standard"
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-standard
provisioner: csi.trident.netapp.io
parameters:
  selector: "performance=standard"
```

```
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extra-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=extra"
allowVolumeExpansion: true
```

- 最初のストレージクラス (`cvs-extreme-extra-protection`）を最初の仮想プールにマッピングします。スナップショット予約が 10% の非常に高いパフォーマンスを提供する唯一のプールです。
- 最後のStorageClass(`cvs-extra-protection`）は、10%のスナップショットリザーブを提供するストレージプールを呼び出します。Tridentは、選択する仮想プールを決定し、スナップショット予約の要件を確実に満たします。

CVSサービスタイプの例

次の例は、CVSサービスタイプの設定例を示しています。

例**1**：最小構成

これは、を使用するバックエンドの最小構成です `storageClass` CVSサービスタイプとデフォルトを指定するには `standardsw` サービスレベル：

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
serviceLevel: standardsw
```

例**2**：ストレージプールの構成

このバックエンド設定の例では、を使用して `storagePools` ストレージプールを設定します。

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
  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/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

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。

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

バックエンドの作成に失敗した場合は、バックエンドの設定に何か問題があります。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
```

構成ファイルで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。

NetApp HCI または SolidFire バックエンドを設定します

Trident環境でElementバックエンドを作成して使用する方法について説明します。

Elementドライバの詳細

Tridentは、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 `solidfire-san`。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_and_block_volume`モードをサポートしています。`volumeMode`の場合`Filesystem`、`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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 olidfire -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 ブロックデバイスです。
「 olidfire -san 」	iSCSI	ファイルシステム	RWO、RWOP	「 xfs 」、「 ext3 」、「 ext4 」

作業を開始する前に

Elementバックエンドを作成する前に、次の情報が必要になります。

- 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 クラスタ管理者またはボリュームを管理できるテナントユーザのクレデンシャル。
- すべての Kubernetes ワーカーノードに適切な iSCSI ツールをインストールする必要があります。を参照してください "ワーカーノードの準備情報"。

バックエンド構成オプション

バックエンド設定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	説明	デフォルト
「バージョン」		常に 1

パラメータ	説明	デフォルト
'torageDriverName'	ストレージドライバの名前	常に「 solidfire-san-」
backendName`	カスタム名またはストレージバックエンド	「 iSCSI_ 」 + ストレージ（ iSCSI ） IP アドレス SolidFire
「エンドポイント」	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP	
「 VIP 」	ストレージ（ iSCSI ）の IP アドレスとポート	
「ラベル」	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（見つからない場合に作成）	
「 InitiatorIFCace 」	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します	デフォルト
UseCHAP'	CHAPを使用してiSCSIを認証します。TridentはCHAPを使用します。	正しいです
「アクセスグループ」	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。
「タイプ」	QoS の仕様	
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します	“”（デフォルトでは適用されません）
「バグトレースフラグ」	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

トラブルシューティングを行い、詳細なログダンプが必要な場合を除き、「 ebugTraceFlags 」は使用しないでください。

例**1**：のバックエンド構成 `solidfire-san` **3**種類のボリュームを備えたドライバ

次の例は、 CHAP 認証を使用するバックエンドファイルと、特定の QoS 保証を適用した 3 つのボリュームタイプのモデリングを示しています。その場合 ' ストレージ・クラスを定義して 'iops`storage クラス・パラメータを使用します

```
---
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**：のバックエンドとストレージクラスの設定 `solidfire-san` 仮想プールを備えたドライバ

この例は、仮想プールとともに、それらを参照するStorageClassesとともに構成されているバックエンド定義ファイルを示しています。

ストレージプールに存在するラベルを、プロビジョニング時にバックエンドストレージLUNにコピーしますTrident。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

以下に示すバックエンド定義ファイルの例では、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。これにより、が設定されます `type` シルバー。仮想プールは、で定義されます `storage` セクション。この例では、一部のストレージプールが独自のタイプを設定し、一部のプールが上記の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ています。を使用する `parameters.selector` 各ストレージクラスは、ボリュームのホスト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

想プール内で定義された要素があります。

最初のStorageClass(`solidfire-gold-four`）が最初の仮想プールにマッピングされます。これは、ゴールドのパフォーマンスとゴールドのパフォーマンスを提供する唯一のプールです `Volume Type QoS`。最後のStorageClass(`solidfire-silver`）は、Silver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が選択する仮想プールを決定し、ストレージ要件が満たされるようにします。

```
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"
```

詳細については、こちらをご覧ください

- "ボリュームアクセスグループ"

ONTAP SAN ドライバ

ONTAP SANドライバの概要

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

ONTAP SANドライバの詳細

Tridentは、ONTAPクラスタと通信するための次の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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 ontap - san 」	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
「 ontap - san 」	iSCSI	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	「 xfs 」、「 ext3 」、「 ext4 」
「 ontap - san 」	NVMe/FC を参照してください NVMe/TCP に関するその他の考慮事項。	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
「 ontap - san 」	NVMe/FC を参照してください NVMe/TCP に関するその他の考慮事項。	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	「 xfs 」、「 ext3 」、「 ext4 」

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 ONTAP - SAN - エコノミー」	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
「 ONTAP - SAN - エコノミー」	iSCSI	ファイルシステム	RWO、RWOP ROXおよびRWXは、ファイルシステムボリュームモードでは使用できません。	「 xfs 」、「 ext3 」、「 ext4 」

- 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。
- 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

Tridentは、ONTAP管理者またはSVM管理者（通常はクラスタユーザ、 `vsadmin`SVMユーザ、または別の名前で同じロールのユーザを使用）として実行することを想定しています `admin`。Amazon FSx for NetApp ONTAP環境では、Tridentは、クラスタユーザまたは `vsadmin`SVMユーザを使用するONTAP管理者またはSVM管理者、または同じロールの別の名前のユーザとして実行される必要があります `fsxadmin`。この`fsxadmin`ユーザは、クラスタ管理者ユーザに代わる限定的なユーザです。

パラメータを使用する場合は `limitAggregateUsage`、クラスタ管理者の権限が必要です。TridentでAmazon FSx for NetApp ONTAPを使用している場合、`limitAggregateUsage`パラメータはユーザアカウントと `fsxadmin`ユーザアカウントでは機能しません `vsadmin`。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Tridentドライバが使用できる、より制限の厳しいロールを作成することは可能ですが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

NVMe/TCPに関するその他の考慮事項

Tridentは、次のドライバを使用してNon-Volatile Memory Express（NVMe）プロトコルをサポートします `ontap-san`。

- IPv6
- 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
- Kubernetesノードのグレースフルシャットダウンまたはグレースフルシャットダウン（24.06）

Tridentは以下をサポートしていません。

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

バックエンドに**ONTAP SAN**ドライバを設定する準備をします

ONTAP SANドライバでONTAPバックエンドを構成するための要件と認証オプションを理解します。

要件

すべてのONTAPバックエンドについて、TridentでSVMにアグリゲートを少なくとも1つ割り当てる必要があります。

複数のドライバを実行し、 1 つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば 'ONTAP-SAN' ドライバを使用する「 -dev 」クラスと 'ONTAP-SAN-エコノミー 'one を使用する「デフォルト」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iSCSIツールをインストールしておく必要があります。を参照してください "ワーカーノードを準備します" を参照してください。

ONTAPバックエンドの認証

Tridentには、ONTAPバックエンドの認証に2つのモードがあります。

- credential based ：必要な権限を持つ ONTAP ユーザのユーザ名とパスワード。ONTAP バージョンとの互換性を最大限に高めるために '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・ロールを使用することを推奨します
- 証明書ベース：Tridentは、バックエンドにインストールされている証明書を使用してONTAPクラスタと通信することもできます。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式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。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

> クレデンシャルと証明書の両方を*指定しようとすると、バックエンドの作成が失敗し、構成ファイルに複数の認証方法が指定されているというエラーが表示されます。

クレデンシャルベースの認証を有効にします

TridentがONTAPバックエンドと通信するには、SVMを対象としたクラスタを対象とした管理者に対するクレデンシャルが必要です。や `vsadmin` などの事前定義された標準のロールを使用することを推奨します

`admin。これにより、今後のONTAPリリースで使用する機能APIが公開される可能性がある将来のTridentリリースとの前方互換性が確保されます。Tridentでは、カスタムのセキュリティログインロールを作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

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注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成または更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証を有効にします

新規または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 ： Base64 でエンコードされたクライアント証明書の値。
- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された 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。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

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、 ONTAP ユーザとして認証するように Common Name （ CN ；共通名）を設定します。

```
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. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っ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. ONTAP クラスタにクライアント証明書とキーをインストールします（手順 1 ）。

```
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' 認証方式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ONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。

```
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. Base64 で証明書、キー、および信頼された CA 証明書をエンコードする。

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

7. 前の手順で得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して

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、クレデンシャルをローテーションしたりできます。これはどちらの方法でも機能します。ユーザ名と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が、証明書を使用するバックエンドはユーザ名とパスワードに基づいて更新できます。これを行うには、既存の認証方法を削除して、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 |
+------------+----------------+--------------------------------------
+--------+---------+
```

パスワードのローテーションを実行する際には、ストレージ管理者が最初に ONTAP でユーザのパスワードを更新する必要があります。この後にバックエンドアップデートが続きます。証明書のローテーションを実行する際に、複数の証明書をユーザに追加することができます。その後、バックエンドが更新されて新しい証明書が使用されるようになります。この証明書に続く古い証明書は、 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、その後のボリューム接続にも影響しません。バックエンドの更新が成功すると、TridentがONTAPバックエンドと通信し、以降のボリューム処理を処理できるようになります。

Trident用のカスタム**ONTAP**ロールの作成

Tridentで処理を実行するためにONTAP adminロールを使用する必要がないように、最小Privilegesを持つONTAPクラスタロールを作成できます。Tridentバックエンド構成にユーザ名を含めると、Trident作成したONTAPクラスタロールが使用されて処理が実行されます。

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

ONTAP CLIノシヨウ

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ユーザのユーザ名を作成します。

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

3. ユーザにロールをマッピングします。

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

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

1. カスタムロールの作成：
   a. クラスタレベルでカスタムロールを作成するには、*[クラスタ]>[設定]*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]>[Storage VM]>[設定]>[ユーザとロール]*を選択し `required SVM`ます。

   b. の横にある矢印アイコン（→*）を選択します。
   c. [Roles]*で[+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. ロールを**Trident**ユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。
   a. で**[**アイコンの追加**]**+*を選択します。
   b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。
   c. [保存（ Save ）] 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"
- "ロールとユーザを使用する"

双方向 **CHAP** を使用して接続を認証します

Tridentでは、ドライバと `ontap-san-economy`ドライバの双方向`CHAP`を使用して`iSCSI`セッションを認証できます `ontap-san`。これには、バックエンド定義でオプションを有効にする必要があり `useCHAP`ます。に設定する `true`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、

ユーザ名とシークレットをバックエンドファイルに設定します。接続の認証には双方向 CHAP を使用することを推奨します。次の設定例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
```

「 useCHAP 」パラメータは、 1 回だけ設定できるブール型のオプションです。デフォルトでは false に設定されています。true に設定したあとで、 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 」フィールドもバックエンド定義に含める必要があります。シークレットは 'tridentctl update を実行してバックエンドを作成した後に変更できます

動作の仕組み

trueに設定する `useCHAP`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を構成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 をセットアップします。
  - SVMのデフォルトのイニシエータセキュリティタイプがnone（デフォルトで設定）*で、*ボリュームに既存のLUNがない場合、Tridentはデフォルトのセキュリティタイプをに設定し `CHAP`、CHAPイニシエータとターゲットのユーザ名とシークレットの設定に進みます。
  - 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 イニシエータとターゲットのユーザ名とシークレットを設定します。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CRDを作成し `tridentbackend`、CHAPシークレットとユーザ名をKubernetesシークレットとして格納します。このバックエンドでTridentによって作成されたすべてのPVSがマウントされ、CHAP経由で接続されます。

クレデンシャルをローテーションし、バックエンドを更新

CHAP 証明書を更新するには 'backend.json ファイルの CHAP パラメータを更新しますこれには '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

バックエンドのCHAPシークレットを更新する場合は、を使用してバックエンドを更新する必要があります `tridentctl`。Tridentではこれらの変更を反映できないため、CLI / ONTAP UIでストレージクラスタのクレデンシャルを更新しないでください。

```
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ず、SVM上のTridentによってクレデンシャルが更新されてもアクティブなままです。新しい接続では更新されたクレデンシャルが使用され、既存の接続は引き続きアクティブになります。古い PVS を切断して再接続すると、更新されたクレデンシャルが使用されます。

ONTAP の**SAN**構成オプションと例

Tridentのインストール時に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、バックエンドの構成例と、バックエンドをStorageClassesにマッピングするための詳細を示します。

バックエンド構成オプション

バックエンド設定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	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	`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	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタ管理LIFまたはSVM管理LIFのIPアドレス。Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。IPv6フラグを使用してTridentがインストールされている場合は、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、のように角かっこで定義する必要があります `[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、を参照して[mcc-best]ください。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。* iSCSIの場合は指定しないでください。**Trident**は、を使用して**"ONTAPの選択的LUNマップ"**、マルチパスセッションの確立に必要な**iSCI LIF**を検出します。が明示的に定義されている場合は、警告が生成され `**dataLIF**`ます。MetroClusterの場合は省略してください。*を参照してください[mcc-best]。	SVMの派生物です
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* [mcc-best]。	SVM 「管理 LIF 」が指定されている場合に生成されます
「 useCHAP 」	CHAPを使用してONTAP SANドライバのiSCSIを認証します（ブーリアン）。バックエンドで指定されたSVMのデフォルト認証として双方向CHAPを設定して使用する場合は、Tridentのをに設定し `true`ます。詳細については、を参照してください "バックエンドにONTAP SANドライバを設定する準備をします" 。	「偽」
「 chapInitiatorSecret 」	CHAP イニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	""
「ラベル」	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。「 useCHAP = TRUE 」の場合は必須	""
「 chapUsername 」	インバウンドユーザ名。「 useCHAP = TRUE 」の場合は必須	""
「 chapTargetUsername 」	ターゲットユーザ名。「 useCHAP = TRUE 」の場合は必須	""

パラメータ	説明	デフォルト
「clientCertificate」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「clientPrivateKey」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「trustedCacertifate」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
「ユーザ名」	ONTAP クラスタとの通信に必要なユーザ名。クレデンシャルベースの認証に使用されます。	""
「password」と入力します	ONTAP クラスタとの通信にパスワードが必要です。クレデンシャルベースの認証に使用されます。	""
'VM'	使用する Storage Virtual Machine	SVM 「管理 LIF 」が指定されている場合に生成されます
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。あとから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`
「集約」	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）。ドライバの場合 `ontap-nas-flexgroup`、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 、使用可能ないずれかのアグリゲートを使用してFlexGroupボリュームをプロビジョニングできます。 SVMでアグリゲートが更新されると、Tridentコントローラを再起動せずにSVMをポーリングすることで、Tridentでアグリゲートが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ようにTridentで特定のアグリゲートを設定している場合、アグリゲートの名前を変更するかSVMから移動すると、SVMアグリゲートのポーリング中にTridentでバックエンドが障害状態になります。アグリゲートをSVMにあるアグリゲートに変更するか、アグリゲートを完全に削除してバックエンドをオンラインに戻す必要があります。	""

パラメータ	説明	デフォルト
「AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。Amazon FSx for NetApp ONTAP バックエンドを使用している場合は、を指定しないで `limitAggregateUsage`ください。指定されたと `vsadmin`には `fsxadmin`、アグリゲートの使用量を取得してTridentを使用して制限するために必要な権限が含まれていません。	""（デフォルトでは適用されません）
「limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、LUNで管理するボリュームの最大サイズも制限します。	""（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、200 です	`100`
「バグトレースフラグ」	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い、詳細なログダンプが必要な場合を除き、は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 `true`と、`Trident`はバックエンドとの通信に`ONTAP REST API`を使用します。に設定する `false`と、`Trident`はバックエンドとの通信に`ONTAP ZAPI`呼び出しを使用します。この機能には`ONTAP 9.11.1`以降が必要です。また、使用する`ONTAP`ログインロールには、アプリケーションへのアクセス権が必要です `ontap`。これは、事前に定義された役割と役割によって実現され `vsadmin` `cluster-admin` ます。Trident 24.06リリースおよびONTAP 9 .15.1以降では、 `userREST`がデフォルトでに設定されて `true`います。`ONTAP ZAPI`呼び出しを使用するには、をに `false`変更してください。 `useREST` `useREST` はNVMe/TCPに完全修飾されています。	`true` ONTAP 9.15.1以降の場合は、それ以外の場合は `false`。
`sanType`	iSCSI、 `nvme`NVMe/TCP、または `fcp`SCSI over Fibre Channel（FC；SCSI over Fibre Channel）に対してを選択します `iscsi`。「**FCP**」（**SCSI over FC**）は、**Trident 24.10**リリースの技術プレビュー機能です。	`iscsi` 空白の場合

パラメータ	説明	デフォルト
`formatOptions`	を使用して、 `formatOptions`コマンドのコマンドライン引数を指定します。この引数 `mkfs`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、好みに応じてボリュームをフォーマットできます。デバイスパスを除いて、mkfsコマンド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`ドライバでのみサポートされています。*	
`limitVolumePoolSize`	ONTAP SANエコノミーバックエンドでLUNを使用する場合の、要求可能な最大FlexVolサイズ。	"" （デフォルトでは適用されません）
`denyNewVolumePools`	バックエンドがLUNを格納するために新しいFlexVolボリュームを作成することを制限します `ontap-san-economy`。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では、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ています。

-E nodiscard：

- keep：mkfsの時点でブロックを破棄しないでください（ブロックの破棄は、最初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されたストレージで有効です）。これは廃止されたオプション「-K」に代わるもので、すべてのファイルシステム（xfs、ext3、およびext4）に適用できます。

ボリュームのプロビジョニング用のバックエンド構成オプション

これらのオプションを使用して、のデフォルトプロビジョニングを制御できます `defaults` 設定のセクション。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	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	"正しい"
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
「ナプショットポリシー」	使用する Snapshot ポリシー	"なし"

パラメータ	説明	デフォルト
「 QOSPolicy 」	作成し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ます。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コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。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ます	""
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 `snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。 `false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。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すべてのボリュームでが有効になります。詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	いいえ
`luksEncryption`	LUKS暗号化を有効にします。を参照してください"Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。 LUKS暗号化はNVMe/TCPではサポートされません。	""
'ecurityStyle'	新しいボリュームのセキュリティ形式	`unix`
階層ポリシー	「none」を使用する階層化ポリシー	ONTAP 9.5より前のSVM-DR設定の場合は「snapshot-only」
`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

デフォルトが定義されている例を次に示します。

```
---
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'
```

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、 `ontap-san`TridentはLUNメタデータに対応するために10%の容量をFlexVolに追加します。LUN は、ユーザが PVC で要求したサイズとまったく同じ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）。ユーザには、要求した使用可能容量が割り当てられます。また、利用可能なスペースがフルに活用されていないかぎり、 LUN が読み取り専用になることもありません。これは、 ONTAP と SAN の経済性には該当しません。

定義されたバックエンドの場合 `snapshotReserve`、Tridentは次のようにボリュームのサイズを計算します。

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

1.1は、LUNメタデータに対応するためにFlexVolに追加される10%のTridentです。= 5%、PVC要求= 5GiBの場合、 `snapshotReserve`ボリュームの合計サイズは5.79GiB、使用可能なサイズは5.5GiBです。 `volume show`次の例のような結果が表示されます。

```
Vserver   Volume       Aggregate    State        Type       Size  Available Used%
--------- ------------ ------------ ---------- ---- ---------- ---------- -----
          _pvc_89f1c156_3801_4de4_9f9d_034d54c395f4
                       online     RW         10GB     5.00GB    0%
          _pvc_e42ec6fe_3baa_4af6_996d_134adbbb8e6d
                       online     RW       5.79GB     5.50GB    0%
          _pvc_e8372153_9ad9_474a_951a_08ae15e1c0ba
                       online     RW          1GB    511.8MB    0%
3 entries were displayed.
```

現在、既存のボリュームに対して新しい計算を行うには、サイズ変更だけを使用します。

最小限の設定例

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的な設定を示しています。これは、バックエンドを定義する最も簡単な方法です。

TridentでAmazon FSx on NetApp ONTAPを使用している場合は、IPアドレスではなく、LIFのDNS名を指定することを推奨します。

ONTAP SANの例

これは、 `ontap-san` ドライバ。

```
---
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>
```

ONTAP SANの経済性の例

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

1. 例

スイッチオーバーやスイッチバックの実行中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。 "SVMのレプリケーションとリカバリ"。

シームレスなスイッチオーバーとスイッチバックを実現するには、 `managementLIF` を省略します。 `dataLIF` および `svm` パラメータ例：

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

証明書ベースの認証の例

この基本的な設定例では、 `clientCertificate`、 `clientPrivateKey`および `trustedCACertificate` （信頼されたCAを使用している場合はオプション）がに入力されます `backend.json` およびは、クライアント証明書、秘密鍵、信頼されたCA証明書のbase64エンコード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

次の例では、 `useCHAP` をに設定します `true`。

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エコノミー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バックエンドでNVMeを使用するSVMを設定しておく必要があります。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
```

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}}_{{.volume.R
equestName}}"
},
"labels": {"cluster": "ClusterA", "PVC":
"{{.volume.Namespace}}_{{.volume.RequestName}}"}
```

<code> ONTAP SANエコノミー</code>ドライバのformatOptions**の例**

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

仮想プールを使用するバックエンドの例

これらのサンプルバックエンド定義ファイルでは、次のような特定のデフォルトがすべてのストレージプールに設定されています。 `spaceReserve` 「なし」の場合は、 `spaceAllocation` との誤り `encryption` 実行されます。仮想プールは、ストレージセクションで定義します。

Tridentでは、[Comments]フィールドにプロビジョニングラベルが設定されます。FlexVol にコメントが設定されます。Tridentは、仮想プールに存在するすべてのラベルをプロビジョニング時にストレージボリュームにコピーします。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

これらの例では、一部のストレージプールが独自の `spaceReserve`、 `spaceAllocation`および `encryption` 値、および一部のプールはデフォルト値よりも優先されます。

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の経済性の例

```
---
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** にマッピングします

次のStorageClass定義は、 [仮想プールを使用するバックエンドの例]。を使用する `parameters.selector` フィールドでは、各StorageClassがボリュームのホスト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内で定義された要素があります。

- 。 `protection-gold` StorageClassは、 `ontap-san` バックエンド：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、内の2番目と3番目の仮想プールにマッピングされます。 `ontap-san` バックエンド：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は内の3番目の仮想プールにマッピングされます `ontap-san-economy` バックエンド：これは、mysqldbタイプアプリケーション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は内の2番目の仮想プールにマッピングされます `ontap-san` バックエンド：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は内の3番目の仮想プールにマッピングされます `ontap-san` バックエンドと内の4番目の仮想プール `ontap-san-economy` バックエンド：これらは、5000クレジットポイントを持つ唯一のプールオファリングです。

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

- 。 `my-test-app-sc` StorageClassはにマッピングされます `testAPP` 内の仮想プール `ontap-san` ドライバ `sanType: nvme`。これは唯一のプールサービスです。 `testApp`。

```
---
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が選択する仮想プールを決定し、ストレージ要件が満たされるようにします。

ONTAP NAS ドライバ

ONTAP NASドライバの概要

ONTAP および Cloud Volumes ONTAP の NAS ドライバを使用した ONTAP バックエンドの設定について説明します。

ONTAP NASドライバの詳細

Tridentは、ONTAPクラスタと通信するための次の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。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 ONTAP - NAS 」	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、 nfs、 smb
「 ONTAP - NAS - エコノミー」	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、 nfs、 smb
「 ONTAP-NAS-flexgroup 」	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、 nfs、 smb

- 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。
- 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

Tridentは、ONTAP管理者またはSVM管理者（通常はクラスタユーザ、 `vsadmin`SVMユーザ、または別の名前で同じロールのユーザを使用）として実行することを想定しています `admin`。

Amazon FSx for NetApp ONTAP環境では、Tridentは、クラスタユーザまたは `vsadmin`SVMユーザを使用するONTAP管理者またはSVM管理者、または同じロールの別の名前のユーザとして実行される必要があります `fsxadmin`。この `fsxadmin`ユーザは、クラスタ管理者ユーザに代わる限定的なユーザです。

パラメータを使用する場合は `limitAggregateUsage`、クラスタ管理者の権限が必要です。TridentでAmazon FSx for NetApp ONTAPを使用している場合、 `limitAggregateUsage`パラメータはユーザアカウントと `fsxadmin`ユーザアカウントでは機能しません `vsadmin`。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Tridentドライバが使用できる、より制限の厳しいロールを作成することは可能ですが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

ONTAP NAS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 NASドライバでONTAPバックエンドを設定するための要件、認証オプション、およびエクスポートポリシーを理解します。

要件

- すべてのONTAPバックエンドについて、TridentでSVMにアグリゲートを少なくとも1つ割り当てる必要があります。
- 複数のドライバを実行し、どちらか一方を参照するストレージクラスを作成できます。たとえば、を使用するGoldクラスを設定できます `ontap-nas` ドライバとを使用するBronzeクラス `ontap-nas-economy` 1つ。
- すべてのKubernetesワーカーノードに適切なNFSツールをインストールしておく必要があります。を参照してください "こちらをご覧ください" 詳細：
- Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。詳細については、を参照してください SMBボリュームをプロビジョニングする準備をします 。

ONTAPバックエンドの認証

Tridentには、ONTAPバックエンドの認証に2つのモードがあります。

- Credential-based：このモードでは、ONTAPバックエンドに十分な権限が必要です。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推奨します。例： `admin` または `vsadmin` ONTAP のバージョンとの互換性を最大限に高めるため。
- 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。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式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。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

> クレデンシャルと証明書の両方を*指定しようとすると、バックエンドの作成が失敗し、構成ファイルに複数の認証方法が指定されているというエラーが表示されます。

クレデンシャルベースの認証を有効にします

TridentがONTAPバックエンドと通信するには、SVMを対象としたクラスタを対象とした管理者に対するクレデンシャルが必要です。や `vsadmin`などの事前定義された標準のロールを使用することを推奨します `admin`。これにより、今後のONTAPリリースで使用する機能APIが公開される可能性がある将来のTridentリリースとの前方互換性が確保されます。Tridentでは、カスタムのセキュリティログインロールを作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

JSON

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "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注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成と更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証を有効にします

新規または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 ： Base64 でエンコードされたクライアント証明書の値。
- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された 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。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

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、 ONTAP ユーザとして認証するように Common

Name （ CN ；共通名）を設定します。

```
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. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っ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. ONTAP クラスタにクライアント証明書とキーをインストールします（手順 1 ）。

```
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' 認証方式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ONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。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. Base64 で証明書、キー、および信頼された CA 証明書をエンコードする。

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

7. 前の手順で得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して

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、クレデンシャルをローテーションしたりできます。これはどちらの方法でも機能します。ユーザ名と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が、証明書を使用するバックエンドはユーザ名とパスワードに基づいて更新できます。これを行うには、既存の認証方法を削除して、新しい認証方法を追加する必要があります。次に、更新されたbackend.jsonファイルに必要なパラメータが含まれたものを使用して実行します `tridentctl update backend`。

```
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 |
+------------+----------------+--------------------------------------
+--------+---------+
```

> パスワードのローテーションを実行する際には、ストレージ管理者が最初に ONTAP でユーザのパスワードを更新する必要があります。この後にバックエンドアップデートが続きます。証明書のローテーションを実行する際に、複数の証明書をユーザに追加することができます。その後、バックエンドが更新されて新しい証明書が使用されるようになります。この証明書に続く古い証明書は、 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、その後のボリューム接続にも影響しません。バックエンドの更新が成功すると、TridentがONTAPバックエンドと通信し、以降のボリューム処理を処理できるようになります。

Trident用のカスタム**ONTAP**ロールの作成

Tridentで処理を実行するためにONTAP adminロールを使用する必要がないように、最小Privilegesを持つONTAPクラスタロールを作成できます。Tridentバックエンド構成にユーザ名を含めると、Trident作成したONTAPクラスタロールが使用されて処理が実行されます。

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

ONTAP CLIノシヨウ

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ユーザのユーザ名を作成します。

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

3. ユーザにロールをマッピングします。

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

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

1. カスタムロールの作成：
   a. クラスタレベルでカスタムロールを作成するには、*[クラスタ]>[設定]*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]>[Storage VM]>[設定]>[ユーザとロール]*を選択し `required SVM`ます。

   b. の横にある矢印アイコン（→*）を選択します。
   c. [Roles]*で[+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. ロールを**Trident**ユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。
   a. で**[**アイコンの追加**]**+*を選択します。
   b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。
   c. [保存（ Save ）] 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"
- "ロールとユーザを使用する"

NFS エクスポートポリシーを管理します

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でエクスポートポリシーを使用する場合は、次の2つのオプションがあります。

- Tridentでは、エクスポートポリシー自体を動的に管理できます。この処理モードでは、許可可能なIPアドレスを表すCIDRブロックのリストをストレージ管理者が指定します。Tridentは、これらの範囲に該当する該当するノードIPを公開時に自動的にエクスポートポリシーに追加します。または、CIDRを指定しない場合は、パブリッシュ先のボリュームで見つかったグローバル対象のユニキャストIPがすべてエクスポートポリシーに追加されます。
- ストレージ管理者は、エクスポートポリシーを作成したり、ルールを手動で追加したりできます。Tridentでは、設定で別のエクスポートポリシー名を指定しないかぎり、デフォルトのエクスポートポリシーが使用されます。

エクスポートポリシーを動的に管理

Tridentでは、ONTAPバックエンドのエクスポートポリシーを動的に管理できます。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、ワーカーノードの IP で許容されるアドレススペースを指定できます。エクスポートポリシーの管理が大幅に簡易化され、エクスポートポリシーを変更しても、ストレージクラスタに対する手動の操作は不要になります。さらに、ボリュームをマウントしていて、指定された範囲のIPを持つワーカーノードだけにストレージクラスタへのアクセスを制限し、きめ細かく自動化された管理をサポートします。

ダイナミックエクスポートポリシーを使用する場合は、Network Address Translation（NAT;ネットワークアドレス変換）を使用しないでください。NATを使用すると、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のNATアドレスを認識するため、エクスポートルールに一致しない場合はアクセスが拒否されます。

Trident 24.10では、 `ontap-nas`ストレージドライバは以前のリリースと同様に動作します。ONTAP NASドライバに変更はありません。Trident 24.10では、ボリュームベースのきめ細かなアクセス制御が可能なのはストレージドライバだけ `ontap-nas-economy`です。

例

2 つの設定オプションを使用する必要があります。バックエンド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
```

この機能を使用する場合は、SVMのルートジャンクションに、ノードのCIDRブロックを許可するエクスポートルール（デフォルトのエクスポートポリシーなど）を含む事前に作成したエクスポートポリシーがあることを確認する必要があります。1つのSVMをTrident専用にするには、必ずNetAppのベストプラクティスに従ってください。

ここでは、上記の例を使用してこの機能がどのように動作するかについて説明します。

- `autoExportPolicy`がに設定されてい `true`ます。これは、Tridentが、このバックエンドを使用してSVMに対してプロビジョニングされたボリュームごとにエクスポートポリシーを作成し、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 `autoexportCIDRs`を示します `svm1`。ボリュームがノードに接続されるまでは、そのボリュームへの不要なアクセスを防止するルールのない空のエクスポートポリシーが使用されます。ボリュームがノードに公開されると、Tridentは、指定したCIDRブロック内のノードIPを含む基盤となる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で使用されるエクスポートポリシーにも追加されます。
  - 例：
    - 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
    - svm_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1cという名前のqtree Tridentでは、という名前のFlexVolのエクスポートポリシー、という名前のqtreeのエクスポートポリシー、`trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`およびという名前の空のエクスポートポリシー `trident_empty`がSVM上に作成されます `trident-403b5326-8482-40db96d0-d83fb3f4daec`。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、関連付けられていないボリュームで再利用されます。
- `autoExportCIDRs`アドレスブロックのリストが含まれます。このフィールドは省略可能で、デフォルト値は ["0.0.0.0/0" 、 "::/0" です。定義されていない場合、Tridentは、パブリケーションを使用して、ワーカーノード上で見つかったグローバルスコープのユニキャストアドレスをすべて追加します。

この例では `192.168.0.0/24`、アドレス空間が提供されています。これは、パブリケーションでこのアドレス範囲に含まれるKubernetesノードIPが、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は、実行するノードを登録すると、ノードのIPアドレスを取得し、で指定されたアドレスブロックと照合し `autoExportCIDRs`ます。公開時に、IPをフィルタリングした後、Tridentは公開先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。

バックエンドの作成後に 'autoExportPolicy' および 'autoExportCIDRs を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 CIDRs を追加したり、既存の CIDRs を削除したりできます。CIDRs を削除する際は、既存の接続が切断されないように注意してください。バックエンドに対して「 autoExportPolicy 」を無効にし、手動で作成したエクスポートポリシーに戻すこともできます。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後、または対応するCRDを `tridentbackend`使用してバックエンドをチェックでき `tridentctl`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
```

ノードを削除すると、Tridentはすべてのエクスポートポリシーをチェックして、そのノードに対応するアクセスルールを削除します。Tridentは、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、不正なマウントを防止します。ただし、このIPがクラスタ内の新しいノードで再利用される場合を除きます。

既存のバックエンドがある場合は、を使用してバックエンドを更新する `tridentctl update backend` と、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名に基づいて、必要に応じてという名前の新しいエクスポートポリシーが2つ作成されます。バックエンドにあるボリュームは、アンマウントして再度マウントしたあとに、新しく作成したエクスポートポリシーを使用します。

> 自動管理されたエクスポートポリシーを使用して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が削除されます。バックエンドが再作成されると、そのバックエンドは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。

稼働中のノードのIPアドレスが更新された場合は、そのノードでTridentポッドを再起動する必要があります。その後、Tridentは管理しているバックエンドのエクスポートポリシーを更新して、IPの変更を反映します。

SMBボリュームをプロビジョニングする準備をします

多少の準備が必要な場合は、次のツールを使用してSMBボリュームをプロビジョニングできます。 `ontap-nas` ドライバ。

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

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

作業を開始する前に

SMBボリュームをプロビジョニングする前に、以下を準備しておく必要があります。

- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2022を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Active Directory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サービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub: CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

手順

1. オンプレミスのONTAPでは、必要に応じてSMB共有を作成することも、Tridentで共有を作成することもできます。

   Amazon FSx for ONTAPにはSMB共有が必要です。

   SMB管理共有は、のいずれかの方法で作成できます "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を使用します。ONTAP CLIを使用してSMB共有を作成するには、次の手順を実行します

   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、共有の作成時に-pathオプションで指定されているパスを確認します。指定したパスが存在しない場合、コマンドは失敗します。

   b. 指定したSVMに関連付けられている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
```

   を参照してください "SMB 共有を作成" 詳細については、

2. バックエンドを作成する際に、SMBボリュームを指定するように次の項目を設定する必要があります。ONTAP バックエンド構成オプションのすべてのFSXについては、を参照してください "FSX（ONTAP の構成オプションと例）"。

パラメータ	説明	例
`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、TridentでSMB共有を作成できるようにする名前、ボリュームへの共通の共有アクセスを禁止する場合はパラメータを空白のままにします。オンプレミスのONTAPでは、このパラメータはオプションです。このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。	`smb-share`
`nasType`	をに設定する必要があります **`smb`**. nullの場合、デフォルトはです `nfs`。	`smb`
'ecurityStyle'	新しいボリュームのセキュリティ形式。をに設定する必要があります **`ntfs`** または **`mixed`** **SMB**ボリューム	`ntfs` または `mixed` SMBボリュームの場合
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておく必要があります。*	""

ONTAP NASの設定オプションと例

Tridentのインストール時にONTAP NAS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、バックエンドの構成例と、バックエンドをStorageClassesにマッピングするための詳細を示します。

バックエンド構成オプション

バックエンド設定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	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「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	ドライバ名+"_"+ dataLIF
「管理 LIF 」	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。IPv6フラグを使用してTridentがインストールされている場合は、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、のように角かっこで定義する必要があります `[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、を参照して[mcc-best]ください。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」

パラメータ	説明	デフォルト
「重複排除	プロトコル LIF の IP アドレス。指定することをお勧めします `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください 。IPv6フラグを使用してTridentがインストールされている場合は、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、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。* MetroClusterの場合は省略してください。*を参照してください[mcc-best]。	指定されたアドレス、または指定されていない場合はSVMから取得されるアドレス（非推奨）
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* [mcc-best]。	SVM 「管理 LIF 」が指定されている場合に生成されます
「 autoExportPolicy 」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Tridentでエクスポートポリシーを自動的に管理できます。	いいえ
「 autoExportCI` 」	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Tridentで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"]'
「ラベル」	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 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	""
「ユーザ名」	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
「 password 」と入力します	クラスタ / 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	

パラメータ	説明	デフォルト
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません qtree-nas-economyとstoragePrefixをONTAP 24文字以上で使用する場合、ボリューム名にはストレージプレフィックスは含まれませんが、qtreeにはストレージプレフィックスが埋め込まれます。	"トライデント"
「集約」	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）。ドライバの場合 `ontap-nas-flexgroup`、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 、使用可能ないずれかのアグリゲートを使用してFlexGroupボリュームをプロビジョニングできます。 SVMでアグリゲートが更新されると、Tridentコントローラを再起動せずにSVMをポーリングすることで、Tridentでアグリゲートが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ようにTridentで特定のアグリゲートを設定している場合、アグリゲートの名前を変更するかSVMから移動すると、SVMアグリゲートのポーリング中にTridentでバックエンドが障害状態になります。アグリゲートをSVMにあるアグリゲートに変更するか、アグリゲートを完全に削除してバックエンドをオンラインに戻す必要があります。	""
「AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	"" （デフォルトでは適用されません）

パラメータ	説明	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。設定されている場合は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を使用して、FlexGroupボリュームがプロビジョニングされます。ONTAP - NAS - FlexGroup *ストレージドライバーでサポートされています。 SVMでアグリゲートリストが更新されると、Tridentコントローラを再起動せずにSVMをポーリングすることで、Trident内のアグリゲートリストが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ようにTridentで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を変更するかSVMから移動すると、Tridentアグリゲートのポーリング中にバックエンドが障害状態になります。アグリゲートリストをSVM上のアグリゲートリストに変更するか、アグリゲートリストを完全に削除してバックエンドをオンラインに戻す必要があります。	""
「limitVolumeSize」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeに対して管理するボリュームの最大サイズを制限し、オプションを使用 `qtreesPerFlexvol`するとFlexVolあたりの最大qtree数をカスタマイズできます。	""（デフォルトでは適用されません）
「バグトレースフラグ」	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} 使用しないでください `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
「nfsvMountOptions」のように入力します	NFSマウントオプションをカンマで区切ったリスト。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、ストレージクラスに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に指定されているマウントオプションを使用してフォールバックします。ストレージクラスまたは構成ファイルでマウントオプションが指定されていない場合、Tridentは関連付けられた永続ボリュームにマウントオプションを設定しません。	""

パラメータ	説明	デフォルト
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	"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、TridentでSMB共有を作成できるようにする名前、ボリュームへの共通の共有アクセスを禁止する場合はパラメータを空白のままにします。オンプレミスのONTAPでは、このパラメータはオプションです。このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空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 `true` と、Tridentはバックエンドとの通信にONTAP REST APIを使用します。に設定する `false` と、Tridentはバックエンドとの通信にONTAP ZAPI呼び出しを使用します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap 。これは、事前に定義された役割と役割によって実現され vsadmin cluster-admin ます。Trident 24.06リリースおよびONTAP 9 .15.1以降では、 userREST`がデフォルトでに設定されて `true`います。ONTAP ZAPI呼び出しを使用するには、をに `false`変更してください。 `useREST	true ONTAP 9.15.1以降の場合は、それ以外の場合は false。
limitVolumePoolSize	ONTAP NASエコノミーバックエンドでqtreeを使用する場合の、要求可能なFlexVolの最大サイズ。	"" （デフォルトでは適用されません）
denyNewVolumePools	を制限し `ontap-nas-economy`バックエンドがqtreeを格納するために新しいFlexVolボリュームを作成することます。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。	

ボリュームのプロビジョニング用のバックエンド構成オプション

これらのオプションを使用して、のデフォルトプロビジョニングを制御できます defaults 設定のセクション。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	説明	デフォルト
「平和の配分」	qtreeに対するスペース割り当て	"正しい"
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
「ナプショットポリシー」	使用する Snapshot ポリシー	"なし"

パラメータ	説明	デフォルト
「 QOSPolicy 」	作成し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ます	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。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ます。経済性に影響する ONTAP - NAS ではサポートされません。	""
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 `snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。 `false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。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すべてのボリュームでは、NAEが有効になります。詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	いいえ
階層ポリシー	「none」を使用する階層化ポリシー	ONTAP 9.5より前のSVM-DR設定の場合は「snapshot-only」
「 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）
「スナップショット方向」	にアクセスする権限を管理します。 `.snapshot` ディレクトリ	NFSv4の場合は「true」NFSv3の場合は「false」
「 exportPolicy 」と入力します	使用するエクスポートポリシー	デフォルト
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。
`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。	""

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コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。

ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

```
---
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'
```

と `ontap-nas-flexgroups`については、 `ontap-nas`Trident新しい計算式を使用して、FlexVol がsnapshotReserveの割合とPVCで正しくサイジングされるようになりました。ユーザがPVCを要求すると、Tridentは新しい計算を使用して、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、ユーザは要求された PVC 内の書き込み可能なスペースを受信し、要求されたスペースよりも少ないスペースを確保できます。v21.07 より前のバージョンでは、ユーザが PVC を要求すると（ 5GiB など）、 snapshotReserve が 50% に設定されている場合、書き込み可能なスペースは 2.5GiB のみになります。これは、ユーザが要求したのはボリューム全体であり、その割合であるため `snapshotReserve`です。Trident 21.07では、ユーザが要求するのは書き込み可能なスペースであり、Tridentではボリューム全体に対する割合として定義されます。 `snapshotReserve`これはには適用されませ `ontap-nas-economy`ん。この機能の仕組みについては、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 = 5GiB の場合、ボリュームの合計サイズは 2/0.5 = 10GiB であり、使用可能なサイズは 5GiB であり、これが PVC 要求で要求されたサイズです。volume show コマンドは ' 次の例のような結果を表示する必要があります

以前のインストールからの既存のバックエンドでは、Tridentのアップグレード時に前述のようにボリュームがプロビジョニングされます。アップグレード前に作成したボリュームについては、変更が反映されるよう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、以前のと2GiBのPVCで `snapshotReserve=50` は、1GiB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 3GiB に変更すると、アプリケーションの書き込み可能なスペースが 6GiB のボリュームで 3GiB になります。

最小限の設定例

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的な設定を示しています。これは、バックエンドを定義する最も簡単な方法です。

ネットアップ ONTAP で Trident を使用している場合は、 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

スイッチオーバーやスイッチバックの実行中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。 "SVMのレプリケーションとリカバリ"。

シームレスなスイッチオーバーとスイッチバックを実現するには、 `managementLIF` を省略します。 `dataLIF` および `svm` パラメータ例：

```
---
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、バックエンドの最小限の設定例です。 `clientCertificate`、 `clientPrivateKey`および `trustedCACertificate` （信頼されたCAを使用している場合はオプション）がに入力されます `backend.json` およびは、クライアント証明書、秘密鍵、信頼されたCA証明書のbase64エンコード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

この例は、動的な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Tridentに指示する方法を示しています。これは、ドライバと `ontap-nas-flexgroup`ドライバで同じように機能し `ontap-nas-economy`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、を示しています `managementLIF` IPv6アドレスを使用している。

```
---
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
```

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

。 `smbShare` SMBボリュームを使用するFSx for ONTAPの場合、パラメータは必須です。

```
---
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}}_{{.volume.R
equestName}}"
},
"labels": {"cluster": "ClusterA", "PVC":
"{{.volume.Namespace}}_{{.volume.RequestName}}"}
```

仮想プールを使用するバックエンドの例

以下に示すサンプルのバックエンド定義ファイルでは、次のような特定のデフォルトがすべてのストレージプールに設定されています。 `spaceReserve` 「なし」の場合は、 `spaceAllocation` との誤り `encryption` 実行されます。仮想プールは、ストレージセクションで定義します。

Tridentでは、[Comments]フィールドにプロビジョニングラベルが設定されます。コメントは、のFlexVolまたはのFlexGroup `ontap-nas-flexgroup`で設定します `ontap-nas`。Tridentは、仮想プールに存在するすべてのラベルをプロビジョニング時にストレージボリュームにコピーします。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

これらの例では、一部のストレージプールが独自の `spaceReserve`、 `spaceAllocation`および `encryption` 値、および一部のプールはデフォルト値よりも優先されます。

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** にマッピングします

次のStorageClass定義は、を参照してください。 [仮想プールを使用するバックエンドの例]。を使用する `parameters.selector` フィールドでは、各StorageClassがボリュームのホスト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内で定義された要素があります。

- 。 `protection-gold` StorageClassは、 `ontap-nas-flexgroup` バックエンド：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、内の3番目と4番目の仮想プールにマッピングされます。 `ontap-nas-flexgroup` バックエンド：金色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は内の4番目の仮想プールにマッピングされます。 `ontap-nas` バックエンド：これは、mysqldbタイプ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"
```

- [t] `protection-silver-creditpoints-20k` StorageClassは、 `ontap-nas-flexgroup` バックエンド：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` バックエンドと内の2番目の仮想プール `ontap-nas-economy` バックエンド：これらは、5000クレジットポイントを持つ唯一のプールオファリングです。

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

Tridentが選択する仮想プールを決定し、ストレージ要件が満たされるようにします。

更新 `dataLIF` 初期設定後

初期設定後にデータLIFを変更するには、次のコマンドを実行して、更新されたデータLIFを新しいバックエンドJSONファイルに指定します。

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

PVCが1つ以上のポッドに接続されている場合は、対応するすべてのポッドを停止してから、新しいデータLIFを有効にするために稼働状態に戻す必要があります。

NetApp ONTAP 対応の Amazon FSX

Amazon FSx for NetApp ONTAPでTridentを使用

"NetApp ONTAP 対応の Amazon FSX" 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の起動や実行を可能にする、フルマネージドのAWSサービスです。FSX for ONTAP を使用すると、使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、AWSにデータを格納するためのシンプルさ、即応性、セキュリティ、拡張性を活用できます。FSX for ONTAP は、ONTAP ファイルシステムの機能と管理APIをサポートしています。

Amazon FSx for NetApp ONTAPファイルシステムをTridentと統合すると、Amazon Elastic Kubernetes Service（EKS）で実行されているKubernetesクラスタが、ONTAPを基盤とするブロックおよびファイルの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。

ファイルシステムは、オンプレミスの ONTAP クラスタに似た、 Amazon FSX のプライマリリソースです。各 SVM 内には、ファイルとフォルダをファイルシステムに格納するデータコンテナである 1 つ以上のボリュームを作成できます。Amazon FSX for NetApp ONTAP を使用すると、 Data ONTAP はクラウド内の管理対象ファイルシステムとして提供されます。新しいファイルシステムのタイプは * NetApp ONTAP * です。

TridentとAmazon FSx for NetApp ONTAPを使用すると、Amazon Elastic Kubernetes Service（EKS）で実行されているKubernetesクラスタが、ONTAPを基盤とするブロックおよびファイルの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"FSx for ONTAPとTridentを統合するには、さらに次のものが必要です。

- 既存の Amazon EKS クラスタまたは 'kubectl' がインストールされた自己管理型 Kubernetes クラスタ
- クラスタのワーカーノードから到達可能な既存のAmazon FSx for NetApp ONTAPファイルシステムおよびStorage Virtual Machine（SVM）。
- 準備されているワーカーノード "NFSまたはiSCSI"。

Amazon LinuxおよびUbuntuで必要なノードの準備手順を実行します "Amazon Machine Images の略" （ AMIS ） EKS の AMI タイプに応じて異なります。

考慮事項

- SMBボリューム：
  - SMBボリュームは、を使用してサポートされます `ontap-nas` ドライバーのみ。

- SMBボリュームは、Trident EKSアドオンではサポートされません。
- Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。詳細については、を参照してください "SMBボリュームをプロビジョニングする準備をします" 。

- Trident 24.02より前のバージョンでは、自動バックアップが有効になっているAmazon FSxファイルシステム上に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以降でこの問題を回避するには、AWS FSx for ONTAPのバックエンド構成ファイルで、 `apiRegion`AWS、AWS、およびAWS `apikey`を `secretKey`指定します `fsxFilesystemID`。

> TridentにIAMロールを指定する場合は、、 `apiKey`、および `secretKey`の各フィールドを`Trident`に明示的に指定する必要はありません `apiRegion`。詳細については、を参照してください "FSX（ONTAP の構成オプションと例）"。

認証

Tridentには2つの認証モードがあります。

- クレデンシャルベース（推奨）：クレデンシャルをAWS Secrets Managerに安全に格納します。ファイルシステムのユーザ、またはSVM用に設定されているユーザを使用できます `fsxadmin` `vsadmin` 。

> Tridentは、SVMユーザ、または別の名前で同じロールのユーザとして実行することを想定しています `vsadmin`。Amazon FSx for NetApp ONTAPには、ONTAPクラスタユーザに代わる限定的なユーザが `admin`い `fsxadmin`ます。`Trident`での使用を強くお勧めします `vsadmin`。

- 証明書ベース：Tridentは、SVMにインストールされている証明書を使用して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、使用しているドライバタイプの認証を参照してください。

- "ONTAP NAS認証"
- "ONTAP SAN認証"

詳細については、こちらをご覧ください

- "Amazon FSX for NetApp ONTAP のドキュメント"
- "Amazon FSX for NetApp ONTAP に関するブログ記事です"

IAMロールと**AWS Secret**を作成する

KubernetesポッドがAWSリソースにアクセスするように設定するには、明示的なAWSクレデンシャルを指定する代わりに、AWS IAMロールとして認証します。

> 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 Secret Managerシークレットの作成

次の例では、Trident CSIクレデンシャルを格納するAWSシークレットマネージャ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 "{"user":"vsadmin","password":"<svmpassword>"}"
```

IAMポリシーの作成

次の例は、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 Secret manager"
```

ポリシー**JSON**ファイル：

```
policy.json:
{
    "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"
}
```

サービスアカウントの作成とIAMロール

次の例では、EKSでサービスアカウント用の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 arn:aws:iam::aws:policy/service-
role/AmazonFSxNCSIDriverPolicy --approve
```

Astra Trident をインストール

Astra Tridentは、KubernetesでのAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や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。

Astra Tridentは次のいずれかの方法でインストールできます。

- Helm
- EKSアドオン

```
If you want to make use of the snapshot functionality, install the CSI
snapshot controller add-on. Refer to
https://docs.aws.amazon.com/eks/latest/userguide/csi-snapshot-
controller.html.
```

Helmを使用してAstra Tridentをインストール

1. Astra Tridentインストーラパッケージをダウンロード

   Astra Tridentインストーラパッケージには、Tridentオペレータの導入とAstra Tridentのインストールに必要なものがすべて含まれています。最新バージョンのAstra TridentインストーラをGitHubの[Assets]セクションからダウンロードして展開します。
   ```
   wget https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
   installer-24.10.0.tar.gz
   tar -xf trident-installer-24.10.0.tar.gz
   cd trident-installer
   ```

2. 次の環境変数を使用して、* cloud provider フラグと cloud identity *フラグの値を設定します。

   ```
   export CP="AWS"
   export CI="'eks.amazonaws.com/role-arn:
   arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'"
   ```

   次の例では、Astra Tridentをインストールし、フラグを `$CP`、 `cloud-identity`に `$CI`設定してい `cloud-provider`ます。

   ```
   helm install trident trident-operator-100.2410.0.tgz --set
   cloudProvider=$CP --set cloudIdentity=$CI --namespace trident
   ```

コマンドを使用して、名前、ネームスペース、グラフ、ステータス、アプリケーションのバージョン、リビジョン番号など、インストールの詳細を確認できます `helm list`。

```
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-100.2410.0    24.10.0
```

EKSアドオンを使用して**Astra Trident**をインストール

Astra Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSとの連携が検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

前提条件

AWS EKS用のAstra Tridentアドオンを設定する前に、次の条件を満たしていることを確認してください。

- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS Marketplace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向けの**Astra Trident**アドオンを有効にする

EKSクラスタ

次のコマンド例は、Astra 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 v24.6.1-eksbuild
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v24.6.1-eksbuild.1 （専用バージョンを使用）
```

オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

管理コンソール

1. で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、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを設定するクラスタの名前をクリックします。
4. をクリックし、[その他のアドオンの入手]*をクリックします。
5. [* S * elect add-ons *]ページで、次の手順を実行します。
   a. [AWS Marketplace EKS-addons]セクションで、* Astra Trident by NetApp *チェックボックスを選択します。
   b. 「 * 次へ * 」をクリックします。
6. [Configure selected add-ons* settings]ページで、次の手順を実行します。
   a. 使用する*バージョン*を選択します。
   b. では、[Not set]*のままにします。
   c. *オプションの構成設定*を展開し、*アドオン構成スキーマ*に従って、*構成値*セクションのconfigurationValuesパラメーター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 `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`）。[Conflict resolution method]で[Override]を選択すると、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しない場合、既存の設定と競合すると、操作は失敗します。表示された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に必要な設定を管理していないことを確認してください。

オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

7. [次へ]*を選択します。
8. [確認して追加]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、インストールされているアドオンが表示されます。

AWS CLI

1. ファイルを作成し `add-on.json` ます。

```
add-on.json
{

   "clusterName": "<eks-cluster>",
   "addonName": "netapp_trident-operator",
   "addonVersion": "v24.6.1-eksbuild.1",
   "serviceAccountRoleArn": "arn:aws:iam::123456:role/astratrident-
role",
   "configurationValues": "{"cloudIdentity":
"'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/astratrident-
role'",
   "cloudProvider": "AWS"}"
}
```

オプションのパラメータを設定する場合 `cloudIdentity`は `cloudProvider`、EKSアドオンを使用したTridentのインストール時にとしてを指定して`AWS`ください。

2. Install the Astra<xmt-block0> Trident</xmt-block> EKS add-on

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

Astra Trident EKSアドオンの更新

EKSクラスタ

- お使いの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。をクラスタ名に置き換え `my-cluster` 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    v24.6.1-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- 前の手順の出力でupdate availableで返されたバージョンにアドオンを更新します。
  ```
  eksctl update addon --name netapp_trident-operator --version v24.6.1-
  eksbuild.1 --cluster my-cluster --force
  ```

  オプションを削除し、いずれかのAmazon EKSアドオン設定が既存の設定と競合している場合 `--force` 、Amazon EKSアドオンの更新は失敗します。競合の解決に役立つ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、管理する必要がある設定がAmazon EKSアドオンで管理されていないことを確認してください。これらの設定はこのオプションで上書きされます。この設定のその他のオプションの詳細については、を参照してください "アドオン"。Amazon EKS Kubernetesフィールド管理の詳細については、を参照してください "Kubernetesフィールド管理"。

管理コンソール

1. Amazon EKSコンソールを開き https://console.aws.amazon.com/eks/home#/clustersます。
2. 左側のナビゲーションペインで、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを更新するクラスタの名前をクリックします。
4. [アドオン]タブをクリックします。
5. をクリックし、[Edit]*をクリックします。
6. [Configure Astra Trident by NetApp *]ページで、次の手順を実行します。
   a. 使用する*バージョン*を選択します。
   b. （オプション）* Optional configuration settings *を展開し、必要に応じて変更できます。
   c. [変更の保存 *] をクリックします。

AWS CLI

次の例では、EKSアドオンを更新します。

```
aws eks update-addon --cluster-name my-cluster netapp_trident-operator vpc-cni
--addon-version v24.6.1-eksbuild.1 \
--service-account-role-arn arn:aws:iam::111122223333:role/role-name
--configuration-values '{}' --resolve-conflicts --preserve
```

Astra Trident EKSアドオンのアンインストールと削除

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。

- クラスタにアドオンソフトウェアを保持–このオプションを選択すると、Amazon EKSによる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、コマンドのオプションをそのまま使用し `--preserve` ます。
- クラスタからアドオンソフトウェアを完全に削除する–クラスターに依存するリソースがない場合にのみ、Amazon EKSアドオンをクラスターから削除することをお勧めします。コマンドからオプションを削除してアドオンを削除し `--preserve delete` ます。

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。

EKSクラスタ

次のコマンドは、Astra 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
```

管理コンソール

1. で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、*[クラスタ]*をクリックします。
3. NetApp Trident CSIアドオンを削除するクラスタの名前をクリックします。
4. タブをクリックし、[Astra Trident by NetApp]をクリックします。
5. [削除（ Remove ）] をクリックします。
6. [Remove netapp_trident-operator confirmation]*ダイアログで、次の手順を実行します。
   a. Amazon EKSでアドオンの設定を管理しないようにするには、*[クラスタに保持]*を選択します。クラスタにアドオンソフトウェアを残して、アドオンのすべての設定を自分で管理できるようにする場合は、この手順を実行します。
   b. 「netapp_trident -operator *」と入力します。
   c. [削除（ Remove ）] をクリックします。

AWS CLI

をクラスタの名前に置き換え `my-cluster` 、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
```

ストレージバックエンドの設定

ONTAP SANとNASドライバの統合

バックエンドファイルは、次の例に示すように、AWS Secret Managerに保存されているSVMのクレデンシャ

ル（ユーザ名とパスワード）を使用して作成できます。

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
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"
  },
  "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"
    }
  }
}
```

バックエンドの作成については、次のページを参照してください。

- "バックエンドに ONTAP NAS ドライバを設定します"
- "バックエンドに ONTAP SAN ドライバを設定します"

FSx for ONTAPドライバの詳細

次のドライバを使用して、TridentとAmazon FSx for NetApp ONTAPを統合できます。

- `ontap-san`：プロビジョニングされる各PVは、それぞれの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る各PVは、完全なAmazon FSx for NetApp ONTAPボリュームです。NFSとSMBで推奨されます。
- 「 ONTAP と SAN の経済性」：プロビジョニングされた各 PV は、 NetApp ONTAP ボリュームの Amazon FSX ごとに構成可能な数の LUN を持つ LUN です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree であり、 NetApp ONTAP ボリュームの Amazon FSX ごとに設定可能な数の qtree があります。
- 「 ONTAP-NAS-flexgroup 」：プロビジョニングされた各 PV は、 NetApp ONTAP FlexGroup ボリューム用の完全な Amazon FSX です。

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

構成例

Secret Managerを使用した**AWS FSx for ONTAP**の設定

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

SMBホリユウムノストレエシクラスノセツテイ

を使用します `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クレデンシャルを指定できます。SMBボリュームは、を使用してサポートされます `ontap-nas` ドライバーのみ。

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

バックエンドの高度な設定と例

バックエンド設定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	説明	例
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	`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	ドライバ名 + "_" + データ LIF

パラメータ	説明	例
「管理 LIF 」	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。IPv6フラグを使用してTridentがインストールされている場合は、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、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。 `aws`フィールドでを指定する場合は `fsxFilesystemID`、を指定する必要はありませ `managementLIF`ん。`Trident`はAWSから`SVM`情報を取得するためです。 `managementLIF`そのため、`SVM`の下のユーザ（`vsadmin`など）のクレデンシャルを指定し、そのユーザにロールが割り当てられている必要があり `vsadmin` ます。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。* ONTAP NASドライバ*：データLIFを指定することを推奨します。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください 。* ONTAP SANドライバ*: iSCSIには指定しないでくださいTridentは、ONTAP選択的LUNマップを使用して、マルチパスセッションの確立に必要なiSCI LIFを検出します。データLIFが明示的に定義されている場合は警告が生成されます。IPv6フラグを使用してTridentがインストールされている場合は、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、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。	

パラメータ	説明	例
「 autoExportPolicy 」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy` と、Tridentでエクスポートポリシーを自動的に管理できます。	「偽」
「 autoExportCI` 」	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy` と、Tridentで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」」」
「ラベル」	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 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
「ユーザ名」	クラスタまたはSVMに接続するためのユーザ名。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のように指定します。	
「 password 」と入力します	クラスタまたは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。	
'VM'	使用する Storage Virtual Machine	SVM管理LIFが指定されている場合に生成されます。
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。作成後に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`

パラメータ	説明	例
「 AggreglimitateUsage 」と入力します	* Amazon FSx for NetApp ONTAP には指定しないでください。*指定されたと `vsadmin`には `fsxadmin`、アグリゲートの使用量を取得してTridentを使用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 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します `qtreesPerFlexvol` オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	“”（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大LUN数。有効な範囲は50、200です。SANのみ。	“100”
「バグトレースフラグ」	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}は使用されません `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
「 nfsvMountOptions 」のように入力します	NFSマウントオプションをカンマで区切ったリスト。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、ストレージクラスに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に指定されているマウントオプションを使用してフォールバックします。ストレージクラスまたは構成ファイルでマウントオプションが指定されていない場合、Tridentは関連付けられた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ボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb`、またはnull。*をに設定する必要があります `smb` SMBボリューム。*をnullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`

パラメータ	説明	例
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	"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にSMB共有の作成を許可する名前。このパラメータは、Amazon FSx for 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。技術プレビュー useREST は技術プレビューとして提供されており、本番環境のワークロードには推奨されません。に設定する true`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。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap 。これは、事前に定義された役割と役割によって実現され vsadmin cluster-admin ます。	「偽」
aws	AWS FSx for ONTAPの構成ファイルでは、次の項目を指定できます。 - 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シークレットキー。	"" "" ""
credentials	AWS Secret Managerに保存するFSx SVMのクレデンシャルを指定します。 - name：シークレットのAmazon Resource Name（ARN）。SVMのクレデンシャルが含まれています。 - type:に設定 awsarn。 を参照してください "AWS Secrets Managerシークレットの作成" を参照してください。	

これらのオプションを使用して、のデフォルトプロビジョニングを制御できます `defaults` 設定のセクション。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	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	「真」
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	「 NONE 」
「ナプショットポリシー」	使用する Snapshot ポリシー	「 NONE 」
「 QOSPolicy 」	作成し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ます。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コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。	「」
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。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ます。経済性に影響する ONTAP - NAS ではサポートされません。	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	状況 `snapshotPolicy` はです `none`、 `else` 「」
'plitOnClone	作成時にクローンを親からスプリットします	「偽」

パラメータ	説明	デフォルト
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。`false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。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すべてのボリュームで NAEが有効になります。詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	「偽」
`luksEncryption`	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。SANのみ。	""
階層ポリシー	使用する階層化ポリシー `none`	`snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておきます。*	「」
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。

SMBボリュームをプロビジョニングする準備をします

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-nas` ドライバ。をクリックしてください ONTAP SANとNASドライバの統合 次の手順を実行します。

作業を開始する前に

SMBボリュームをプロビジョニングする前に `ontap-nas` ドライバー、あなたは以下を持っている必要があります。

- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2019を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Active Directory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サービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub:

CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

手順

1. SMB共有を作成SMB管理共有は、のいずれかの方法で作成できます "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を使用します。ONTAP CLIを使用してSMB共有を作成するには、次の手順を実行します

   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、共有の作成時に-pathオプションで指定されているパスを確認します。指定したパスが存在しない場合、コマンドは失敗します。

   b. 指定したSVMに関連付けられている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
```

   を参照してください "SMB 共有を作成" 詳細については、

2. バックエンドを作成する際に、SMBボリュームを指定するように次の項目を設定する必要があります。ONTAP バックエンド構成オプションのすべてのFSXについては、を参照してください "FSX（ONTAP の構成オプションと例）"。

パラメータ	説明	例
`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にSMB共有の作成を許可する名前。このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
`nasType`	をに設定する必要があります **`smb`**. nullの場合、デフォルトはです `nfs`。	`smb`
'ecurityStyle'	新しいボリュームのセキュリティ形式。をに設定する必要があります **`ntfs`** または **`mixed`** **SMB**ボリューム	`ntfs` または `mixed` SMBボリュームの場合

パラメータ	説明	例
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておく必要があります。*	""

ストレージクラスと**PVC**を設定する

Kubernetes StorageClassオブジェクトを設定してストレージクラスを作成し、Tridentでボリュームのプロビジョニング方法を指定します。設定したKubernetes StorageClassを使用してPVへのアクセスを要求するPersistentVolume（PV）と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。

ストレージクラスを作成する。

Kubernetes StorageClassオブジェクトの設定

では、 "Kubernetes StorageClassオブジェクト"そのクラスで使用されるプロビジョニングツールとしてTridentが指定され、ボリュームのプロビジョニング方法がTridentに指示されます。例：

```
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"
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

ストレージクラスを作成する。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` をクリックして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ことを確認します。

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

PVおよびPVCの作成

A "永続ボリューム_" （PV）は、Kubernetesクラスタ上のクラスタ管理者がプロビジョニングする物理ストレージリソースです。。 "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設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVとPVCを作成したら、ポッドにボリュームをマウントできます。

マニフェストの例

PersistentVolumeサンプルマニフェスト

このサンプルマニフェストは、StorageClassに関連付けられた10Giの基本PVを示しています。 `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えたPVC

この例は、という名前のStorageClassに関連付けられたRWXアクセスを持つ基本的なPVCを示しています `basic-csi`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています。 `protection-gold`。

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

PVおよびPVCの作成

手順

1. PVを作成します。

```
kubectl create -f pv.yaml
```

2. PVステータスを確認します。

```
kubectl get pv
NAME         CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. PVC を作成します。

```
kubectl create -f pvc.yaml
```

4. 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
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

Trident属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	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ています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています

属性	を入力します	値	提供	リクエスト	でサポートされます
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、GCP-cvs、azure-NetApp-files 、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、ONTAP-NAS-エコノミー 、ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

^1 ^ ： ONTAP Select システムではサポートされていません

サンプルアプリケーションのデプロイ

サンプルアプリケーションをデプロイします。

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
```

次に、PVCをポッドに接続するための基本的な設定例を示します。基本設定：

```
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
```

進捗状況は次を使用して監視できます。 `kubectl get pod --watch`。

2. ボリュームがマウントされていることを確認します。 `/my/mount/path`。

```
kubectl exec -it task-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
```

1. ポッドを削除できるようになりました。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 task-pv-pod
```

EKSクラスタでの**Astra Trident EKS**アドオンの設定

Astra Tridentは、KubernetesでのAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や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。Astra Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSとの連携が検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

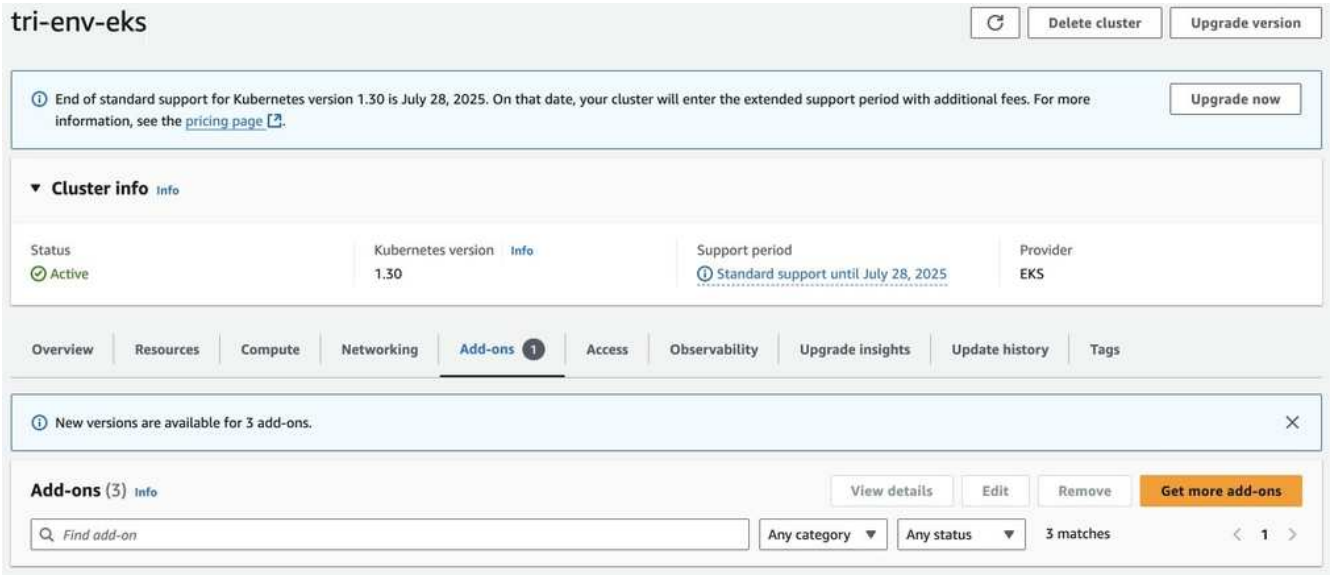
前提条件

AWS EKS用のAstra Tridentアドオンを設定する前に、次の条件を満たしていることを確認してください。

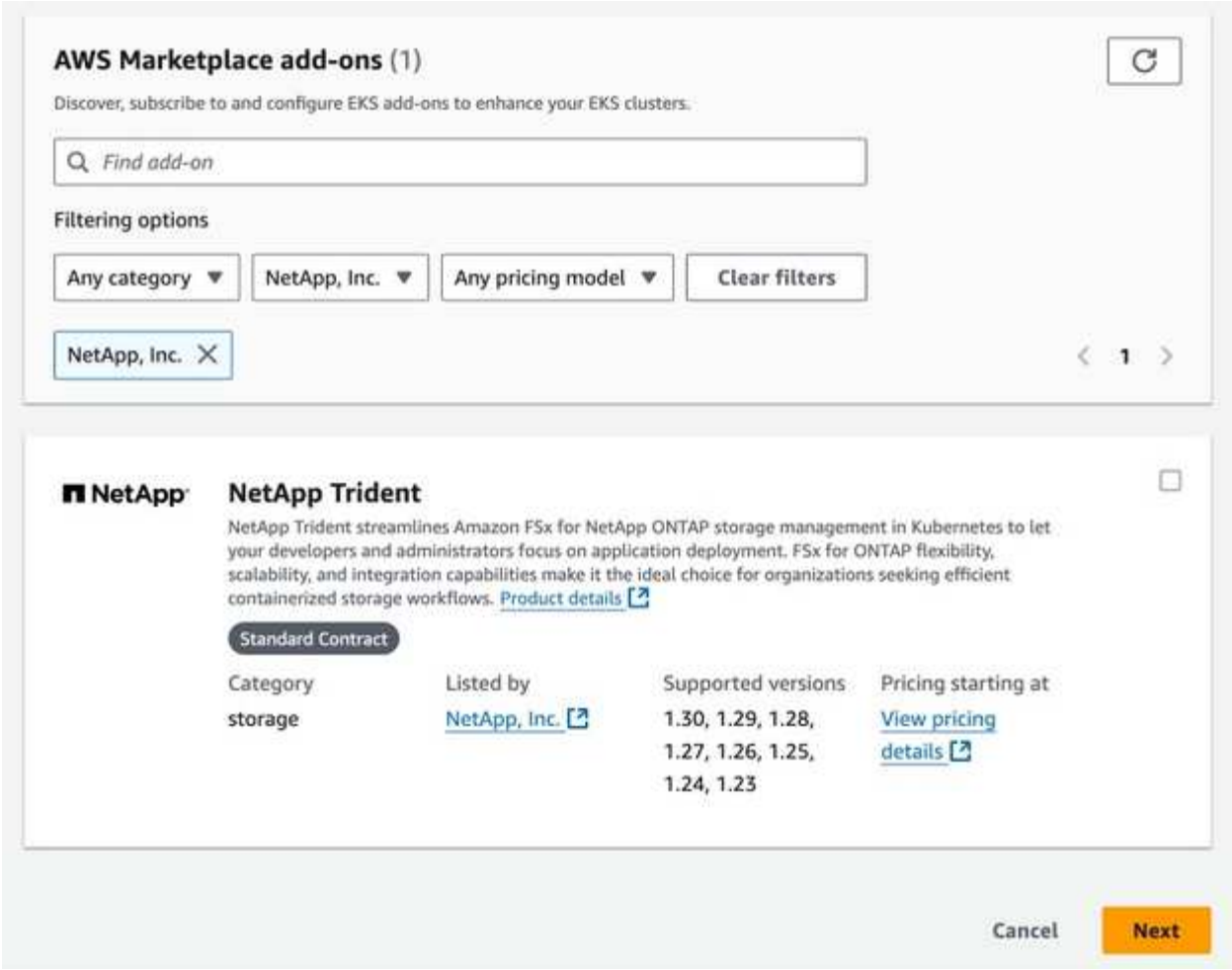
- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

手順

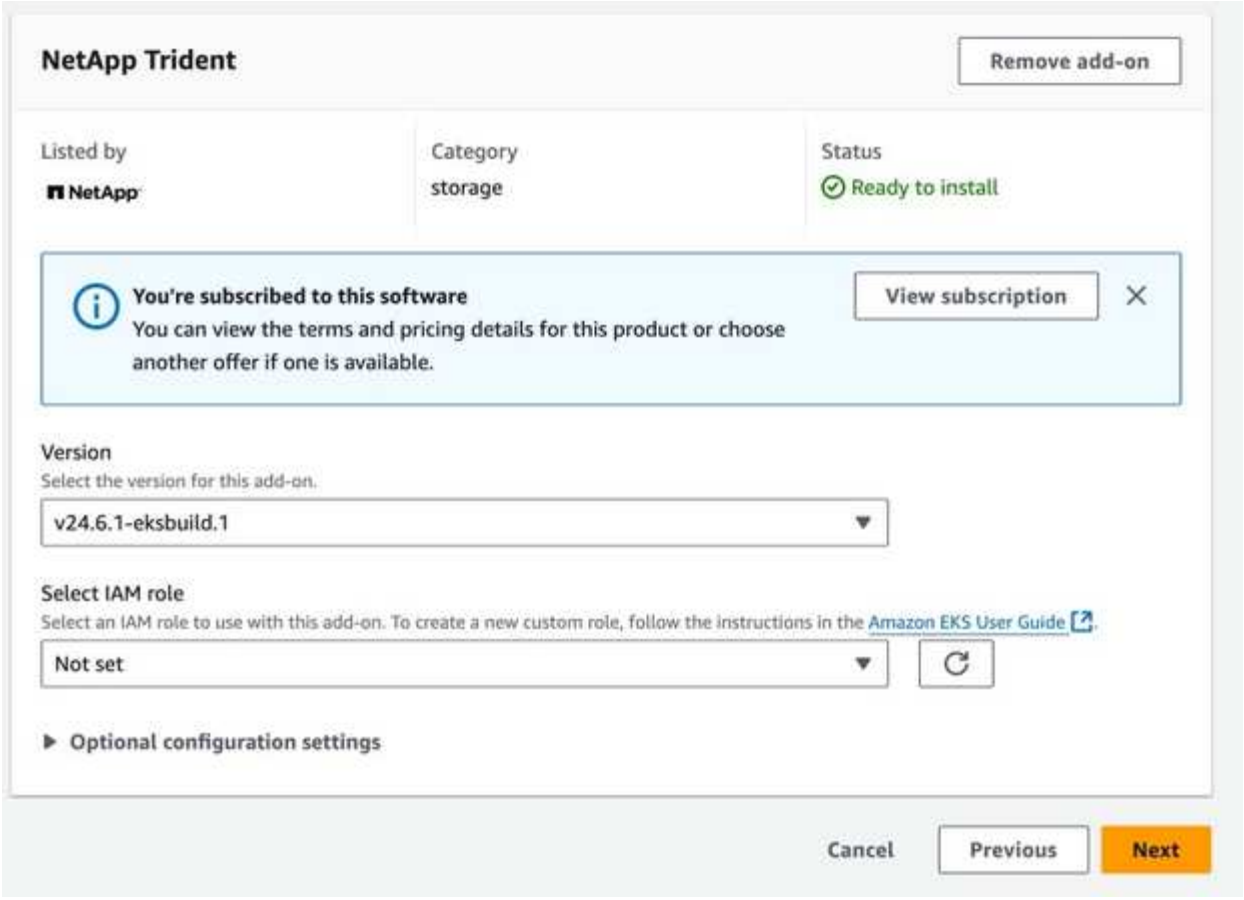
1. EKS Kubernetesクラスタで、*アドオン*タブに移動します。



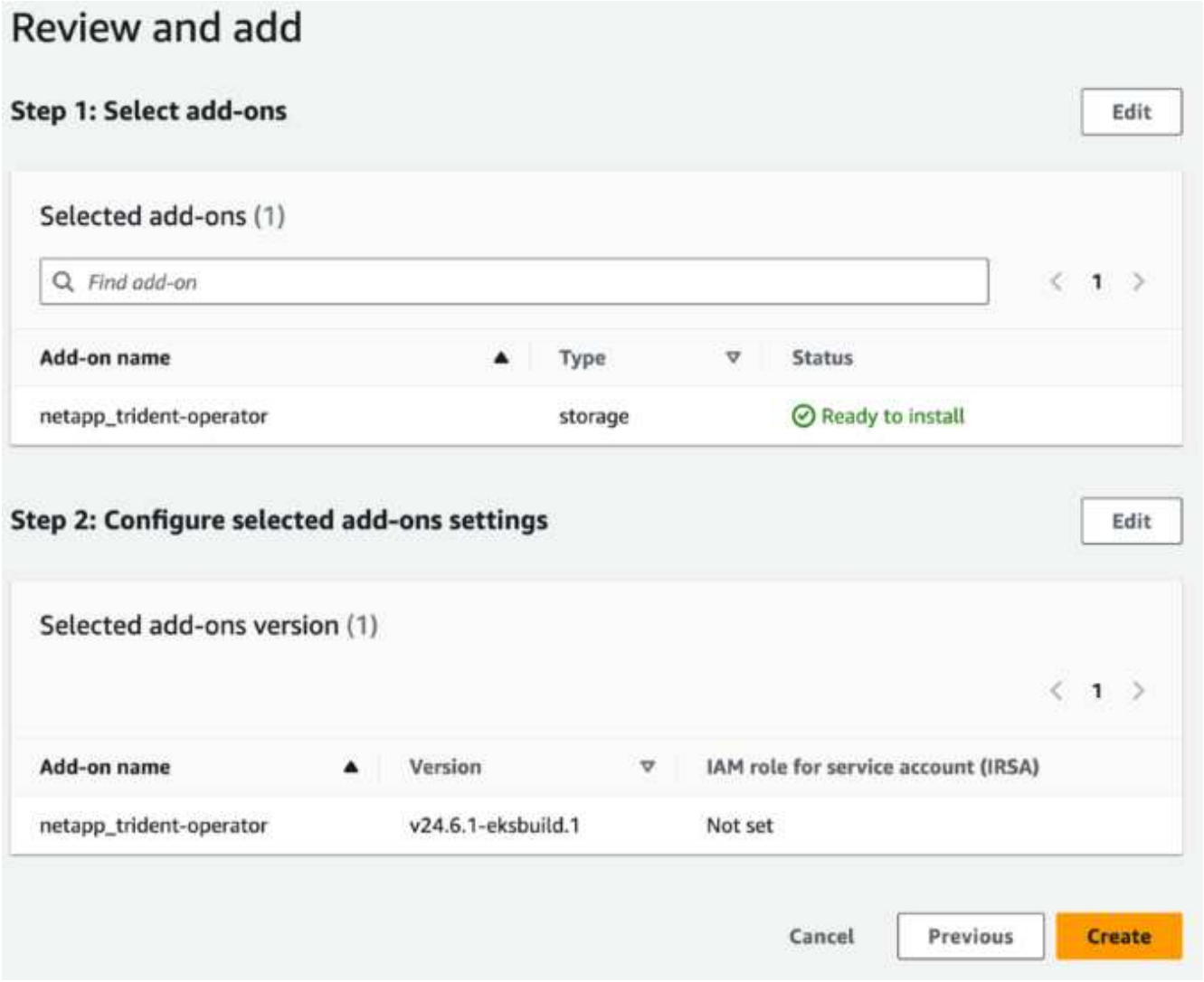
2. [AWS Marketplace add-ons]*にアクセスし、_storage_categoryを選択します。



3. NetApp Trident *を探し、Astra Tridentアドオンのチェックボックスを選択します。
4. 必要なアドオンのバージョンを選択します。



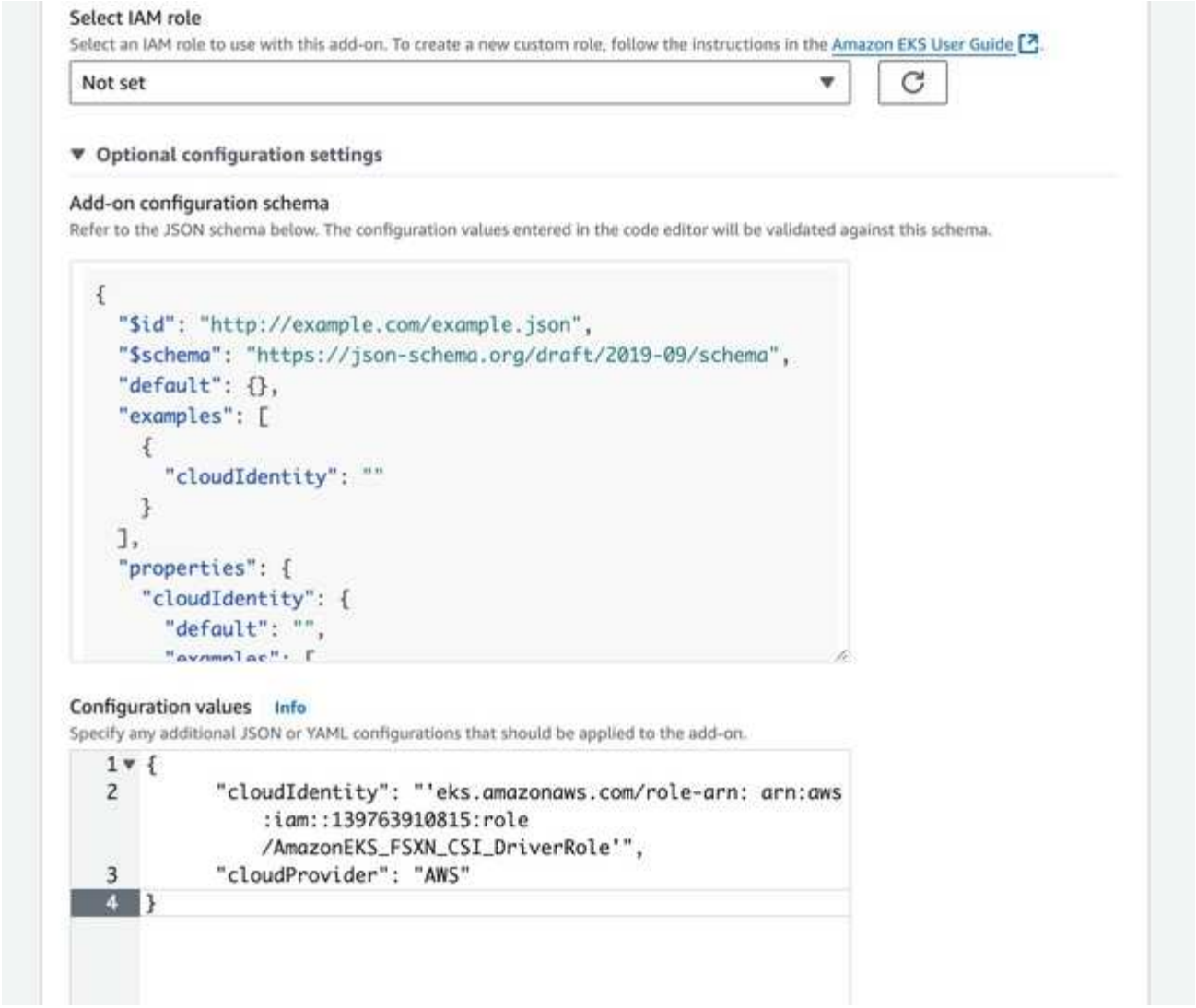
5. ノードから継承するIAMロールオプションを選択します。



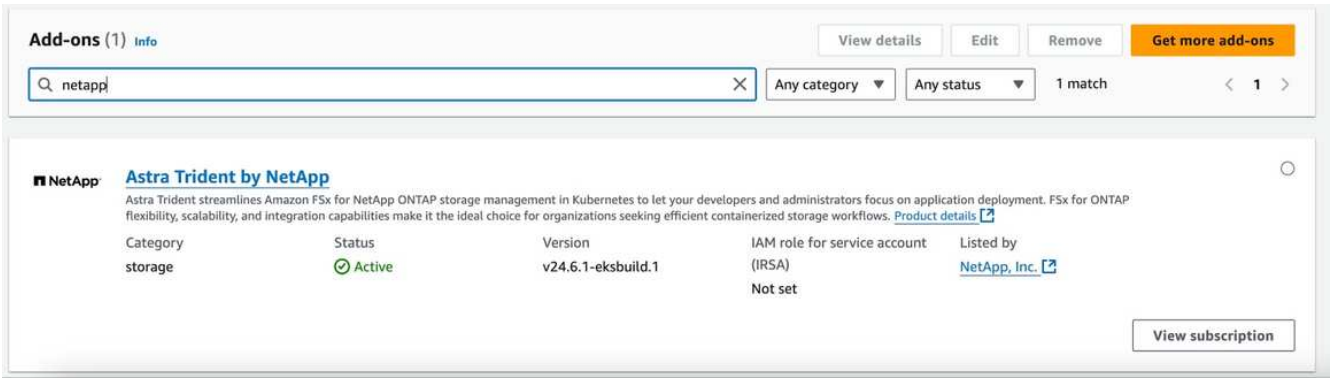
6. （オプション）必要に応じてオプションの設定を行い、* Next *を選択します。

Add-on構成スキーマ*に従って、* Configuration Values *セクションのconfigurationValuesパラメーター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 `eks.amazonaws.com/role-arn: arn:aws:iam::464262061435:role/AmazonEKS_FSXN_CSI_DriverRole`）。[Conflict resolution method]で[Override]を選択すると、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しない場合、既存の設定と競合すると、操作は失敗します。表示された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に必要な設定を管理していないことを確認してください。

オプションのパラメータを設定する場合 `cloudIdentity`は `cloudProvider`、EKSアドオンを使用したTridentのインストール時にとしてを指定して `AWS`ください。



7. 「* Create *」を選択します。
8. アドオンのステータスが_Active_であることを確認します。



CLIを使用した**Astra Trident EKS**アドオンのインストールとアンインストール

CLIを使用して**Astra Trident EKS**アドオンをインストールします。

次のコマンド例は、Astra Trident EKSアドオンをインストールします（専用バージョンを使用）。

```
eksctl create addon --cluster K8s-arm --name netapp_trident-operator --version
```

```
v24.6.1-eksbuild
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v24.6.1-eksbuild.1
```

オプションのパラメータを設定する場合 `cloudIdentity`は、EKSアドオンを使用してTridentをインストールするときにを指定して `cloudProvider`ください。

CLIを使用して**Astra 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、Astra 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 を使用してバックエンドを作成します

バックエンドは、Tridentとストレージシステムの間の関係を定義します。Tridentは、そのストレージシステムとの通信方法や、Tridentがそのシステムからボリュームをプロビジョニングする方法を解説します。Tridentをインストールしたら、次の手順でバックエンドを作成します。 `TridentBackendConfig`Custom Resource Definition（CRD）を使用すると、Kubernetesインターフェイスから直接Tridentバックエンドを作成および管理できます。これは、またはKubernetesディストリビューション用の同等のCLIツールを使用して実行できます `kubectl`。

TridentBackendConfig

TridentBackendConfig(tbc, tbconfig, tbackendconfig)は、を使用してTridentバックエンドを管理できるフロントエンドの名前空間CRDです。 `kubectl`Kubernetes管理者やストレージ管理者は、Kubernetes CLIを使用して直接バックエンドを作成、管理できるようになりまし(`tridentctl`た。専用のコマンドラインユーティリティは必要ありません）。

「 TridentBackendConfig 」オブジェクトを作成すると、次のようになります。

- バックエンドは、指定した設定に基づいてTridentによって自動的に作成されます。これは内部的には（tbe、 tridentbackend）CRとして表され `TridentBackend`ます。
- は TridentBackendConfig、Tridentによって作成されたに一意にバインドされます TridentBackend。

各「 TridentBackendConfig 」は、「 TridentBackend 」を使用して 1 対 1 のマッピングを維持します。前者はバックエンドの設計と構成をユーザに提供するインターフェイスで、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。

`TridentBackend`CRSはTridentによって自動的に作成されます。これらは * 変更しないでください。バックエンドを更新するには、オブジェクトを変更し `TridentBackendConfig`ます。

「 TridentBackendConfig 」 CR の形式については、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
```

の例を確認することもできます "Trident インストーラ" 目的のストレージプラットフォーム / サービスの設定例を示すディレクトリ。

。 `spec` バックエンド固有の設定パラメータを使用します。この例では、バックエンドはを使用します `ontap-san` storage driverおよびでは、に示す構成パラメータを使用します。ご使用のストレージドライバの設定オプションのリストについては、 "ストレージドライバのバックエンド設定情報"。

「 PEC 」セクションには、「 credentials 」フィールドと「 eleetionPolicy 」フィールドも含まれています。これらのフィールドは、「 TridentBackendConfig 」 CR に新しく導入されました。

- `credentials` ：このパラメータは必須フィールドで、ストレージシステム / サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- `DeleetionPolicy`: 「 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指定できます。
  - 「削除」：これにより、「 TridentBackendConfig 」 CR とそれに関連付けられた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。
  - 「管理」：「 TridentBackendConfig 」 CR を削除しても、バックエンド定義は引き続き表示され、「 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 」が作成された後で更新できます。

バックエンドの名前は 'PEC.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。'PEC.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お勧めします

で作成されたバックエンドに `tridentctl`は、関連付けられたオブジェクトはありません `TridentBackendConfig`。このようなバックエンドをで管理するには、 `kubectl`CRを作成し `TridentBackendConfig`ます。同一の設定パラメータ（、、 `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`。Tridentは、新しく作成されたを既存のバックエンドに自動的にバインドし `TridentBackendConfig`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 ' 次の手順を実行する必要があります

1. を作成し "Kubernetes Secret"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。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。

バックエンドを作成したら、「 kubectl get tbc <tbc-name> -n <trident-namespac>` 」を使用してバックエンドのステータスを確認し、詳細を収集できます。

手順 **1** ： **Kubernetes Secret** を作成します

バックエンドのアクセスクレデンシャルを含むシークレットを作成します。ストレージサービス / プラットフォームごとに異なる固有の機能です。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: t@Ax@7q(>
```

次の表に、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ます。

ストレージプラットフォームのシークレットフィールド概要	秘密	**Field** 概要の略
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID 。CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP

ストレージプラットフォームのシークレットフィールド概要	秘密	**Field** 概要の略
ONTAP	ユーザ名	クラスタ / SVM に接続するためのユーザ名。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 / 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	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」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。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
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。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合

このステップで作成されたシークレットは、次のステップで作成された「 TridentBackendConfig 」オブジェクトの「 PEC.credentials 」フィールドで参照されます。

手順**2**：を作成します `TridentBackendConfig` **CR**

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は ' 次に示す 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
```

手順**3**：のステータスを確認します `TridentBackendConfig` **CR**

これで「 TridentBackendConfig 」 CR が作成され、ステータスを確認できるようになりました。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が含まれています `configRef` をに設定します `TridentBackendConfig` crのuid
- Unbound ： "" を使用して表現されています「 TridentBackendConfig 」オブジェクトはバックエンドにバインドされません。新しく作成されたすべての TridentBackendConfig ' CRS は、デフォルトでこのフェーズに入ります。フェーズが変更された後、再度 Unbound に戻すことはできません。
- `Deleting`: `TridentBackendConfig` CR `deletionPolicy` が削除対象に設定されました。をクリックします `TridentBackendConfig` CRが削除され、削除状態に移行します。
  - バックエンドに永続的ボリューム要求（PVC）が存在しない場合、を削除する `TridentBackendConfig`と、`Trident`はバックエンドと`CR`を削除します `TridentBackendConfig`。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。次に 'TridentBackendConfig'CR が削除フェーズに入りますバックエンドおよび 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 に関連付けられているバックエンドが誤って削除されたか、意図的に削除されました。「 TridentBackendConfig 」 CR には削除されたバックエンドへの参照がありま

す。「 TridentBackendConfig 」 CR は、「 $eleetionPolicy 」の値に関係なく削除できます。

- `Unknown`：TridentはCRに関連付けられたバックエンドの状態または存在を特定できません `TridentBackendConfig`。たとえば、APIサーバが応答していない場合やCRDが見つからない場合 `tridentbackends.trident.netapp.io`などです。これには介入が必要な場合があります

この段階では、バックエンドが正常に作成されます。など、いくつかの操作を追加で処理することができます "バックエンドの更新とバックエンドの削除"。

（オプション）手順 **4** ：詳細を確認します

バックエンドに関する詳細情報を確認するには、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、「 TridentBackendConfig 」の YAML ／ JSON ダンプを取得することもできます。

```
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`CRに応答して作成されたバックエンドの `TridentBackendConfig`とが `backendUUID`格納され `backendName`ます。この `lastOperationStatus`フィールドには、CRの最後の操作のステータスが表示されます。このステータス `TridentBackendConfig`は、ユーザーがトリガーした場合（ユーザーがで何かを変更した場合など）、またはTridentによってトリガーされた場合 `spec`（Tridentの再起動中など）です。成功または失敗のいずれかです。 phase`CRとバックエンド間の関係のステータスを表します `TridentBackendConfig。上の例では、の phase`値がバインドされています。つまり、CRがバックエンドに関連付けられていることを意味します `TridentBackendConfig。

イベントログの詳細を取得するには、「 kubectl -n trident describe describe tbc <tbc -cr-name> 」コマンドを実行します。

> tridentctl を使用して ' 関連付けられた TridentBackendConfig' オブジェクトを含むバックエンドを更新または削除することはできません「 tridentctl 」と「 TridentBackendConfig 」の切り替えに関連する手順を理解するには、次の手順に従います。 "こちらを参照してください"。

バックエンドの管理

kubectl を使用してバックエンド管理を実行します

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します

を削除することで、 `TridentBackendConfig`（に基づいて）バックエンドを削除または保持するようにTridentに指示し `deletionPolicy`ます。バックエンドを削除するには、が`delete`に設定されていることを確認します `deletionPolicy`。のみを削除するには `TridentBackendConfig`、がretainに設定されていることを確認します `deletionPolicy`。これにより、バックエンドが引き続き存在し、を使用して管理できます `tridentctl`。

次のコマンドを実行します。

```
kubectl delete tbc <tbc-name> -n trident
```

Tridentでは、で使用されていたKubernetesシークレットは削除されません `TridentBackendConfig`。Kubernetes ユーザは、シークレットのクリーンアップを担当します。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してください。

既存のバックエンドを表示します

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します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新するには、オブジェクトで使用されるKubernetes Secretを `TridentBackendConfig`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で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（使用する ONTAP SVM の名前など）を更新する必要があります。
  - 更新できます `TridentBackendConfig` 次のコマンドを使用して、Kubernetesから直接オブジェクトを作成します。

```
kubectl apply -f <updated-backend-file.yaml>
```

- または、既存の `TridentBackendConfig` 次のコマンドを使用してCRを実行します。

```
kubectl edit tbc <tbc-name> -n trident
```

- バックエンドの更新に失敗した場合、バックエンドは最後の既知の設定のまま残ります。ログを表示して原因を確認するには、「 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。
- 構成ファイルで問題を特定して修正したら、 update コマンドを再実行できます。

tridentctl を使用してバックエンド管理を実行します

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します

を作成したら "バックエンド構成ファイル"を使用して、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、バックエンドの設定に何か問題があります。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
```

構成ファイルの問題を特定して修正したら ' 再度 create コマンドを実行します

バックエンドを削除します

Tridentからバックエンドを削除するには、次の手順を実行します。

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
```

TridentでプロビジョニングされたボリュームとこのバックエンドからSnapshotが残っている場合、バックエンドを削除すると、そのバックエンドで新しいボリュームがプロビジョニングされなくなります。バックエンドは「削除」状態のままになり、 Trident は削除されるまでそれらのボリュームとスナップショットを管理し続けます。

既存のバックエンドを表示します

Trident が認識しているバックエンドを表示するには、次の手順を実行します。

- 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
```

- すべての詳細を確認するには、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します

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、バックエンドの設定に問題があるか、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
```

構成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できます

バックエンドを使用するストレージクラスを特定します

ここでは ' バックエンド・オブジェクトの tridentctl 出力と同じ JSON を使用して回答で実行できる質問の例を示しますこれには 'jq' ユーティリティが使用されますこのユーティリティを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。

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ためのオプション

を導入しました `TridentBackendConfig`管理者は現在、バックエンドを2つの方法で管理できるようになっています。これには、次のような質問があります。

- tridentctl を使用して作成したバックエンドは 'TridentBackendConfig' で管理できますか
- 「 TridentBackendConfig 」を使用して作成したバックエンドは、「 tridentctl 」を使用して管理できますか。

管理 `tridentctl` を使用してバックエンドを `TridentBackendConfig`

このセクションでは 'tridentBackendConfig' オブジェクトを作成して Kubernetes インターフェイスから直接 'tridentctl を使用して作成されたバックエンドの管理に必要な手順について説明します

これは、次のシナリオに該当します。

- 既存のバックエンドには `TridentBackendConfig` を使用して作成されたためです `tridentctl`。
- 「 tridentctl 」で作成された新しいバックエンドと、その他の「 TridentBackendConfig 」オブジェクトが存在します。

どちらのシナリオでも、バックエンドは引き続き存在し、Tridentはボリュームをスケジューリングして処理します。管理者には次の 2 つの選択肢があります。

- tridentctl を使用して ' バックエンドを使用して作成したバックエンドを管理します
- 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は ' バックエンドが tridentctl ではなく 'kubectl' を使用して管理されることを意味します

「 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。その仕組みの概要を以下に示します。

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。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。同一の構成パラメータ (`PEC.backendName`'`PEC.storagePrefix`'`PEC.storageDriverName') を指定するように注意する必要があります'PEC.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

手順 **0** ：バックエンドを特定します

を作成します `TridentBackendConfig` 既存のバックエンドにバインドする場合は、バックエンド設定を取得する必要があります。この例では、バックエンドが次の JSON 定義を使用して作成されているとします。

```
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"
            }
        }
```

```
    ]
}
```

手順 **1** ： **Kubernetes Secret** を作成します

次の例に示すように、バックエンドのクレデンシャルを含むシークレットを作成します。

```
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
```

手順**2**：を作成します `TridentBackendConfig` **CR**

次の手順では '（この例のように）事前に存在する 'ONTAP-NAS-backend' に自動的にバインドされる 'TridentBackendConfig’CR を作成します次の要件が満たされていることを確認します。

- 「 'PEC.backendName' 」に同じバックエンド名が定義されています。
- 設定パラメータは元のバックエンドと同じです。
- 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である必要があります。
- 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
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
```

手順**3**：のステータスを確認します `TridentBackendConfig` **CR**

「 TridentBackendConfig 」が作成された後、そのフェーズは「バインド」されている必要があります。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

```
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 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

これで ' 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ます

管理 `TridentBackendConfig` を使用してバックエンドを `tridentctl`

tridentBackendConfig を使用して作成されたバックエンドを一覧表示するには 'tridentctl を使用しますまた、管理者は、「 TridentBackendConfig 」を削除し、「 pec.deletionPolicy` 」が「 re 」に設定されていることを確認することで、「 tridentctl 」を使用してこのようなバックエンドを完全に管理することもできます。

手順 **0** ：バックエンドを特定します

たとえば ' 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 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

出力からはそのことがわかります `TridentBackendConfig` は正常に作成され、バックエンドにバインドされています（バックエンドのUUIDを確認してください）。

手順**1**：確認します `deletionPolicy` がに設定されます `retain`

の価値を見てみましょう `deletionPolicy`。これはに設定する必要があり `retain`ます。これにより、CRが削除されてもバックエンド定義が存在し、で管理できるように `TridentBackendConfig`なり `tridentctl`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
```

「削除ポリシー」が「再取得」に設定されていない限り、次の手順に進まないでください。

手順**2**：を削除します `TridentBackendConfig` **CR**

最後の手順は、「 TridentBackendConfig 」 CR を削除することです。「削除ポリシー」が「取得」に設定されていることを確認したら、削除を続行できます。

```
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 |
+-------------------+----------------
+--------------------------------------+---------+----------+
```

オブジェクトが削除されると、 `TridentBackendConfig`Tridentは実際にはバックエンド自体を削除せずにオブジェクトを削除します。

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 StorageClassオブジェクトを設定してストレージクラスを作成し、Tridentでボリュームのプロビジョニング方法を指定します。

Kubernetes StorageClassオブジェクトの設定

は、 "Kubernetes StorageClassオブジェクト"そのクラスで使用されるプロビジョニングツールとしてTridentを識別し、ボリュームのプロビジョニング方法をTridentに指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

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。 [ストレージクラスノサンプル] に、使用または変更できる基本的なサンプルを示します。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` をクリックして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ことを確認します。

```
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"
        ]
      }
    }
  ]
}
```

ストレージクラスノサンプル

Tridentが提供し "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"ます。

または、 `sample-input/storage-class-csi.yaml.templ` インストーラに付属しており、 *`BACKEND_TYPE`* ストレージドライバの名前を指定します。

```
./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
```

ストレージクラスを管理する

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します

- 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
```

- 同期されたTridentの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ています。永続ボリューム要求（ PVC ）に永続ボリュームが指定されていない場合に、永続ボリュームのプロビジョニングに使用するストレージクラスです。

- ストレージクラスの定義でアノテーションの「 torageclass.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。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。
- 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設定できます。

```
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アノテーションを削除できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、このアノテーションが含まれている Trident インストーラバンドルにも例があります。

クラスタ内のデフォルトのストレージクラスは一度に1つだけにしてください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

これは、Tridentバックエンドオブジェクト用に出力するJSONを使用して回答できる質問の例 `tridentctl`です。これはユーティリティを使用し `jq`ます。このユーティリティは、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、ご使用のストレージクラスに置き換えてください。

このストレージクラスを使用して作成された永続ボリュームは変更されず、Tridentで引き続き管理されます。

> Tridentでは、作成するボリュームに対して空白が適用され `fsType`ます。`iSCSI`バックエンドの場合は、`StorageClass`で強制することを推奨します `parameters.fsType`。既存のストレージクラスを削除し、指定したで再作成してください `parameters.fsType`。

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

設定したKubernetes StorageClassを使用してPVへのアクセスを要求するPersistentVolume（PV）と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 "永続ボリューム_" （PV）は、Kubernetesクラスタ上のクラスタ管理者がプロビジョニングする物理ストレージリソースです。。 "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設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

PVとPVCを作成したら、ポッドにボリュームをマウントできます。

マニフェストの例

PersistentVolumeサンプルマニフェスト

このサンプルマニフェストは、StorageClassに関連付けられた10Giの基本PVを示しています。 `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

PersistentVolumeClaimサンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えたPVC

次の例は、という名前のStorageClassに関連付けられた、RWOアクセスが設定された基本的なPVCを示しています。 `basic-csi`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe / TCP対応PVC

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています。 `protection-gold`。

```
---
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

次の例は、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。

キホンセツテイ

```
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
```

NVMe/TCPの基本構成

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      resources: {}
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: pvc-san-nvme
```

PVおよび**PVC**の作成

手順

1. PVを作成します。

```
kubectl create -f pv.yaml
```

2. PVステータスを確認します。

```
kubectl get pv
NAME        CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. PVC を作成します。

```
kubectl create -f pvc.yaml
```

4. 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
```

5. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

> 進捗状況は次を使用して監視できます。 `kubectl get pod --watch`。

6. ボリュームがマウントされていることを確認します。 `/my/mount/path`。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

7. ポッドを削除できるようになりました。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 task-pv-pod
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

ボリュームを展開します

Tridentを使用すると、Kubernetesユーザは作成後にボリュームを拡張できます。ここでは、 iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、 iSCSI Persistent Volume （ PV ）を拡張できます。

> iSCSI ボリュームの拡張は 'ONTAP-SAN"ONTAP-SAN-エコノミー "olidfire-SAN' ドライバによってサポートされており 'Kubernetes 1.16 以降が必要です

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `true`。

```
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 'allowVolumeExpansion パラメータを含めるように編集します

手順 **2** ：作成した **StorageClass** を使用して **PVC** を作成します

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、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は永続的ボリューム（PV）を作成し、この永続的ボリューム要求（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します

サイズを変更するポッドにPVを接続します。iSCSI PV のサイズ変更には、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上のボリュームを拡張し、デバイスを再スキャンして、ファイルシステムのサイズを変更します。
- 接続されていない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上のボリュームを拡張します。PVC がポッドにバインドされると、 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。展開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、ポッドが作成され、「 1-pvc 」が使用されます。

```
 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** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.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
 ...
```

手順 **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、拡張が正常に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 ボリュームを拡張します

Tridentでは、、 `ontap-nas-economy` `ontap-nas-flexgroup`、、、 `gcp-cvs` `azure-netapp-files`` およびバックエンドでプロビジョニングされるNFS PVSのボリューム拡張がサポートされます ``ontap-nas`。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PV のサイズを変更するには ' まず 'allowVolumeExpansion フィールドを true に設定してボリュームを拡張できるようにストレージ・クラスを構成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
```

このオプションを指定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・クラスを編集するだけで ' ボリュームの拡張が可能になります

手順 **2** ：作成した **StorageClass** を使用して **PVC** を作成します

```
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用に20MiB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** を展開します

新しく作成した20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します `spec.resources.requests.storage` 1GiBへ：

```
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
...
```

手順 **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かどうかを検証できます。

```
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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
```

ボリュームをインポート

tridentctl import を使用して、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

Tridentにボリュームをインポートする目的は次のとおりです。

- 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はデータセットのクローンを使用
- 障害が発生したKubernetesクラスタを再構築します
- ディザスタリカバリ時にアプリケーションデータを移行

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（読み取り/書き込み）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。

- アクティブな接続がないボリュームをインポートすることを推奨します。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。

Kubernetesは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、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- PVCで指定する必要がありますが、 `StorageClass`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。そのため、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してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、最初にににに設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合 `delete`にすると、PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、TridentはPVCを管理し、バックエンドでFlexVolとLUNの名前を変更します。フラグを渡して管理対象外のボリュームをインポートできます `--no-manage`。を使用する場合 `--no-manage` 、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を実行しません。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その他の処理も無視されます。

このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外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便利です。

- PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーション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します

を使用できます `tridentctl import` をクリックしてボリュームをインポートします。

手順

1. Persistent Volume Claim（PVC；永続的ボリューム要求）ファイルを作成します（例： `pvc.yaml`）をクリックします。PVCファイルには、が含まれている必要があります `name`、 `namespace`、 `accessModes`および `storageClassName`。必要に応じて、を指定できます `unixPermissions` 定義されています。

   最小仕様の例を次に示します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class
```

PV名やボリュームサイズなどの追加のパラメータは指定しないでください。これにより原因、インポートコマンドが失敗する可能性があります。

2. コマンドを使用して `tridentctl import`、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（ONTAP FlexVol、Element Volume、Cloud Volumes Serviceパスなど）を指定します。 `-f`PVCファイルへのパスを指定するには、引数が必要です。

```
tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
```

例

サポートされているドライバについて、次のボリュームインポートの例を確認してください。

ONTAP NASおよびONTAP NAS FlexGroup

Tridentは、ドライバと `ontap-nas-flexgroup`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 `ontap-nas`ます

- 。 `ontap-nas-economy` ドライバでqtreeをインポートおよび管理できない。
- 。 `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

「 ontap/nas 」ドライバで作成される各ボリュームは、 ONTAP クラスタ上の FlexVol です。「 ontap/nas 」ドライバを使用して FlexVol をインポートする方法は同じです。ONTAP クラスタにすでに存在する FlexVol 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。

ONTAP NASの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

管理対象ボリューム

次の例は、という名前のボリュームをインポートします `managed_volume` という名前のバックエンドで `ontap_nas`：

```
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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理対象外のボリューム

引数を使用した場合 `--no-manage`、Tridentはボリュームの名前を変更しません。

次に、をインポートする例を示します `unmanaged_volume` をクリックします `ontap_nas` バックエンド：

```
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   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

ONTAP SAN

Tridentは、ドライバと `ontap-san-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 `ontap-san`ます

Tridentでは、単一のLUNを含むONTAP SAN FlexVolをインポートできます。これは、ドライバと一致してい `ontap-san`ます。ドライバは、PVCごとにFlexVolを作成し、FlexVol内にLUNを作成します。TridentはFlexVol

をインポートし、PVC定義に関連付けます。

ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリュームのインポートを示しています。

管理対象ボリューム

管理対象ボリュームの場合、TridentはFlexVolの名前を形式に、FlexVol内のLUNの名前をに `lun0`変更 `pvc-<uuid>`します。

次の例は、をインポートします `ontap-san-managed` にあるFlexVol `ontap_san_default` バックエンド：

```
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    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `ontap_san` バックエンド：

```
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ノードのIQNとIQNを共有するigroupにLUNをマッピングすると、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、イニシエータを削除するか、LUNのマッピング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
```

要素（ **Element** ）

Tridentは、NetApp Elementソフトウェアとドライバを使用したNetApp HCIボリュームインポートをサポートしています `solidfire-san`。

> Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合、Tridentはエラーを返します。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

要素の例

次に、をインポートする例を示します `element-managed` バックエンドのボリューム `element_default`。

```
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    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud Platform の **1** つです

Tridentは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 `gcp-cvs`ます。

NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、エクスポートパスがの場合などです `10.0.0.1:/adroit-jolly-swift`、ボリュームのパスはです `adroit-jolly-swift`。

Google Cloud Platformの例

次に、をインポートする例を示します `gcp-cvs` バックエンドのボリューム `gcpcvs_YEppr` を指定します `adroit-jolly-swift`。

```
tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-
file> -n trident

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files の特長

Tridentは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 `azure-netapp-files`ます。

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特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、マウントパスがの場合などです `10.0.0.2:/importvol1`、ボリュームのパスはです `importvol1`。

Azure NetApp Filesの例

次に、をインポートする例を示します `azure-netapp-files` バックエンドのボリューム `azurenetappfiles_40517` を指定します `importvol1`。

```
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    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

ボリュームの名前とラベルをカスタマイズする

Tridentでは、作成したボリュームにわかりやすい名前とラベルを割り当てることができます。これにより、ボリュームを特定し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできます。また、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、テンプレートに準拠します。

作業を開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

1. ボリュームの作成、インポート、クローニングの各処理。
2. ontap-nas-economyドライバの場合、qtreeボリュームの名前だけがテンプレート名に準拠します。
3. ontap-san-economyドライバの場合、名前テンプレートに準拠するのはLUN名のみです。

制限

1. カスタマイズ可能なボリューム名は、ONTAPオンプレミスドライバとのみ互換性があります。
2. カスタマイズ可能なボリューム名は、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

1. 名前テンプレートの無効な構文が原因でエラーが発生した場合、バックエンドの作成は失敗します。ただし、テンプレートアプリケーションが失敗した場合は、既存の命名規則に従ってボリュームに名前が付けられます。
2. 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の名前が指定されている場合、ストレージプレフィックスは適用されません。任意の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 }}"
      }
}
  ]
}
```

名前テンプレートの例

*例1 *：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

*例2 *：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

考慮すべきポイント

1. ボリュームインポートの場合、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： `{"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の後に続きます。
3. Tridentでは、storageプレフィックスを指定したスライス演算子の使用はサポートされていません。
4. テンプレートによってボリューム名が一意にならない場合、Tridentではいくつかのランダムな文字が追加されて一意のボリューム名が作成されます。
5. NASエコノミーボリュームのカスタム名の長さが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に名前を付けます。他のすべてのONTAPドライバでは、ボリューム名が名前の上限を超えると、ボリュームの作成プロセスが失敗します。

ネームスペース間で**NFS**ボリュームを共有します

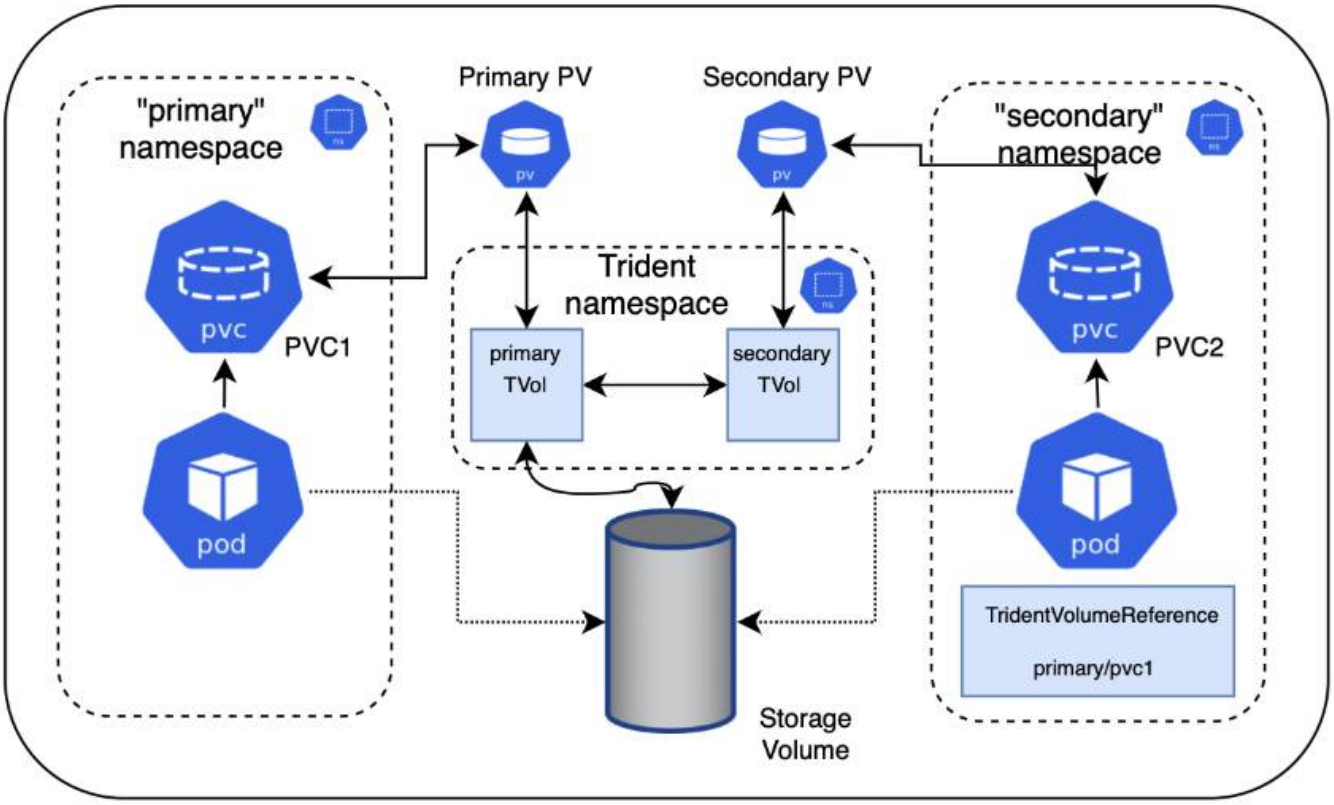
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

の機能

TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany（RWX）NFSボリュームを安全に共有できます。このKubernetesネイティブ解決策 には、次のようなメリットがあります。

- セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- すべてのTrident NFSボリュームドライバで動作
- tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。



クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1 ボリュームを共有するようにソース**PVC**を設定します

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2 デスティネーションネームスペースに**CR**を作成する権限を付与します

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3 デスティネーションネームスペースに**TridentVolumeReference**を作成します

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4 宛先名前空間に下位**PVC**を作成します

宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

ソースネームスペースとデスティネーションネームスペースを設定します

セキュリティを確保するために、ネームスペース間共有では、ソースネームスペースの所有者、クラスタ管理

者、および宛先ネームスペースの所有者によるコラボレーションとアクションが必要です。ユーザロールは各手順で指定します。

手順

1. ソース名前空間の所有者： PVCを作成します (`pvc1`）をソースネームスペースに追加し、デスティネーションネームスペースとの共有権限を付与します (`namespace2`)を使用します `shareToNamespace` アノテーション

```
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。

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例： `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- を使用して、すべてのネームスペースに共有できます `*`。例： `trident.netapp.io/shareToNamespace: *`
- PVCを更新してを含めることができます `shareToNamespace` アノテーションはいつでも作成できます。

2. *クラスタ管理者：*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先名前空間の所有者： PVCを作成します (`pvc2`）をデスティネーションネームスペースに展開します (`namespace2`)を使用します `shareFromPVC` 送信元PVCを指定する注釈。

```
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
```

宛先PVCのサイズは、送信元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の注釈を読み取り `shareFromPVC`、ソースPVストレージリソースを共有する独自のストレージリソースのない下位ボリュームとしてデスティネーションPVを作成します。宛先PVCとPVは、通常どおり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

複数のネームスペースで共有されているボリュームは削除できます。Tridentは、ソースネームスペース上のボリュームへのアクセスを削除し、そのボリュームを共有する他のネームスペースへのアクセスを維持します。このボリュームを参照しているネームスペースをすべて削除すると、Tridentによってボリュームが削除されます。

使用 `tridentctl get` 下位のボリュームを照会する

を使用する[`tridentctl` ユーティリティを使用すると、を実行できます `get` コマンドを使用して下位のボリュームを取得します。詳細については、リンク:./trident-reference/tridentctl.htmlを参照してください

[tridentctl コマンドとオプション]。

```
Usage:
  tridentctl get [option]
```

フラグ：

- `-h, --help：ボリュームのヘルプ。
- --parentOfSubordinate string：クエリを下位のソースボリュームに制限します。
- --subordinateOf string:クエリをボリュームの下位に制限します。

制限

- Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれないようにすることはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません shareToNamespace または shareFromNamespace 注釈またはを削除します TridentVolumeReference CR。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします "ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

CSI トポロジを使用します

Tridentでは、を使用して、Kubernetesクラスタ内のノードを選択的に作成して接続できます "CSI トポロジ機能"。

概要

CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、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なっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用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ではCSIトポロジを使用しています。

CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

- `VolumeBindingMode`をに設定する `Immediate`と、Tridentはトポロジを認識せずにボリュームを作成します。ボリュームバインディングと動的プロビジョニングは、 PVC が作成されるときに処理されます。これはデフォルト `VolumeBindingMode`であり、トポロジの制約を適用しないクラスタに適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。
- 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。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

「 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 トポロジ機能とは無関係に使用できます。

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- を実行するKubernetesクラスタ "サポートされるKubernetesバージョン"

```
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、トポロジ対応と `topology.kubernetes.io/zone`を示すラベルを付ける必要があり(`topology.kubernetes.io/region`ます。これらのラベル*は、Tridentをトポロジ対応にするためにTridentをインストールする前に、クラスタ内のノード*に設定しておく必要があります。

```
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"}]
```

手順 **1** ：トポロジ対応バックエンドを作成する

Tridentストレージバックエンドは、アベイラビリティゾーンに基づいて選択的にボリュームを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ブロックを運ぶことができます `supportedTopologies` 。ストレージクラスがそのようなバックエンドを使用する場合、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から要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

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

```
{
 "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"}
]
}
```

`supportedTopologies`は、バックエンドごとに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ストレージクラスの場合、Tridentはバックエンドにボリュームを作成します。

また ' ストレージ・プールごとに 'upportedTopologies を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、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

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、トポロジ情報を含めるように StorageClasses を定義できます。これにより、作成された PVC 要求の候補となるストレージプール、および Trident によってプロビジョニングされたボリュームを使用できるノードのサブセットが決まります。

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"
```

前述のStorageClass定義では、 `volumeBindingMode`がに設定されて `WaitForFirstConsumer`います。この `StorageClass` で要求された `PVC` は、ポッドで参照されるまで処理されません。およびに、 `allowedTopologies`使用するゾーンとリージョンを示します。StorageClassは `netapp-san-us-east1`、上記で定義したバックエンドにPVCを作成し`san-backend-us-east1`ます。

ステップ **3** ： **PVC** を作成して使用する

StorageClass を作成してバックエンドにマッピングすると、 PVC を作成できるようになりました。

以下の例「 PEC 」を参照してください。

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
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、次のような結果になります。

```
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 'us-east1` 領域に存在するノード上のポッドをスケジュールするよう Kubernetes に指示し 'us-east1-a' または `us-east1-b' ゾーン内に存在する任意のノードから選択します

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追加 `supportedTopologies`

既存のバックエンドは 'tridentctl backend update を使用して 'upportedTopologies のリストを含むように更新できますこれは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

- "コンテナのリソースを管理"
- "ノードセレクタ"
- "アフィニティと非アフィニティ"
- "塗料および耐性"

スナップショットを操作します

永続ボリューム（PV）のKubernetesボリュームSnapshotを使用すると、ボリュームのポイントインタイムコピーを作成できます。Tridentを使用して作成したボリュームのSnapshotの作成、Tridentの外部で作成したSnapshotのインポート、既存のSnapshotからの新しいボリュームの作成、Snapshotからのボリュームデータのリカバリを実行できます。

概要

ボリュームSnapshotは、でサポートされます `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `gcp-cvs`および `azure-netapp-files` ドライバ。

作業を開始する前に

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とカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

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、を参照してください ボリュームSnapshotコントローラの導入。

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では、内蔵の非表示の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** を作成します

手順

1. を作成します `VolumeSnapshotClass`。詳細については、を参照してください "ボリュームSnapshotクラス"。
   - は `driver`Trident CSIドライバを示しています。
   - `deletionPolicy` は、です `Delete` または `Retain`。に設定すると `Retain`を使用すると、ストレージクラスタの基盤となる物理`Snapshot`が、の場合でも保持されます `VolumeSnapshot` オブジェクトが削除された。

     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

   - 次に、既存のPVCのスナップショットを作成する例を示します。

```
cat snap.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

   - 次の例は、という名前のPVCのボリュームSnapshotオブジェクトを作成します。 `pvc1` Snapshotの名前はに設定されます `pvc1-snap`。ボリュームSnapshotはPVCに似ており、に関連付けられています `VolumeSnapshotContent` 実際のスナップショットを表すオブジェクト。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 次の情報を確認できます。 `VolumeSnapshotContent` のオブジェクト `pvc1-snap` ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。。 `Snapshot Content Name` このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 `Ready To Use` パラメータは、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

を使用できます `dataSource` という名前のVolumeSnapshotを使用してPVCを作成するには `<pvc-name>` データのソースとして。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> 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。を参照してください "KB：Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"。

次に、を使用してPVCを作成する例を示します。 `pvc1-snap` をデータソースとして使用します。

```
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**のインポート

Tridentでは、クラスタ管理者がを"Kubernetesの事前プロビジョニングされたSnapshotプロセス"使用して、オブジェクトを作成したり、Tridentの外部で作成されたSnapshotをインポートしたりできます `VolumeSnapshotContent`。

作業を開始する前に

TridentでSnapshotの親ボリュームが作成またはインポートされている必要があります。

手順

1. *クラスタ管理者：*バックエンドSnapshotを参照するオブジェクトを作成します `VolumeSnapshotContent`。これにより、TridentでSnapshotワークフローが開始されます。
   - バックエンドスナップショットの名前を `annotations` として `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - で指定します `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` `snapshotHandle`。この情報は、呼び出しで外部スナップショットによってTridentに提供される唯一の情報です `ListSnapshots`。

     。 `<volumeSnapshotContentName>` CRの命名規則のため、バックエンドスナップショット名が常に一致するとは限りません。

     例

     次の例では、 `VolumeSnapshotContent` バックエンドスナップショットを参照するオブジェクト `snap-01`。

```
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>
```

2. クラスタ管理者： `VolumeSnapshot` を参照するCR `VolumeSnapshotContent` オブジェクト。これにより、 `VolumeSnapshot` 指定された名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap` を参照しています。 `VolumeSnapshotContent` 名前付き `import-snap-content`。

```
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（アクション不要）：*外部スナップショットは、新しく作成されたを認識して `VolumeSnapshotContent`呼び出しを実行します `ListSnapshots`。Tridentによってが作成され`TridentSnapshot`ます。
   - 外部スナップショットは、 `VolumeSnapshotContent` 終了： `readyToUse` および `VolumeSnapshot` 終了： `true`。
   - Tridentのリターン `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim` 新しい `VolumeSnapshot`を参照してください `spec.dataSource` （または `spec.dataSourceRef`）nameは `VolumeSnapshot` 名前。

   例

   次に、を参照するPVCを作成する例を示します。 `VolumeSnapshot` 名前付き `import-snap`。

```
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
```

Snapshotを使用してボリュームデータをリカバリします

snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` スナップショットからデータを直接リカバリするディレクトリ。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

Snapshotが関連付けられている**PV**を削除する

スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。ボリュームSnapshotを削除してTridentボリュームを削除します。

ボリューム**Snapshot**コントローラの導入

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、次のように導入できます。

手順

1. ボリュームのSnapshot作成

```
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. スナップショットコントローラを作成します。

```
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
```

必要に応じて、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

関連リンク

- "ボリューム Snapshot"
- "ボリュームSnapshotクラス"

Tridentの管理と監視

Tridentのアップグレード

Tridentのアップグレード

24.02リリース以降、Tridentはリリースサイクルを4カ月に短縮し、毎年3つのメジャーリリースを提供しています。新しいリリースは、以前のリリースに基づいて構築され、新機能、パフォーマンスの強化、バグの修正、および改善が提供されます。Tridentの新機能を利用するには、少なくとも年に1回アップグレードすることをお勧めします。

アップグレード前の考慮事項

Tridentの最新リリースにアップグレードする場合は、次の点を考慮してください。

- 特定のKubernetesクラスタ内のすべてのネームスペースには、Tridentインスタンスを1つだけインストールする必要があります。
- Trident 23.07以降では、v1ボリュームスナップショットが必要です。alphaまたはbetaスナップショットはサポートされません。
- でCloud Volumes Service for Google Cloudを作成した場合"CVS サービスタイプ"は、Trident 23.01からのアップグレード時にまたは `zoneredundantstandardsw`サービスレベルを使用するようにバックエンド構成を更新する必要があります `standardsw`。バックエンドでを更新しない `serviceLevel`と、ボリュームで障害が発生する可能性があります。詳細については、を参照してください "CVSサービスタイプのサンプル" 。
- をアップグレードする場合は、を `StorageClasses`Tridentで使用するために指定することが重要です `parameter.fsType`。既存のボリュームを停止することなく、削除や再作成を実行できます `StorageClasses`。
  - これは、強制の 要件 です "セキュリティコンテキスト" SAN ボリュームの場合。
  - sample inputディレクトリには、https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templなどの例が含まれています[`storage-class-basic.yaml.templ`）とリンク：https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-bronze-default.yaml[`storage-class-bronze-default.yaml`^]をクリックします。
  - 詳細については、を参照してください "既知の問題"。

ステップ**1**：バージョンを選択します

Tridentバージョンは、日付ベースの命名規則に従い `YY.MM`ます。「YY」は年の最後の2桁、「mm」は月です。ドットリリースは規則に従い `YY.MM.X`ます。「X」はパッチレベルです。アップグレード前のバージョンに基づいて、アップグレード後のバージョンを選択します。

- インストールされているバージョンの4リリースウィンドウ内にある任意のターゲットリリースに直接アップグレードできます。たとえば、23.04（または任意の23.04 DOTリリース）から24.06に直接アップグレードできます。
- 4つのリリースウィンドウ以外のリリースからアップグレードする場合は、複数の手順でアップグレードを実行します。4リリースのウィンドウに適合する最新リリースにアップグレードするには、アップグレ

ード元ののののアップグレード手順を使用し "以前のバージョン" ます。たとえば、22.01を実行していて、24.06にアップグレードする場合は、次の手順を実行します。

a. 22.07から23.04への最初のアップグレード。

b. その後、23.04から24.06にアップグレードします。

> OpenShift Container PlatformでTridentオペレータを使用してアップグレードする場合は、Trident 21.01.1以降にアップグレードする必要があります。21.01.0 でリリースされた Trident オペレータには、 21.01.1 で修正された既知の問題が含まれています。詳細については、 "GitHub の問題の詳細"。

ステップ**2:**元のインストール方法を決定します

Tridentを最初にインストールしたバージョンを確認するには、次の手順を実行します。

1. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してTridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Tridentは手動またはHelmを使用してTridentオペレータを使用してインストールされています。
2. オペレータポッドがある場合は、を使用して、 `kubectl describe torc`TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合、TridentはHelmを使用してインストールされています。
   - Helmラベルがない場合、TridentはTridentオペレータを使用して手動でインストールされています。

ステップ**3**：アップグレード方法を選択します

通常は、最初のインストールと同じ方法でアップグレードする必要がありますが、可能です。"インストール方法を切り替えます"Tridentをアップグレードする方法は2つあります。

- "Tridentオペレータを使用してアップグレード"

> レビューすることをお勧めします "オペレータのアップグレードワークフローについて理解する" オペレータでアップグレードする前に。

*

オペレータにアップグレードしてください

オペレータのアップグレードワークフローについて理解する

Tridentオペレータを使用してTridentをアップグレードする前に、アップグレード中に発生するバックグラウンドプロセスについて理解しておく必要があります。これには、Tridentコントローラ、コントローラポッドとノードポッド、およびローリング更新を可能にするノードデーモンセットに対する変更が含まれます。

Tridentオペレータのアップグレード処理

Tridentをインストールしてアップグレードするには"Tridentオペレータを使用するメリット"、既存のマウントボリュームを中断することなく、TridentオブジェクトとKubernetesオブジェクトを自動的に処理する必要があります。このようにして、Tridentはダウンタイムなしでアップグレードをサポートできます。"ローリング更新"TridentオペレータはKubernetesクラスタと通信して次のことを行います。

- Trident Controller環境とノードデーモンセットを削除して再作成します。
- TridentコントローラポッドとTridentノードポッドを新しいバージョンに置き換えます。
  - 更新されていないノードは、残りのノードの更新を妨げません。
  - ボリュームをマウントできるのは、Trident Node Podを実行しているノードだけです。

KubernetesクラスタのTridentアーキテクチャの詳細については、を参照してください"Tridentのアーキテクチャ"。

オペレータのアップグレードワークフロー

Tridentオペレータを使用してアップグレードを開始すると、次の処理が実行されます。

1. Trident演算子*：
   a. 現在インストールされているTridentのバージョン（version_n_）を検出します。
   b. CRD、RBAC、Trident SVCなど、すべてのKubernetesオブジェクトを更新
   c. version_n_用のTrident Controller環境を削除します。
   d. version_n+1_用のTrident Controller環境を作成します。
2. * Kubernetes *は、_n+1_用にTridentコントローラポッドを作成します。
3. Trident演算子*：
   a. _n_のTridentノードデーモンセットを削除します。オペレータは、Node Podが終了するのを待たない。
   b. _n+1_のTridentノードデーモンセットを作成します。
4. * Kubernetes * Trident Node Pod_n_を実行していないノードにTridentノードポッドを作成します。これにより、1つのノードに複数のTrident Node Pod（バージョンに関係なく）が存在することがなくなります。

Trident operatorまたはHelmを使用したTridentインストールのアップグレード

Tridentは、Tridentオペレータを使用して手動でアップグレードすることも、Helmを使用してアップグレードすることもできます。Tridentオペレータのインストールから別のTridentオペレータのインストールにアップグレードすることも、インストールからTridentオペレータのバージョンにアップグレードすることもできます `tridentctl` 。Trident Operatorのインストールをアップグレードする前にを参照してください"アップグレード方法を選択します"。

手動インストールのアップグレード

クラスタを対象としたTridentオペレータインストールから、クラスタを対象とした別のTridentオペレータインストールにアップグレードできます。バージョン21.01以降のTridentでは、すべてクラスタを対象とした演

算子が使用されます。

> 名前空間を対象とした演算子（バージョン20.07～20.10）を使用してインストールされたTridentからアップグレードするには、Tridentののアップグレード手順を使用します"インストールされているバージョン"。

このタスクについて

Tridentにはバンドルファイルが用意されています。このファイルを使用して、オペレータをインストールしたり、Kubernetesバージョンに対応する関連オブジェクトを作成したりできます。

- クラスタでKubernetes 1.24を実行している場合は、を使用し "Bundle_pre_1_25.yaml"ます。
- クラスタでKubernetes 1.25以降を実行している場合は、を使用します "bundle_post_1_25.yaml"。

作業を開始する前に

を実行しているKubernetesクラスタを使用していることを確認します "サポートされるKubernetesバージョン"。

手順

1. Tridentのバージョンを確認します。

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

2. 現在のTridentインスタンスのインストールに使用したTridentオペレータを削除します。たとえば、23.07からアップグレードする場合は、次のコマンドを実行します。

```
kubectl delete -f 23.07.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

3. を使用して初期インストールをカスタマイズした場合 `TridentOrchestrator` 属性を編集できます `TridentOrchestrator` インストールパラメータを変更するオブジェクト。これには、ミラーリングされたTridentおよびCSIイメージレジストリをオフラインモードに指定したり、デバッグログを有効にしたり、イメージプルシークレットを指定したりするための変更が含まれます。
4. ご使用の環境に適したバンドルYAMLファイルを使用してTridentをインストールします（__は、または `bundle_post_1_25.yaml`使用している<bundle.yaml> `bundle_pre_1_25.yaml`のバージョンに基づいています）。たとえば、Trident 24.10をインストールする場合は、次のコマンドを実行します。

```
kubectl create -f 24.10.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

Helmインストールのアップグレード

Trident Helmのインストールをアップグレードできます。

Tridentがインストールされ`helm upgrade`ているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は、クラスタをアップグレードする前に、values.yamlをに`true`設定するかコマンドに追加する`--set excludePodSecurityPolicy=true`ように更新する必要があります。`excludePodSecurityPolicy`

Trident Helmをアップグレードせずにクラスタを1.24から1.25にアップグレードした場合、Helmのアップグレードは失敗します。Helmのアップグレードを実行するには、次の手順を前提条件として実行します。

1. からhelm-mapkubeapisプラグインをインストールします https://github.com/helm/helm-mapkubeapis。
2. Tridentがインストールされているネームスペースで、Tridentリリースのドライランを実行します。リソースが一覧表示され、クリーンアップされます。

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. クリーンアップを実行するには、helmを使用してフルランを実行します。

```
helm mapkubeapis trident  --namespace trident
```

手順

1. を使用する "Helmを使用してTridentをインストール"と、を使用してワンステップでアップグレードできます`helm upgrade trident netapp-trident/trident-operator --version 100.2410.0`。Helmリポジトリを追加しなかった場合、またはHelmリポジトリを使用してアップグレードできない場合は、次の手順を実行します。
   a. から最新のTridentリリースをダウンロードし"GitHubの_Assets_sectionを参照してください"ます。
   b. コマンドを使用し`helm upgrade`ます。は、`trident-operator-24.10.0.tgz`アップグレード先のバージョンを反映しています。

```
helm upgrade <name> trident-operator-24.10.0.tgz
```

   初期インストール時にカスタムオプションを設定した場合（TridentイメージとCSIイメージのプライベートなミラーレジストリの指定など）は、`helm upgrade`コマンド`--set`これらのオプションがupgradeコマンドに含まれるようにするため、それらのオプションの値をdefaultにリセットします。

2. を実行します`helm list`グラフとアプリのバージョンが両方ともアップグレードされていることを確認します。を実行します`tridentctl logs`デバッグメッセージを確認します。

からのアップグレード`tridentctl`**Trident**オペレータへのインストール

からTridentの最新リリースにアップグレードできます`tridentctl`インストール：既存のバックエンドとPVCは自動的に使用可能になります。

インストール方法を切り替える前に、 "インストール方法を切り替える"。

手順

1. 最新のTridentリリースをダウンロードします。

```
# Download the release required [24.10.0]
mkdir 24.10.0
cd 24.10.0
wget
https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

2. マニフェストから「 tridentオーケストラ 」 CRD を作成します。

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

3. クラスタを対象としたオペレータを同じネームスペースに導入します。

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

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

4. TridentをインストールするためのCRを作成し `TridentOrchestrator`ます。

```
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m
trident-csi-xrst8                   2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0          5m41s
```

5. Tridentが目的のバージョンにアップグレードされたことを確認

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v24.10.0
```

tridentctl を使用してアップグレードします

を使用して、既存のTridentインストールを簡単にアップグレードできます `tridentctl`。

このタスクについて

Tridentのアンインストールと再インストールは、アップグレードとして機能します。Tridentをアンインストールしても、Trident環境で使用されている永続的ボリューム要求（PVC）と永続的ボリューム（PV）は削除されません。すでにプロビジョニングされているPVCは、Tridentがオフラインの間も使用できます。また、その間に作成されたPVCがオンラインに戻ったあとも、Tridentはボリュームをプロビジョニングします。

作業を開始する前に

レビュー "アップグレード方法を選択します" を使用してアップグレードする前に `tridentctl`。

手順

1. のuninstallコマンドを実行し `tridentctl`て、CRDと関連オブジェクトを除くTridentに関連付けられているすべてのリソースを削除します。

```
./tridentctl uninstall -n <namespace>
```

2. Tridentを再インストールします。を参照してください "tridentctlを使用したTridentのインストール"。

アップグレードプロセスを中断しないでください。インストーラが完了するまで実行されることを確認します。

tridentctlを使用したTridentの管理

には、 "Trident インストーラバンドル"Tridentに簡単にアクセスできるコマンドラインユーティリティが含まれてい `tridentctl`ます。十分なPrivilegesを持つKubernetesユーザは、Tridentをインストールしたり、Tridentポッドを含むネームスペースを管理したりできます。

コマンドとグローバルフラグ

走れ `tridentctl help` 使用可能なコマンドのリストを取得するには `tridentctl` または、 `--help` 特定のコマンドのオプションとフラグのリストを取得するには、任意のコマンドにフラグを付けます。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl`ユーティリティは、次のコマンドとグローバルフラグをサポートしています。

コマンド

create
Tridentにリソースを追加します。

delete
Tridentから1つ以上のリソースを削除します。

get
Tridentから1つ以上のリソースを取得します。

help
任意のコマンドに関するヘルプ。

images
Tridentが必要とするコンテナイメージの表を印刷します。

import
既存のリソースをTridentにインポートします。

install
Trident をインストール

logs
Tridentからログを印刷します。

send
Tridentからリソースを送信します。

uninstall
Tridentをアンインストールします。

update
Tridentでリソースを変更します。

update backend state
バックエンド処理を一時的に中断します。

upgrade
Tridentでリソースをアップグレードします。

「バージョン」
Tridentのバージョンを印刷します。

グローバルフラグ

-d、 --debug
デバッグ出力。

-h、 --help
ヘルプ `tridentctl`。

-k、 --kubeconfig string
を指定します。 `KUBECONFIG` コマンドをローカルまたはKubernetesクラスタ間で実行するパス。

または、 `KUBECONFIG` 特定のKubernetesクラスタと問題をポイントする変数 `tridentctl` そのクラスタにコマンドを送信します。

-n、 --namespace string
Trident環境のネームスペース。

-o、 --output string
出力形式。JSON の 1 つ | yaml | name | wide | ps （デフォルト）。

-s、 --server string
Trident RESTインターフェイスのアドレス/ポート。

Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

コマンドオプションとフラグ

作成

コマンドを使用し `create`て、Tridentにリソースを追加します。

```
tridentctl create [option]
```

オプション（ **Options** ）
`backend`：Tridentにバックエンドを追加します。

削除

コマンドを使用し `delete`て、Tridentから1つ以上のリソースを削除します。

```
tridentctl delete [option]
```

オプション（ **Options** ）
`backend`：Tridentから1つ以上のストレージバックエンドを削除します。
`snapshot`：Tridentから1つ以上のボリュームSnapshotを削除します。

`storageclass`：Tridentから1つ以上のストレージクラスを削除します。
`volume`：Tridentから1つ以上のストレージボリュームを削除します。

取得

コマンドを使用し `get`て、Tridentから1つ以上のリソースを取得します。

```
tridentctl get [option]
```

オプション（ **Options** ）

`backend`：Tridentから1つ以上のストレージバックエンドを取得します。
`snapshot`：Tridentから1つ以上のスナップショットを取得します。
`storageclass`：Tridentから1つ以上のストレージクラスを取得します。
`volume`：Tridentから1つ以上のボリュームを取得します。

フラグ

`-h、 --help`：ボリュームのヘルプ。
`--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
`--subordinateOf string`:クエリをボリュームの下位に制限します。

イメージ

フラグを使用して `images`、Tridentが必要とするコンテナイメージのテーブルを印刷します。

```
tridentctl images [flags]
```

フラグ

`-h、 --help`:画像のヘルプ。
`-v、 --k8s-version string`: Kubernetesクラスタのセマンティックバージョン。

ボリュームをインポートします

コマンドを使用し `import volume`て、既存のボリュームをTridentにインポートします。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

エイリアス

`volume、 v`

フラグ

`-f、 --filename string`: YAMLまたはJSON PVCファイルへのパス。
`-h、 --help`：ボリュームのヘルプ。
`--no-manage`：PV/PVCのみを作成します。ボリュームのライフサイクル管理を想定しないでください。

をインストールします

フラグを使用し `install`てTridentをインストールします。

```
tridentctl install [flags]
```

フラグ

`--autosupport-image string`：AutoSupportテレメトリのコンテナイメージ（デフォルトは「NetApp / Trident AutoSupport：<current-version>」）。
`--autosupport-proxy string`: AutoSupportテレメトリを送信するためのプロキシのアドレス/ポート。
`--enable-node-prep`：必要なパッケージをノードにインストールしようとします。
`--generate-custom-yaml`:何もインストールせずにYAMLファイルを生成します。
`-h、 --help`：インストールのヘルプ。
`--http-request-timeout`：TridentコントローラのREST APIのHTTP要求タイムアウトを上書きします（デフォルトは1m30秒）。
`--image-registry string`:内部イメージレジストリのアドレス/ポート。
`--k8s-timeout duration`：すべてのKubernetes処理のタイムアウト（デフォルトは3m0）。
`--kubelet-dir string`: kubeletの内部状態のホストの場所(デフォルトは/var/lib/kubelet)。
`--log-format string`：Tridentログ形式（text、json）（デフォルトは「text」）。
`--node-prep`：指定したデータストレージプロトコルを使用してボリュームを管理できるように、TridentでKubernetesクラスタのノードを準備できるようにします。現在 **`iscsi`**`サポートされている値は、のみです。
`--pv string：Tridentが使用するレガシーPVの名前。これが存在しないことを確認します（デフォルトは「Trident」）。
`--pvc string`: Tridentが使用するレガシーPVCの名前。これが存在しないことを確認します（デフォルトは「Trident」）。
`--silence-autosupport`: AutoSupportバンドルをNetAppに自動的に送信しないでください(デフォルトはTRUE)。
`--silent`:インストール中にMOST出力を無効にします。
`--trident-image string`:インストールするTridentイメージ。
`--use-custom-yaml`:セットアップディレクトリに存在する既存のYAMLファイルを使用します。
`--use-ipv6`：Tridentの通信にIPv6を使用します。

ログ

フラグを使用して `logs`Tridentからログを出力します。

```
tridentctl logs [flags]
```

フラグ

`-a、 --archive`：特に指定がないかぎり、すべてのログを含むサポートアーカイブを作成します。
`-h、 --help`：ログのヘルプ。
`-l、 --log string`：表示するTridentログ。Trident | auto | Trident - operator | allのいずれか（デフォルトは「auto」）。
`--node string`：ノードポッドログの収集元となるKubernetesノード名。
`-p,: --previous`以前のコンテナインスタンスが存在する場合は、そのインスタンスのログを取得します。
`--sidecars`:サイドカーコンテナのログを取得します。

送信

Tridentからリソースを送信するには、コマンドを使用し `send`ます。

```
tridentctl send [option]
```

オプション（ **Options** ）

`autosupport`：ネットアップにAutoSupport アーカイブを送信します。

をアンインストールします

フラグを使用して `uninstall`Tridentをアンインストールします。

```
tridentctl uninstall [flags]
```

フラグ

`-h, --help`:アンインストールのヘルプ。
`--silent`:アンインストール中のほとんどの出力を無効にします。

更新

Tridentのリソースを変更するには、コマンドを使用し `update`ます。

```
tridentctl update [option]
```

オプション（ **Options** ）

`backend`: Tridentのバックエンドを更新します。

バックエンドの状態を更新

を使用します `update backend state` バックエンド処理を一時停止または再開するコマンド。

```
tridentctl update backend state <backend-name> [flag]
```

考慮すべきポイント

- TridentBackendConfig（tbc）を使用してバックエンドを作成した場合、ファイルを使用してバックエンドを更新することはできません `backend.json` 。
- がtbcに設定されている場合 `userState` は、コマンドを使用して変更することはできません `tridentctl update backend state <backend-name> --user-state suspended/normal` 。
- tbcで設定した後にvia tridentctlを設定できるようにするには `userState`、 `userState`tbcからフィールドを削除する必要があります。これは、コマンドを使用して実行でき `kubectl edit tbc` ます。フィールドを削除したら `userState`、コマンドを使用してバックエンドのを変更 `userState`できます `tridentctl update backend state`。
- を使用して `tridentctl update backend state` を変更し `userState`ます。またはファイルを使用して更新することもでき `userState` `TridentBackendConfig` `backend.json` ます。これにより、バックエンドの完全な再初期化がトリガーされ、時間がかかる場合があります。

  フラグ

  `-h、 --help`:バックエンド状態のヘルプ。
  `--user-state`:に設定 `suspended` バックエンド処理を一時停止します。をに設定します `normal` バックエンド処理を再開します。に設定すると `suspended`：

- `AddVolume` `Import Volume` 一時停止しています。
- `CloneVolume`、 `ResizeVolume`、 `PublishVolume`、 `UnPublishVolume`、、 `CreateSnapshot`

GetSnapshot RestoreSnapshot、、 DeleteSnapshot、 RemoveVolume、、
GetVolumeExternal ReconcileNodeAccess 引き続き使用できます。

バックエンド構成ファイルまたはのフィールドを使用して、バックエンドの状態を更新することもできます userState TridentBackendConfig backend.json。詳細については、およびを参照して "バックエンドを管理するためのオプション" "kubectl を使用してバックエンド管理を実行します"ください。

- 例： *

JSON

ファイルを使用してを更新するには、次の手順を実行し `userState` `backend.json` ます。

1. ファイルを編集して `backend.json` 、値が「中断」に設定されたフィールドを含め `userState` ます。
2. コマンドと更新されたファイルへのパスを使用して、バックエンドを更新し `tridentctl backend update` `backend.json` ます。

   例： `tridentctl backend update -f /<path to backend JSON file>/backend.json`

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "<redacted>",
    "svm": "nas-svm",
    "backendName": "customBackend",
    "username": "<redacted>",
    "password": "<redacted>",
    "userState": "suspended",
}
```

YAML

tbcが適用されたら、コマンドを使用して編集できます `kubectl edit <tbc-name> -n <namespace>` 。次に、オプションを使用してバックエンド状態をsuspendに更新する例を示し `userState: suspended` ます。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

バージョン

使用 `version` のバージョンを印刷するためのフラグ `tridentctl` 実行中のTridentサービス

```
tridentctl version [flags]
```

フラグ

`--client`:クライアントバージョンのみ(サーバは不要)。
`-h, --help`:バージョンのヘルプ。

プラグインのサポート

Tridentctlはkubectlに似たプラグインをサポートしています。Tridentctlは、プラグインバイナリファイル名が"tridentctl -<plugin>"というスキームに沿っている場合にプラグインを検出し、そのバイナリがPATH環境変数のリストにあるフォルダにあることを示します。検出されたすべてのプラグインは、tridentctlヘルプのpluginセクションに表示されます。オプションで、環境変数TRIDENTCTL_PLUGIN_PATHにプラグインフォルダを指定して検索を制限することもできます(例: `TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`)。変数が使用されている場合、tridenctlは指定されたフォルダのみを検索します。

Tridentの監視

Tridentには、Tridentのパフォーマンスの監視に使用できるPrometheus指標エンドポイントのセットが用意されています。

概要

Tridentの指標を使用すると、次のことを実行できます。

- Tridentの健全性と設定を常に確認しておきます。処理が成功した方法と、想定どおりにバックエンドと通信できるかどうかを調べることができます。
- バックエンドの使用状況の情報を調べて、バックエンドでプロビジョニングされているボリュームの数や消費されているスペースなどを確認します。
- 利用可能なバックエンドにプロビジョニングされたボリュームの量のマッピングを維持します。
- パフォーマンスを追跡する。Tridentがバックエンドと通信して処理を実行するのにかかる時間を確認できます。

デフォルトでは 'Trident のメトリックは '/metrics エンドポイントのターゲットポート 8001' に公開されていますこれらの指標は、 Trident のインストール時にデフォルトで * 有効になります。

必要なもの

- TridentがインストールされたKubernetesクラスタ。
- Prometheus インスタンス。これは a である場合もある "コンテナ化された Prometheus 環境" または、Prometheus をとして実行することもできます "ネイティブアプリケーション"。

手順 1 ： Prometheus ターゲットを定義する

Prometheusターゲットを定義して、指標を収集し、Tridentが管理するバックエンドや作成するボリュームなどに関する情報を取得する必要があります。ここで "ブログ"は、PrometheusとGrafanaをTridentで使用して指標を取得する方法について説明します。このブログでは、KubernetesクラスタでPrometheusをオペレータとして実行する方法と、Trident指標を取得するためのServiceMonitorの作成について説明しています。

手順 2 ： Prometheus ServiceMonitor を作成します

Trident のメトリックを使用するには、「 trident-csi 」サービスを監視し、「 m etrics 」ポートを監視する Prometheus ServiceMonitor を作成する必要があります。ServiceMonitor のサンプルは次のようになります。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
      release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
    - trident
  endpoints:
  - port: metrics
    interval: 15s
```

このServiceMonitor定義は、サービスから返されたメトリックを取得し `trident-csi`、特にサービスのエンドポイントを検索し `metrics`ます。その結果、PrometheusはTridentの指標を認識するように設定されました。

Kubeletは、Tridentから直接利用できるメトリクスに加えて、独自のメトリクスエンドポイントを介して多くのメトリクスを公開して `kubelet_volume_*`います。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "こちらをご覧ください"。

ステップ 3 ： PrompQL を使用して Trident 指標を照会する

PrompQL は、時系列データまたは表データを返す式を作成するのに適しています。

次に、 PrompQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Trident**からの**HTTP 2XX**応答の割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- ステータスコードによる**Trident**からの**REST**応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Trident**によって実行された操作の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Tridentの使用状況情報の取得

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Trident AutoSupportテレメトリの詳細

デフォルトでは、TridentはPrometheus指標と基本的なバックエンド情報を1日おきにNetAppに送信します。

- TridentからNetAppへのPrometheus指標と基本的なバックエンド情報の送信を停止するには、Tridentのインストール時にフラグを渡し `--silence-autosupport`ます。
- Tridentは、経由でコンテナログをNetAppサポートにオンデマンドで送信することもできます `tridentctl send autosupport`。ログをアップロードするには、Tridentをトリガーする必要があります。ログを送信する前に、NetAppを承認する必要があります`https://www.netapp.com/company/legal/privacy-policy/`["プライバシーポリシー"^]。
- 指定されていない場合、Tridentは過去24時間のログをフェッチします。
- フラグを使用してログの保持期間を指定できます `--since`。例： `tridentctl send autosupport --since=1h`。この情報は、Tridentと一緒にインストールされたコンテナを介して収集および送信され `trident-autosupport`ます。コンテナイメージはから入手できます "Trident AutoSupport の略"。
- Trident AutoSupport は、個人情報（ PII ）や個人情報を収集または送信しません。Tridentコンテナイメージ自体には適用されないが付属して "EULA" います。データのセキュリティと信頼に対するネットアップの取り組みについて詳しくは、こちらをご覧 "こちらをご覧ください"ください。

Tridentによって送信されるペイロードの例は次のようになります。

```
---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true
```

- AutoSupport メッセージは、ネットアップの AutoSupport エンドポイントに送信されます。プライベートレジストリを使用してコンテナイメージを格納している場合は '--image_registry' フラグを使用できます
- インストール YAML ファイルを生成してプロキシ URL を設定することもできます。これは 'tridentctl install --generate-custom-yaml を使用して YAML ファイルを作成し 'trident-deployment.yaml の trident-autosupport コンテナに --proxy-url 引数を追加することによって実行できます

Trident指標を無効にする

- メトリックがレポートされないようにするには '--generate-custom-yaml' フラグを使用してカスタム YAML を生成し ' これらを編集して 'trident-main' コンテナに対して --metrics フラグが呼び出されないよう

にします

Trident をアンインストールします

Tridentのアンインストールには、Tridentのインストールと同じ方法を使用する必要があります。

このタスクについて

- アップグレード、依存関係の問題、またはアップグレードの失敗または不完全な実行後に見つかったバグの修正が必要な場合は、Tridentをアンインストールし、該当する手順に従って以前のバージョンを再インストールする必要があります"バージョン"。これは、以前のバージョンに_downgrade_toを実行するための唯一の推奨方法です。
- アップグレードと再インストールを簡単に行うために、Tridentをアンインストールしても、Tridentによって作成されたCRDや関連オブジェクトは削除されません。Tridentとそのすべてのデータを完全に削除する必要がある場合は、を参照してください"TridentとCRDを完全に取り外します。"。

作業を開始する前に

Kubernetesクラスタの運用を停止する場合は、をアンインストールする前に、Tridentで作成されたボリュームを使用するすべてのアプリケーションを削除する必要があります。これにより、PVCが削除される前にKubernetesノードで非公開になります。

元のインストール方法を決定する

Tridentのアンインストールには、インストール時と同じ方法を使用する必要があります。アンインストールする前に、Tridentの最初のインストールに使用したバージョンを確認してください。

1. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してTridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Tridentは手動またはHelmを使用してTridentオペレータを使用してインストールされています。
2. オペレータポッドがある場合は、を使用して、 `kubectl describe tproc trident`TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合、TridentはHelmを使用してインストールされています。
   - Helmラベルがない場合、TridentはTridentオペレータを使用して手動でインストールされています。

Tridentオペレータのインストールをアンインストールする

Tridentオペレータのインストールは手動でアンインストールすることも、Helmを使用してアンインストールすることもできます。

手動インストールのアンインストール

オペレータを使用してTridentをインストールした場合は、次のいずれかの方法でアンインストールできます。

1. 編集 **`TridentOrchestrator` CR**を実行し、アンインストールフラグを設定します：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"uninstall":true}}'
```

をクリックします `uninstall` フラグはに設定されています `true`は、TridentオペレータがTridentをアンインストールしますが、TridentOrchestrator自体は削除されません。Trident を再度インストールする場合は、 TridentOrchestrator をクリーンアップして新しい Trident を作成する必要があります。

2. 削除 **`TridentOrchestrator`**：Tridentの展開に使用したCRを削除する `TridentOrchestrator` と、`Trident`をアンインストールするようにオペレータに指示します。オペレータがの削除を処理し `TridentOrchestrator`、インストール時に作成したTridentポッドを削除して、Tridentデプロイメントとデーモンセットの削除に進みます。

```
kubectl delete -f deploy/<bundle.yaml> -n <namespace>
```

Helmインストールのアンインストール

Helmを使用してTridentをインストールした場合は、を使用してアンインストールできます `helm uninstall`。

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME           NAMESPACE       REVISION        UPDATED
STATUS           CHART                           APP VERSION
trident        trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

のアンインストール `tridentctl` インストール

Tridentに関連付けられているすべてのリソース（CRDおよび関連オブジェクトを除く）を削除するには、のコマンドを `tridentctl`使用し `uninstall`ます。

```
./tridentctl uninstall -n <namespace>
```

Trident for Docker

導入の前提条件

Tridentを導入する前に、必要なプロトコルの前提条件をホストにインストールして設定する必要があります。

要件を確認します

- の導入がすべてを満たしていることを確認します "要件"。
- サポートされているバージョンの Docker がインストールされていることを確認します。Docker のバージョンが最新でない場合は、 "インストールまたは更新します"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールおよび設定されていることを確認します。

NFSツール

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

RHEL 8以降

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

Ubuntu

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

NFSツールをインストールしたあとにワーカーノードをリブートして、コンテナにボリュームを接続する際の障害を回避します。

iSCSIツール

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

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 sg3_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：

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

4. マルチパスを有効化：

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

「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」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. 'iSCSI' を有効にして開始します

```
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 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：

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

4. マルチパスを有効化：

```
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「 defVaults 」に含まれていることを確認します。

5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されてい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
```

NVMeツール

オペレーティングシステムに対応した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や、使用しているカーネルバージョンに対応するNVMeパッケージがない場合は、ノードのカーネルバージョンをNVMeパッケージで更新しなければならないことがあります。

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
```

Tridentの導入

Trident for Dockerは、NetAppストレージプラットフォームのDockerエコシステムと直接統合できます。ストレージプラットフォームから Docker ホストまで、ストレージリソースのプロビジョニングと管理をサポートします。また、将来プラットフォームを追加するためのフレームワークもサポートします。

同じホスト上でTridentの複数のインスタンスを同時に実行できます。これにより、複数のストレージシステムとストレージタイプへの同時接続が可能になり、 Docker ボリュームに使用するストレージをカスタマイズできます。

必要なもの

を参照してください"導入の前提条件"。前提条件を満たしていることを確認したら、Tridentを導入する準備が整います。

Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降）

作業を開始する前に

従来のデーモンメソッドでDocker 1.13/17.03より前のTridentを使用している場合は、マネージプラグインメソッドを使用する前に、Tridentプロセスを停止してDockerデーモンを再起動してください。

1. 実行中のインスタンスをすべて停止します。

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. Docker を再起動します。

```
systemctl restart docker
```

3. Docker Engine 17.03 （新しい 1.13 ）以降がインストールされていることを確認します。

```
docker --version
```

バージョンが最新でない場合は、 "インストール環境をインストールまたは更新します"。

手順

1. 構成ファイルを作成し、次のオプションを指定します。
   - `config`: デフォルトのファイル名は 'config.json ですが ' ファイル名に config オプションを指定することで ' 選択した任意の名前を使用できます構成ファイルは ' ホスト・システムの /etc/netappdvp ディレクトリに格納されている必要があります
   - 「 log-level 」：ログレベルを指定します（「 debug` 」、「 info 」、「 warn 」、「 error 」、「 fatal 」）。デフォルトは「 info 」です。
   - `ebug`: デバッグログを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。
     i. 構成ファイルの場所を作成します。

```
sudo mkdir -p /etc/netappdvp
```

     ii. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/config.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
EOF
```

2. 管理プラグインシステムを使用してTridentを起動します。を、使用しているプラグインのバージョン（xxx.xx.x）に置き換えます `<version>`。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. Tridentを使用して、構成済みシステムのストレージを消費します。

a. 「 firstVolume 」という名前のボリュームを作成します。

```
docker volume create -d netapp --name firstVolume
```

b. コンテナの開始時にデフォルトのボリュームを作成します。

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

c. ボリューム「 firstVolume 」を削除します。

```
docker volume rm firstVolume
```

従来の方法（バージョン **1.12** 以前）

作業を開始する前に

1. バージョン 1.10 以降の Docker がインストールされていることを確認します。

```
docker --version
```

使用しているバージョンが最新でない場合は、インストールを更新します。

```
curl -fsSL https://get.docker.com/ | sh
```

または "ご使用のディストリビューションの指示に従ってください"。

2. NFS または iSCSI がシステムに対して設定されていることを確認します。

手順

1. NetApp Docker Volume Plugin をインストールして設定します。

a. アプリケーションをダウンロードして開梱します。

```
wget
https://github.com/NetApp/trident/releases/download/v24.10.0/trident-
installer-24.10.0.tar.gz
tar zxf trident-installer-24.10.0.tar.gz
```

b. ビンパス内の場所に移動します。

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/
sudo chown root:root /usr/local/bin/trident
sudo chmod 755 /usr/local/bin/trident
```

c. 構成ファイルの場所を作成します。

```
sudo mkdir -p /etc/netappdvp
```

d. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/ontap-nas.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1"
}
EOF
```

2. バイナリを配置して構成ファイルを作成したら、目的の構成ファイルを使用してTridentデーモンを起動します。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

指定されていない場合、ボリュームドライバのデフォルト名は「NetApp」です。

デーモンが開始されたら、 Docker CLI インターフェイスを使用してボリュームを作成および管理できます

3. ボリュームを作成します

```
docker volume create -d netapp --name trident_1
```

4. コンテナの開始時に Docker ボリュームをプロビジョニング：

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
alpine ash
```

5. Docker ボリュームを削除します。

```
docker volume rm trident_1
docker volume rm trident_2
```

システム起動時にTridentを起動する

システムベースのシステムのサンプルユニットファイルは、から入手できます `contrib/trident.service.example` Gitリポジトリで実行します。RHELでファイルを使用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

   複数のインスタンスを実行している場合は、ユニットファイルに一意の名前を使用してください。

```
cp contrib/trident.service.example
/usr/lib/systemd/system/trident.service
```

2. ファイルを編集し、概要（ 2 行目）を変更してドライバ名と構成ファイルのパス（ 9 行目）を環境に合わせます。
3. 変更を取り込むためにシステムをリロードします。

```
systemctl daemon-reload
```

4. サービスを有効にします。

   この名前は '/usr/lib/systemd/system' ディレクトリ内のファイルの名前によって異なります

```
systemctl enable trident
```

5. サービスを開始します。

```
systemctl start trident
```

6. ステータスを確認します。

```
systemctl status trident
```

ユニット・ファイルを変更するときは ' 変更を認識するために 'systemctl daemon-reload コマンドを実行します

Trident をアップグレードまたはアンインストールする

使用中のボリュームに影響を与えることなく、Trident for Dockerを安全にアップグレードできます。アップグレードプロセスでは、 `docker volume`プラグインへのコマンドが正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

Trident for Dockerをアップグレードするには、次の手順を実行します。

手順

1. 既存のボリュームを表示します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

2. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインをアップグレードします。

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```

Tridentの18.01リリースは、nDVPに代わるものです。イメージからイメージに `netapp/trident-plugin`直接アップグレードする必要があり `netapp/ndvp-plugin`ます。

4. プラグインを有効にします。

```
docker plugin enable netapp:latest
```

5. プラグインが有効になっていることを確認します。

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

古いバージョンのTrident（20.10より前のバージョン）からTrident 20.10以降にアップグレードすると、エラーが発生することがあります。詳細については、を参照してください "既知の問題"。エラーが発生した場合は、まずプラグインを無効にしてからプラグインを削除し、追加のconfigパラメータを渡して必要なTridentバージョンをインストールする必要があります。`docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

をアンインストールします

Trident for Dockerをアンインストールするには、次の手順を実行します。

手順

1. プラグインで作成されたボリュームをすべて削除します。
2. プラグインを無効にします。

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインを削除します。

```
docker plugin rm netapp:latest
```

ボリュームを操作します

必要に応じてTridentドライバ名を指定した標準コマンドを使用すると、ボリュームの作成、クローニング、および削除を簡単に実行でき `docker volume`ます。

ボリュームを作成します

- デフォルトの名前を使用して、ドライバでボリュームを作成します。

```
docker volume create -d netapp --name firstVolume
```

- 特定のTridentインスタンスを使用してボリュームを作成します。

```
docker volume create -d ntap_bronze --name bronzeVolume
```

何も指定しない場合 "オプション（ Options ）"、ドライバのデフォルトが使用されます。

- デフォルトのボリュームサイズを上書きします。次の例を参照して、ドライバで 20GiB ボリュームを作成してください。

```
docker volume create -d netapp --name my_vol --opt size=20G
```

ボリュームサイズは、オプションの単位（ 10G 、 20GB 、 3TiB など）を含む整数値で指定します。単位を指定しない場合、デフォルトは g ですサイズの単位は、 2 の累乗（ B 、 KiB 、 MiB 、 GiB 、 TiB ）または 10 の累乗（ B 、 KB 、 MB 、 GB 、 TB ）のいずれかです。略記単位では、 2 の累乗が使用されます（ G=GiB 、 T=TiB 、…）。

ボリュームを削除します

- 他の Docker ボリュームと同様にボリュームを削除します。

```
docker volume rm firstVolume
```

「 olidfire -san 」ドライバを使用する場合、上記の例ではボリュームを削除してパージします。

Trident for Dockerをアップグレードするには、次の手順を実行します。

ボリュームのクローンを作成します

、 `ontap-san` `solidfire-san`、、およびを `gcp-cvs` `storage drivers`使用する場合 `ontap-nas` 、Tridentはボリュームをクローニングできます。ドライバまたは `ontap-nas-economy`ドライバを使用している場合、 `ontap-nas-flexgroup`クローニングはサポートされません。既存のボリュームから新しいボリュームを作成すると、新しい Snapshot が作成されます。

- ボリュームを調べて Snapshot を列挙します。

```
docker volume inspect <volume_name>
```

- 既存のボリュームから新しいボリュームを作成します。その結果、新しい Snapshot が作成されます。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume>
```

- ボリューム上の既存の Snapshot から新しいボリュームを作成します。新しい Snapshot は作成されません。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

例

```
docker volume inspect firstVolume

[
    {
        "Driver": "ontap-nas",
        "Labels": null,
        "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
        "Name": "firstVolume",
        "Options": {},
        "Scope": "global",
        "Status": {
            "Snapshots": [
                {
                    "Created": "2017-02-10T19:05:00Z",
                    "Name": "hourly.2017-02-10_1505"
                }
            ]
        }
    }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

外部で作成したブロックデバイス（またはそのクローン）には、パーティションがなく、ファイルシステムがTridentでサポートされている場合（例：-formatted `/dev/sdc1`はTrident経由でアクセスできません）にのみ、Trident *を使用してコンテナからアクセスできます `ext4。

ドライバ固有のボリュームオプション

ストレージドライバにはそれぞれ異なるオプションがあり、ボリュームの作成時に指定することで結果をカスタマイズできます。構成済みのストレージシステムに適用されるオプションについては、以下を参照してください。

ボリューム作成処理では、これらのオプションを簡単に使用できます。CLI の操作中に '-o' 演算子を使用して

' オプションと値を指定しますこれらは、 JSON 構成ファイルの同等の値よりも優先されます。

ONTAP ボリュームのオプション

NFS と iSCSI のどちらの場合も、 volume create オプションには次のオプションがあります。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 1GiB です。
「平和のための準備」を参照してください	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値は 'none(シン・プロビジョニング) と 'volume(シック・プロビジョニング) です
「ナプショットポリシー」	Snapshot ポリシーが目的の値に設定されます。デフォルトは「 none 」です。つまり、ボリュームに対してスナップショットが自動的に作成されることはありません。ストレージ管理者によって変更されていない限り、「 default 」という名前のポリシーがすべての ONTAP システムに存在し、 6 個の時間単位 Snapshot 、 2 個の日単位 Snapshot 、および 2 個の週単位 Snapshot を作成して保持します。スナップショットに保存されているデータは ' ボリューム内の任意のディレクトリ内の .snapshot ディレクトリに移動してリカバリできます
「スナップショット予約」	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5% ）。 Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、 ONTAP-NAS-エコノミー を除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
'plitOnClone	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトはです `false`。クローンボリュームのクローニングは、作成直後に親からクローンをスプリットする方法を推奨します。これは、ストレージ効率化の効果がまったくないためです。たとえば、空のデータベースをクローニングしても時間は大幅に短縮されますが、ストレージはほとんど削減されません。そのため、クローンはすぐにスプリットすることを推奨します。

オプション	説明
「暗号化」	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、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 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すべてのボリュームでが有効になります。 詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。
階層ポリシー	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、 NFS * のみ * 用の追加オプションです。

オプション	説明
「 unixPermissions 」	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは ' アクセス権は '--rwxr-xr-x' または数値表記 0755 に設定され 'root' は所有者になりますテキスト形式または数値形式のどちらかを使用できます。
「スナップショット方向」	これをに設定します `true` がを作成します `.snapshot` ボリュームにアクセスしているクライアントから認識できるディレクトリ。デフォルト値はです `false`の可視性を意味します `.snapshot` ディレクトリはデフォルトで無効になっています。一部のイメージ（公式のMySQLイメージなど）が、 `.snapshot` ディレクトリが表示されます。
「 exportPolicy 」と入力します	ボリュームで使用するエクスポートポリシーを設定します。デフォルトは「デフォルト」です。
'ecurityStyle'	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトは「 unix 」です。有効な値は「 unix 」と「 immimixed 」です。

以下の追加オプションは、 iSCSI * のみ * 用です。

オプション	説明
「 filesystemtype 」です	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトは「 ext4 」です。有効な値は「 ext3 」、「 ext4 」、「 xfs 」です。
「平和の配分」	これをに設定します `false` LUNのスペース割り当て機能を無効にします。デフォルト値はです `true`つまり、ボリュームのスペースが不足し、ボリューム内のLUNに書き込みを受け付けられなくなったときに、ONTAP からホストに通知されます。また、このオプションで ONTAP 、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 10GiB ボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- Snapshot を使用して 100GiB のボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- setuid ビットが有効になっているボリュームを作成します。

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

最小ボリュームサイズは 20MiB です。

スナップショット予約が指定されておらず、スナップショットポリシーがの場合、 `none`Tridentは0%のスナップショット予約を使用します。

- Snapshot ポリシーがなく、 Snapshot リザーブがないボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- Snapshot ポリシーがなく、カスタムの Snapshot リザーブが 10% のボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- Snapshot ポリシーを使用し、カスタムの Snapshot リザーブを 10% に設定してボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- Snapshot ポリシーを設定してボリュームを作成し、 ONTAP のデフォルトの Snapshot リザーブ（通常は 5% ）を受け入れます。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element ソフトウェアのボリュームオプション

Element ソフトウェアのオプションでは、ボリュームに関連付けられているサービス品質（ QoS ）ポリシーのサイズと QoS を指定できます。ボリュームが作成されると '-o type=service_level' という命名法を使用して ' ボリュームに関連付けられた QoS ポリシーが指定されます

Element ドライバを使用して QoS サービスレベルを定義する最初の手順は、少なくとも 1 つのタイプを作成し、構成ファイル内の名前に関連付けられた最小 IOPS 、最大 IOPS 、バースト IOPS を指定することです。

Element ソフトウェアのその他のボリューム作成オプションは次のとおりです。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルト値は 1GiB または設定エントリ … 「 defaults 」： 「 size 」：「 5G 」 } 。
「ブロックサイズ」	512 または 4096 のいずれかを使用します。デフォルトは 512 または config エントリ DefaultBlockSize です。

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

```
{
    "...": "..."
    "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
            }
        }
    ]
}
```

上記の構成では、 Bronze 、 Silver 、 Gold の 3 つのポリシー定義を使用します。これらの名前は任意です。

- 10GiB の Gold ボリュームを作成します。

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 100GiB Bronze ボリュームを作成します。

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

ログを収集します

トラブルシューティングに役立つログを収集できます。ログの収集方法は、 Docker プラグインの実行方法によって異なります。

トラブルシューティング用にログを収集する

手順

1. 推奨される管理プラグイン方式を使用して（コマンドを使用して）Tridentを実行している場合は `docker plugin`、次のように表示します。

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
journalctl -u docker | grep 4fb97d2b956b
```

   標準的なロギングレベルでは、ほとんどの問題を診断できます。十分でない場合は、デバッグロギングを有効にすることができます。

2. デバッグロギングをイネーブルにするには、デバッグロギングをイネーブルにしてプラグインをインストールします。

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

   または、プラグインがすでにインストールされている場合にデバッグログを有効にします。

```
docker plugin disable <plugin>
docker plugin set <plugin> debug=true
docker plugin enable <plugin>
```

3. ホストでバイナリ自体を実行している場合、ログはホストの `/var/log/netappdvp` ディレクトリ。デバッグロギングを有効にするには、を指定します `-debug` プラグインを実行すると、

一般的なトラブルシューティングのヒント

- 新しいユーザーが実行する最も一般的な問題は、プラグインの初期化を妨げる構成ミスです。この場合、プラグインをインストールまたは有効にしようとすると、次のようなメッセージが表示されることがあります。

  「デーモンからのエラー応答 : ダイヤル UNIM/run/docx/plugins/<id>/NetApp/smock: connect: no such file or directory`

これは、プラグインの起動に失敗したことを意味します。幸い、このプラグインには、発生する可能性の高い問題のほとんどを診断するのに役立つ包括的なログ機能が組み込まれています。

- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl ステータス rpcbind またはそれに相当する処理を実行することで確認できます

複数のTridentインスタンスの管理

複数のストレージ構成を同時に使用する必要がある場合は、 Trident の複数のインスタンスが必要です。複数のインスタンスの鍵は、コンテナ化されたプラグインでは「 --alias 」オプション、ホストで Trident をインスタンス化する場合は「 --volume-driver 」オプションを使用して、それぞれ異なる名前を指定することです。

Docker Managed Plugin （バージョン 1.13 / 17.03 以降）の手順

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. 別のエイリアスと構成ファイルを指定して、 2 番目のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

   たとえば、 gold ボリュームの場合：

```
docker volume create -d gold --name ntapGold
```

   たとえば、 Silver ボリュームの場合：

```
docker volume create -d silver --name ntapSilver
```

従来の（バージョン 1.12 以前）の場合の手順

1. カスタムドライバ ID を使用して NFS 設定でプラグインを起動します。

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. カスタムドライバ ID を使用して、 iSCSI 構成でプラグインを起動します。

```
sudo trident --volume-driver=netapp-san --config=/path/to/config
-iscsi.json
```

3. ドライバインスタンスごとに Docker ボリュームをプロビジョニングします。

   たとえば、 NFS の場合：

```
docker volume create -d netapp-nas --name my_nfs_vol
```

   たとえば、 iSCSI の場合：

```
docker volume create -d netapp-san --name my_iscsi_vol
```

ストレージ構成オプション

Trident構成で使用可能な設定オプションを参照してください。

グローバル構成オプション

これらの設定オプションは、使用するストレージプラットフォームに関係なく、すべてのTrident構成に適用されます。

オプション	説明	例
「バージョン」	構成ファイルのバージョン番号	`1`
'torageDriverName'	ストレージドライバの名前	`ontap-nas`、 `ontap-san`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `solidfire-san`
'toragePrefix'	ボリューム名のオプションのプレフィックス。デフォルト： `netappdvp_`。	`staging_`

オプション	説明	例
「 limitVolumeSize 」と入力します	ボリュームサイズに関するオプションの制限。デフォルト：""（強制なし）	`10g`

Element バックエンドには 'toragePrefix' （デフォルトを含む）を使用しないでくださいデフォルトでは 'solidfire-san' ドライバはこの設定を無視し ' 接頭辞を使用しませんDocker ボリュームマッピングには特定の tenantID を使用するか、 Docker バージョン、ドライバ情報、名前の munging が使用されている可能性がある場合には Docker から取得した属性データを使用することを推奨します。

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。「 size 」オプションは、すべてのコントローラタイプで使用できます。デフォルトのボリュームサイズの設定方法の例については、 ONTAP の設定に関するセクションを参照してください。

オプション	説明	例
「 size 」	新しいボリュームのオプションのデフォルトサイズ。デフォルト：`1G`	`10G`

ONTAP の設定

ONTAP を使用する場合は、上記のグローバル構成値に加えて、次のトップレベルオプションを使用できます。

オプション	説明	例
「管理 LIF 」	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name （ FQDN ；完全修飾ドメイン名）を指定できます。	`10.0.0.1`

オプション	説明	例
「重複排除	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*：を指定することをお勧めします `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。 • ONTAP SANドライバ*: iSCSIには指定しないでくださいTridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCI LIFを検出します。が明示的に定義されている場合は、警告が生成され`dataLIF`ます。	`10.0.0.2`
'VM'	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	`svm_nfs`
「ユーザ名」	ストレージデバイスに接続するユーザ名	`vsadmin`
「 password 」と入力します	ストレージ・デバイスに接続するためのパスワード	`secret`
「集約」	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）。ドライバの場合 `ontap-nas-flexgroup`、このオプションは無視されます。SVMに割り当てられたすべてのアグリゲートを使用して、FlexGroupボリュームがプロビジョニングされます。	`aggr1`
「 AggreglimitateUsage 」と入力します	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	`75%`

オプション	説明	例
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。デフォルトは「 -o nfsvers=3 」です。* 「 ONTAP-NAS' 」および「 ONTAP-NAS-エコノミー 」ドライバ専用です。 "ここでは、 NFS ホストの設定情報を参照してください"。	`-o nfsvers=4`
「 igroupName 」と入力します	Tridentでは、ノードごとにASを`netappdvp`作成および管理し`igroups`ます。 この値は変更したり省略したりすることはできません。 でのみ使用できます **`ontap-san`** ドライバ。	`netappdvp`
「 limitVolumeSize 」と入力します	要求可能な最大ボリュームサイズ。	`300g`
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数は [50 、 300] の範囲で指定する必要があります。デフォルトは 200 です。 *のため `ontap-nas-economy` ドライバ。このオプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	`300`
`sanType`	ドライバでのみサポートされてい **`ontap-san`**`ます。` ***iSCSI**、 **`nvme`NVMe/TCP**、または **`fcp`SCSI over Fibre Channel（FC；SCSI over Fibre Channel）**に対してを選択します **`iscsi`**。「FCP」（SCSI over FC）は、Trident 24.10リリースの技術プレビュー機能です。*	`iscsi` 空白の場合
`limitVolumePoolSize`	* `ontap-san-economy`および`ontap-san-economy`ドライバでのみサポートされています。*ONTAP ONTAP NASエコノミードライバおよびONTAP SANエコノミードライバでFlexVolサイズを制限します。	`300g`

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。

オプション	説明	例
「平和のための準備」を参照してください	スペースリザベーションモード `none` （シンプロビジョニング）または `volume` （シック）	「 NONE 」
「ナプショットポリシー」	使用するSnapshotポリシー。デフォルトはです `none`	「 NONE 」
「スナップショット予約」	Snapshotリザーブの割合。デフォルトはONTAP のデフォルトをそのまま使用する場合はです	`10`
'plitOnClone	作成時に親からクローンをスプリットします。デフォルトはです `false`	「偽」
「暗号化」	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、クラスタで NVE のライセンスが設定され、有効になっている必要があります。 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すべてのボリュームでが有効になります。 詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	正しいです
「 unixPermissions 」	プロビジョニングされたNFSボリュームのNASオプション。デフォルトはです `777`	`777`
「スナップショット方向」	ディレクトリにアクセスするためのNASオプション `.snapshot`。	NFSv4の場合は「true」NFSv3の場合は「false」
「 exportPolicy 」と入力します	NFSエクスポートポリシーで使用するNASオプション。デフォルトはです `default`	`default`
'ecurityStyle'	プロビジョニングされたNFSボリュームにアクセスするためのNASオプション。 NFSのサポート `mixed` および `unix` セキュリティ形式デフォルトはです `unix`。	`unix`
「 filesystemtype 」です	ファイルシステムタイプを選択するためのSANオプション。デフォルトはです `ext4`	`xfs`
階層ポリシー	使用する階層化ポリシー。デフォルトはです `none`; `snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合	「 NONE 」

スケーリングオプション

「 ONTAP-NAS' 」ドライバと「 ONTAP-SAN' 」ドライバは、各 Docker ボリューム用の ONTAP FlexVol を作成します。ONTAP では、クラスタノードあたり最大 1 、 000 個の FlexVol がサポートされます。クラスタの最大 FlexVol 数は 12 、 000 です。Docker ボリューム要件がこの制限に適合する場合、「 ONTAP - NAS 」ドライバは FlexVol が提供する Docker ボリューム単位のスナップショットやクローン作成などの追加機能により、 NAS 解決策の方が望ましいとされます。

FlexVol の制限で対応できる容量よりも多くの Docker ボリュームが必要な場合は、「 ONTAP - NAS - エコノミー」または「 ONTAP - SAN - エコノミー」ドライバを選択します。

「 ONTAP - NAS - エコノミー」ドライバは、自動的に管理される FlexVol プール内の ONTAP qtree として Docker ボリュームを作成します。qtree の拡張性は、クラスタノードあたり最大 10 、 000 、クラスタあたり最大 2 、 40 、 000 で、一部の機能を犠牲にすることで大幅に向上しています。「 ONTAP - NAS - エコノミー」ドライバは、 Docker ボリューム単位のスナップショットまたはクローン作成をサポートしていません。

> Swarm は複数のノード間でのボリューム作成のオーケストレーションを行わないため 'ONTAP-NAS-エコノミー のドライバは現在 Docker Swarm ではサポートされていません

「 ONTAP と SAN の経済性」のドライバは、自動的に管理される FlexVol の共有プール内で、 ONTAP LUN として Docker ボリュームを作成します。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、 SAN ワークロードのスケーラビリティが向上します。ストレージアレイに応じて、 ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、 Docker ボリューム単位の Snapshot とクローニングをサポートします。

ドライバを選択する `ontap-nas-flexgroup`と、数十億個のファイルを含むペタバイト規模まで拡張可能な単一ボリュームへの並列処理を強化できます。FlexGroup のユースケースとしては、 AI / ML / DL 、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Tridentでは、FlexGroupボリュームのプロビジョニング時に、SVMに割り当てられているすべてのアグリゲートが使用されます。Trident での FlexGroup のサポートでは、次の点も考慮する必要があります。

- ONTAP バージョン 9.2 以降が必要です。
- 本ドキュメントの執筆時点では、 FlexGroup は NFS v3 のみをサポートしています。
- SVM で 64 ビットの NFSv3 ID を有効にすることを推奨します。
- 推奨されるFlexGroupメンバー/ボリュームの最小サイズは100GiBです。
- FlexGroupボリュームではクローニングはサポートされていません。

FlexGroupとFlexGroupに適したワークロードについては、を参照してください "NetApp FlexGroupボリュームベストプラクティスおよび実装ガイド"。

同じ環境で高度な機能と大規模な拡張性を実現するために 'ONTAP-NAS' を使用して Docker Volume Plugin の複数のインスタンスを実行し ' もう 1 つは「 ONTAP-NAS-エコノミー 」を使用して実行できます

Trident用のカスタム**ONTAP**ロール

Tridentで処理を実行するためにONTAP adminロールを使用する必要がないように、最小Privilegesを持つONTAPクラスタロールを作成できます。Tridentバックエンド構成にユーザ名を含めると、Trident作成したONTAPクラスタロールが使用されて処理が実行されます。

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

ONTAP CLIノシヨウ

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ユーザのユーザ名を作成します。

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

3. ユーザにロールをマッピングします。

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

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

1. カスタムロールの作成：
   a. クラスタレベルでカスタムロールを作成するには、*[クラスタ]>[設定]*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]>[Storage VM]>[設定]>[ユーザとロール]*を選択し `required SVM`ます。

   b. の横にある矢印アイコン（→*）を選択します。
   c. [Roles]*で[+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. ロールを**Trident**ユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。
   a. で**[**アイコンの追加**]**+*を選択します。
   b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。
   c. [保存（ Save ）] 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"
- "ロールとユーザを使用する"

ONTAP 構成ファイルの例

<code>ontap-nas</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "defaults": {
      "size": "10G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code>ontap-nas-flexgroup</code> ドライバの**NFS**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",
    "defaults": {
      "size": "100G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

<code>ontap-nas-economy</code> ドライバの**NFS**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",
    "aggregate": "aggr1"
}
```

<code>ontap-san</code> ドライバの**iSCSI**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code>ontap-san-economy</code> ドライバの**NFS**の例

```
{
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi_eco",
    "username": "vsadmin",
    "password": "password",
    "aggregate": "aggr1",
    "igroupName": "netappdvp"
}
```

<code>ontap-san</code> ドライバの**NVMe/TCP**の例

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

Element ソフトウェアの設定

Element ソフトウェア（ NetApp HCI / SolidFire ）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
「エンドポイント」	\ <a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">https://<login>:<password>@<mvip>/json-rpc/<element-version>;	\ https://admin:admin@192.168.160.3/json-rpc/8.0
「 VIP 」	iSCSI の IP アドレスとポート	10.0.0.7 ： 3260
「 tenantname 」	使用する SolidFire テナント（見つからない場合に作成）	`docker`
「 InitiatorIFCace 」	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	`default`
「タイプ」	QoS の仕様	以下の例を参照してください

オプション	説明	例
「 LegacyNamePrefix 」のように入力します	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンのTridentを使用していて、既存のボリュームでアップグレードを実行した場合は、volume-nameメソッドでマッピングされた古いボリュームにアクセスするためにこの値を設定する必要があります。	`netappdvp-`

「 olidfire -san 」ドライバは Docker Swarm をサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "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
            }
        }
    ]
}
```

既知の問題および制限事項

ここでは、TridentでDockerを使用する場合の既知の問題および制限事項について説明します。

Trident Docker Volume Plugin を旧バージョンから 20.10 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。

回避策

1. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
```

2. プラグインを削除します。

```
docker plugin rm -f netapp:latest
```

3. 追加の 'config' パラメータを指定して ' プラグインを再インストールします

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

ボリューム名は 2 文字以上にする必要があります。

これは Docker クライアントの制限事項です。クライアントは、 1 文字の名前を Windows パスと解釈します。 "バグ 25773 を参照"。

Docker Swarmには特定の動作があり、ストレージとドライバの組み合わせごとにTridentがDocker Swarmをサポートできないようになっています。

- Docker Swarm は現在、ボリューム ID ではなくボリューム名を一意のボリューム識別子として使用します。
- ボリューム要求は、 Swarm クラスタ内の各ノードに同時に送信されます。
- ボリュームプラグイン（Tridentを含む）は、Swarmクラスタ内の各ノードで個別に実行する必要があります。ONTAPの動作方法とドライバと `ontap-san`ドライバの機能により、 `ontap-nas`これらの制限内で動作できるのはこれらのドライバだけです。

その他のドライバには、競合状態などの問題があります。このような問題が発生すると、ボリュームを同じ名前で異なる ID にする機能が Element に備わっているため、「勝者」を明確にせずに 1 回の要求で大量のボリュームを作成できるようになります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。

ベストプラクティスと推奨事項

導入

Tridentを導入する際には、ここに記載されている推奨事項に従ってください。

専用のネームスペースに導入します

"ネームスペース"異なるアプリケーション間で管理を分離し、リソース共有の障壁となります。たとえば、あるネームスペースの PVC を別のネームスペースから使用することはできません。Tridentは、Kubernetesクラスタ内のすべてのネームスペースにPVリソースを提供するため、Privilegesを昇格させたサービスアカウントを利用します。

また、 Trident ポッドにアクセスすると、ユーザがストレージシステムのクレデンシャルやその他の機密情報にアクセスできるようになります。アプリケーションユーザと管理アプリケーションが Trident オブジェクト定義またはポッド自体にアクセスできないようにすることが重要です。

クォータと範囲制限を使用してストレージ消費を制御します

Kubernetes には、 2 つの機能があります。これらの機能を組み合わせることで、アプリケーションによるリソース消費を制限する強力なメカニズムが提供されます。。 "ストレージクォータメカニズム" 管理者は、グローバルおよびストレージクラス固有の、容量とオブジェクト数の使用制限をネームスペース単位で実装できます。さらに、を使用します "範囲制限" 要求がプロビジョニングツールに転送される前に、 PVC 要求が最小値と最大値の両方の範囲内にあることを確認します。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "クォータの活用方法"。

ストレージ構成

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の SVM を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします "SVM-DR" Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください "NetApp Hardware Universe の略" 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、 Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident の「 ONTAP - NAS 」と「 ONTAP - SAN 」ドライバは、作成された Kubernetes 永続ボリューム（ PV ）ごとに FlexVol をプロビジョニングします。「 ONTAP-NAS-エコノミー 」ドライバは、 PVS 200 個につき約 1 つの FlexVol を作成します（ 50 ～ 300 の範囲で構成可能）。「 ONTAP-SAN-エコノミー 」ドライバは、 PVS 100 個につき約 1 つの FlexVol を作成します（ 50 ～ 200 の範囲で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

「 mAX-VOLUMES 」の値は、環境に固有のいくつかの条件によって異なります。

- ONTAP クラスタ内の既存のボリュームの数
- 他のアプリケーション用に Trident 外部でプロビジョニングするボリュームの数
- Kubernetes アプリケーションで消費されると予想される永続ボリュームの数

「 mAX-volumes 」の値は、 ONTAP クラスタ内のすべてのノードでプロビジョニングされたボリュームの合計数であり、個々の ONTAP ノードではプロビジョニングされません。その結果、 ONTAP クラスタノードの

Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、 2 ノードの ONTAP クラスタでは、最大 2 、 000 個の FlexVol をホストできます。最大ボリューム数を 1250 に設定していると、非常に妥当な結果が得られます。ただし、のみの場合 "アグリゲート" あるノードから SVM に割り当てられている場合や、あるノードから割り当てられたアグリゲートをプロビジョニングできない場合（容量など）は、他のノードが Trident でプロビジョニングされたすべてのボリュームのターゲットになります。これは、「 mAX-VOLUMES 」の値に達する前にそのノードのボリューム数の上限に達する可能性があることを意味し、 Trident とそのノードを使用する他のボリューム処理の両方に影響します。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

Trident で作成できるボリュームの最大サイズを制限

Trident で作成できるボリュームの最大サイズを設定するには、「 backend.json 」の定義で「 limitVolumeSize 」パラメータを使用します。

ストレージアレイでボリュームサイズを制御するだけでなく、 Kubernetes の機能も利用する必要があります。

Tridentで作成される**FlexVol**の最大サイズを制限する

ONTAPドライバSAN-EconomyドライバおよびONTAP NAS-Economyドライバのプールとして使用されるFlexVolの最大サイズを設定するには、 `limitVolumePoolSize` `backend.json`定義でパラメータを使用します。

双方向 **CHAP** を使用するように **Trident** を設定します

バックエンド定義で CHAP イニシエータとターゲットのユーザ名とパスワードを指定し、 Trident を使用して SVM で CHAP を有効にすることができます。を使用する `useCHAP` バックエンド構成のパラメータであるTridentは、CHAPを使用してONTAP バックエンドのiSCSI接続を認証します。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、 Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。これはに役立ちます "Bully を防止します" Trident SVM 外のワークロードに影響を及ぼす、制御不能なコンテナ。

SVM の QoS ポリシーはいくつかの手順で作成します。正確な情報については、ご使用の ONTAP バージョンのマニュアルを参照してください。次の例は、 SVM で使用可能な合計 IOPS を 5000 に制限する QoS ポリシーを作成します。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

また、使用しているバージョンの ONTAP でサポートされている場合は、最小 QoS を使用してコンテナ化されたワークロードへのスループットを保証することもできます。アダプティブ QoS は SVM レベルのポリシーには対応していません。

コンテナ化されたワークロード専用の IOPS は、さまざまな要素によって異なります。その中には、次のようなものがあります。

- ストレージアレイを使用するその他のワークロード。Kubernetes 環境とは関係なく、ストレージリソースを利用するほかのワークロードがある場合は、それらのワークロードが誤って影響を受けないように注意する必要があります。
- 想定されるワークロードはコンテナで実行されます。IOPS 要件が高いワークロードをコンテナで実行する場合は、 QoS ポリシーの値が低いとエクスペリエンスが低下します。

SVM レベルで割り当てた QoS ポリシーを使用すると、 SVM にプロビジョニングされたすべてのボリュームで同じ IOPS プールが共有されることに注意してください。コンテナ化されたアプリケーションの 1 つまたは少数のに高い IOPS が必要な場合、コンテナ化された他のワークロードに対する Bully になる可能性があります。その場合は、外部の自動化を使用したボリュームごとの QoS ポリシーの割り当てを検討してください。

ONTAP バージョン 9.8 より前の場合は、 QoS ポリシーグループを SVM * only * に割り当ててください。

Trident の **QoS** ポリシーグループを作成

Quality of Service （ QoS ；サービス品質）は、競合するワークロードによって重要なワークロードのパフォーマンスが低下しないようにします。ONTAP の QoS ポリシーグループには、ボリュームに対する QoS オプションが用意されており、ユーザは 1 つ以上のワークロードに対するスループットの上限を定義できます。QoSの詳細については、を参照してください。 "QoS によるスループットの保証"。
QoS ポリシーグループはバックエンドまたはストレージプールに指定でき、そのプールまたはバックエンドに作成された各ボリュームに適用されます。

ONTAP には、従来型とアダプティブ型の 2 種類の QoS ポリシーグループがあります。従来のポリシーグループは、最大スループット（以降のバージョンでは最小スループット）がフラットに表示されます。アダプティブ QoS では、ワークロードのサイズの変更に合わせてスループットが自動的に調整され、 TB または GB あたりの IOPS が一定に維持されます。これにより、何百何千という数のワークロードを管理する大規模な環境では大きなメリットが得られます。

QoS ポリシーグループを作成するときは、次の点に注意してください。

- バックエンド構成の「金庫」ブロックに「 QOSPolicy 」キーを設定する必要があります。次のバックエンド設定例を参照してください。

```
  ---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
- labels:
    performance: extreme
  defaults:
    adaptiveQosPolicy: extremely-adaptive-pg
- labels:
    performance: premium
  defaults:
    qosPolicy: premium-pg
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoSポリシーグループの詳細については、を参照してください。 "ONTAP 9.8 QoS コマンド"。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Trident によって作成される NFS ボリュームと iSCSI LUN へのアクセスを制限することは、 Kubernetes 環境のセキュリティ体制に欠かせない要素です。これにより、 Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がないことです。つまり、 PVS はグローバルオブジェクトですが、 PVC にバインドされている場合は、同じネームスペース内のポッドからのみアクセス可能です。* 適切な場合は、名前空間を使用して分離することが重要です。 *

Kubernetes 環境でデータセキュリティを使用する場合、ほとんどの組織で最も懸念されるのは、コンテナ内のプロセスがホストにマウントされたストレージにアクセスできることですが、コンテナ用ではないためです。 "ネームスペース" この種の妥協を防ぐように設計されています。ただし、特権コンテナという例外が 1 つあります。

権限付きコンテナは、通常よりもホストレベルの権限で実行されるコンテナです。デフォルトでは拒否されないため、を使用してこの機能を無効にしてください "ポッドセキュリティポリシー"。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にある

ターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShiftなど）や、ユーザアプリケーションをスケジュールできない他のノードを導入する場合は、ストレージリソースへのアクセスをさらに制限するために別 々 のエクスポートポリシーを使用する必要があります。これには、これらのインフラノードに導入されているサービス（ OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスタ内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Tridentはエクスポートポリシーを自動的に作成、管理できます。これにより、 Trident はプロビジョニング対象のボリュームへのアクセスを Kubernetes クラスタ内のノードに制限し、ノードの追加や削除を簡易化します。

また、エクスポートポリシーを手動で作成し、各ノードのアクセス要求を処理する 1 つ以上のエクスポートルールを設定することもできます。

- 「 vserver export-policy create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーを作成します。
- 「 vserver export-policy rule create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーにルールを追加します。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします `showmount` アプリケーション**SVM**用

「 SVM 」機能を使用すると、 NFS クライアントが SVM に照会して使用可能な NFS エクスポートのリストを表示できます。Kubernetes クラスタに導入されたポッドは、データ LIF に対する「 howmount-e 」コマンドを問題に送信し、アクセス権がないマウントも含め、使用可能なマウントのリストを受信できます。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

SVM レベルの ONTAP CLI コマンドを使用して、 SVM の howmount を無効にする必要があります。

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire のベストプラクティス

Trident に SolidFire ストレージを設定するためのベストプラクティスをご確認ください。

SolidFire アカウントを作成します

各 SolidFire アカウントは固有のボリューム所有者で、 Challenge Handshake Authentication Protocol （ CHAP ；チャレンジハンドシェイク認証プロトコル）クレデンシャルのセットを受け取ります。アカウントに割り当てられたボリュームには、アカウント名とその CHAP クレデンシャルを使用してアクセスするか、ボリュームアクセスグループを通じてアクセスできます。アカウントには最大 2 、 000 個のボリュームを関連付けることができますが、 1 つのボリュームが属することのできるアカウントは 1 つだけです。

QoS ポリシーを作成する

標準的なサービス品質設定を作成して保存し、複数のボリュームに適用する場合は、 SolidFire のサービス品質（ QoS ）ポリシーを使用します。

QoS パラメータはボリューム単位で設定できます。QoS を定義する 3 つの設定可能なパラメータである Min IOPS 、 Max IOPS 、 Burst IOPS を設定することで、各ボリュームのパフォーマンスが保証されます。

4KB のブロックサイズの最小 IOPS 、最大 IOPS 、バースト IOPS の値を次に示します。

IOPS パラメータ	定義（ **Definition** ）	最小価値	デフォルト値	最大値（ **4KB** ）
最小 IOPS	ボリュームに対して保証されたレベルのパフォーマンス。	50	50	15000
最大 IOPS	パフォーマンスはこの制限を超えません。	50	15000	200,000
バースト IOPS	短時間のバースト時に許容される最大 IOPS 。	50	15000	200,000

Max IOPS と Burst IOPS は最大 200 、 000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロックサイズと帯域幅は、 IOPS に直接影響します。ブロックサイズが大きくなると、システムはそのブロックサイズを処理するために必要なレベルまで帯域幅を増やします。帯域幅が増えると、システムが処理可能な IOPS は減少します。を参照してください "SolidFire のサービス品質" QoS およびパフォーマンスの詳細については、を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（ VAG ）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、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。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。

Trident と強化された CSI プロビジョニングツールは、 CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、 SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、 CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、 CHAP 認証の有無に

関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP 認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します "NetApp ライブラリ" 最新バージョンの場合。

- ONTAP *
- "『 NFS Best Practice and Implementation Guide 』を参照してください"
- "『 SAN アドミニストレーションガイド』" （ iSCSI の場合）
- "RHEL 向けの iSCSI のクイック構成"
- Element ソフトウェア *
- "SolidFire for Linux を設定しています"
- NetApp HCI *
- "NetApp HCI 導入の前提条件"
- "NetApp Deployment Engine にアクセスします"
- アプリケーションのベストプラクティス情報 *
- "ONTAP での MySQL に関するベストプラクティスです"
- "SolidFire での MySQL に関するベストプラクティスです"
- "NetApp SolidFire および Cassandra"
- "SolidFire での Oracle のベストプラクティス"
- "SolidFire での PostgreSQL のベストプラクティスです"

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です "NetApp ライブラリ" 最新のドキュメントを検索できます。

Tridentの統合

Tridentを統合するには、ドライバの選択と導入、ストレージクラスの設計、仮想プールの設計、永続的ボリューム要求（PVC）によるストレージプロビジョニングへの影響、ボリューム処理、Tridentを使用したOpenShiftサービスの導入など、設計とアーキテクチャの要素を統合する必要があります。

ドライバの選択と展開

ストレージシステム用のバックエンドドライバを選択して導入します。

ONTAP バックエンドドライバ

ONTAP バックエンドドライバは、使用されるプロトコルと、ストレージシステムでのボリュームのプロビジ

ョニング方法によって異なります。そのため、どのドライバを展開するかを決定する際には、慎重に検討する必要があります。

アプリケーションに共有ストレージを必要とするコンポーネント（同じ PVC にアクセスする複数のポッド）がある場合、 NAS ベースのドライバがデフォルトで選択されますが、ブロックベースの iSCSI ドライバは非共有ストレージのニーズを満たします。アプリケーションの要件と、ストレージチームとインフラチームの快適さレベルに基づいてプロトコルを選択してください。一般的に、ほとんどのアプリケーションでは両者の違いはほとんどないため、共有ストレージ（複数のポッドで同時にアクセスする必要がある場合）が必要かどうかに基づいて判断することがよくあります。

使用可能なONTAP バックエンドドライバは次のとおりです。

- [ONTAP-NAS] ：プロビジョニングされた各 PV は、フル ONTAP FlexVol です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree で、 FlexVol あたりの qtree 数を設定できます（デフォルトは 200 ）。
- 「 ONTAP-NAS-flexgroup 」：フル ONTAP FlexGroup としてプロビジョニングされた各 PV と、 SVM に割り当てられたすべてのアグリゲートが使用されます。
- 「 ONTAP - SAN 」：プロビジョニングされた各 PV は、固有の FlexVol 内の LUN です。
- 「 ONTAP-SAN-エコノミー 」：各 PV がプロビジョニングされた LUN で、 FlexVol あたりの LUN 数を設定できます（デフォルトは 100 ）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、すべての機能がTridentを通じて公開されているわけではないことに注意してください。一部の機能は、プロビジョニング後にストレージ管理者が適用する必要があります。上付き文字の脚注は、機能やドライバごとに機能を区別します。

ONTAP NAS ドライバ	**Snapshot**	クローン	動的なエクスポートポリシー	マルチアタッチ	**QoS**	サイズ変更	レプリケーション
「 ONTAP - NAS 」	はい。	はい。	○脚注： 5[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]
「 ONTAP - NAS - エコノミー」	○脚注： 3[]	○脚注： 3[]	○脚注： 5[]	はい。	○脚注： 3[]	はい。	○脚注： 3[]
「 ONTAP-NAS-flexgroup 」	○脚注： 1[]	いいえ	○脚注： 5[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]

Tridentでは、ONTAP向けに2つのSANドライバを提供しています。その機能は次のとおりです。

ONTAP SAN ドライバ	**Snapshot**	クローン	マルチアタッチ	双方向 **CHAP**	**QoS**	サイズ変更	レプリケーション
「 ontap - san 」	はい。	はい。	○脚注： 4[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]
「 ONTAP - SAN - エコノミー」	はい。	はい。	○脚注： 4[]	はい。	○脚注： 3[]	はい。	○脚注： 3[]

> 上記の表の脚注: Yes [1]: Tridentで管理されていないYes [2]: Tridentで管理されているがPVではないYes [3]: Tridentで管理されていないPVで管理されていないYes [4]: rawブロックボリュームでサポートYes [5]: Tridentでサポート

PV に細分化されていない機能は FlexVol 全体に適用され、 PVS （共有 FlexVol 内の qtree または LUN ）にはすべて共通のスケジュールが適用されます。

上の表に示すように 'ONTAP-NAS' と「 ONTAP-NAS-エコノミー 」の機能の多くは同じですしかし 'ONTAP-NAS-エコノミー のドライバは ' スケジュールを PV 単位で制御する機能を制限するため ' これは特に災害復旧やバックアップ計画に影響を与える可能性がありますONTAP ストレージ上で PVC クローン機能を活用したい開発チームの場合 ' これは 'ONTAP-NAS' ONTAP -SAN' または ONTAP -SAN' のいずれかのドライバを使用する場合にのみ可能です

> 「 olidfire -san 」ドライバは PVC のクローン作成にも対応しています。

Cloud Volumes ONTAP バックエンドドライバ

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（ NFS 、 SMB / CIFS 、 iSCSI ）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAP の互換性のあるドライバは、「 ONTAP-NAS' 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-SAN' 」、「 ONTAP-SAN-エコノミー 」です。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

ONTAP バックエンドドライバ用の**Amazon FSX**

Amazon FSx for NetApp ONTAPを使用すると、AWSにデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用しながら、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用できます。FSx for ONTAPは、多くのONTAPファイルシステム機能と管理APIをサポートしています。Cloud Volume ONTAP の互換性のあるドライバはです `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san` および `ontap-san-economy`。

NetApp HCI / SolidFireバックエンドドライバ

NetApp HCI / SolidFire プラットフォームで使用される「 olidfire -SAN」 ドライバは、管理者が QoS 制限に基づいて Trident の Element バックエンドを構成するのに役立ちます。Trident によってプロビジョニングされるボリュームに特定の QoS 制限を設定するためにバックエンドを設計する場合は、バックエンドファイルの「 type 」パラメータを使用します。管理者は 'limitVolumeSize' パラメータを使用して ' ストレージ上に作成できるボリューム・サイズを制限することもできます現在、ボリュームのサイズ変更やボリュームのレプリケーションなどの Element ストレージ機能は、 'olidfire-san' ドライバではサポートされていません。これらの処理は、 Element ソフトウェアの Web UI から手動で実行する必要があります。

SolidFire ドライバ	**Snapshot**	クローン	マルチアタッチ	**CHAP**	**QoS**	サイズ変更	レプリケーション
「 olidfire -san 」	はい。	はい。	○脚注： 2[]	はい。	はい。	はい。	○脚注： 1[]

> 脚注:はい脚注: 1[]: Tridentで管理されていません脚注: 2[]: raw-blockボリュームでサポートされています

Azure NetApp Files バックエンドドライバ

Tridentはドライバを使用して `azure-netapp-files`サービスを管理し"Azure NetApp Files の特長"ます。

このドライバとその設定方法の詳細については、を参照してください"Azure NetApp Files 向けの Trident バックエンド構成"。

Azure NetApp Files ドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
「 azure-NetApp-files 」と入力します	はい。	はい。	はい。	はい。	はい。	○脚注： 1[]

脚注:はい脚注: 1[]: Tridentで管理されていません

Google Cloudバックエンドドライバ上のCloud Volumes Service

Tridentはドライバを使用し `gcp-cvs`てGoogle Cloud上のCloud Volumes Serviceとリンクします。

`gcp-cvs`ドライバは仮想プールを使用してバックエンドを抽象化し、Tridentがボリュームの配置を決定できるようにします。管理者がファイルに仮想プールを定義し `backend.json`ます。ストレージクラスには、ラベルで仮想プールを識別するセレクタが使用されます。

- バックエンドで仮想プールが定義されている場合、Tridentはそれらの仮想プールが制限されているGoogle Cloudストレージプール内にボリュームを作成しようとします。
- バックエンドで仮想プールが定義されていない場合、Tridentはリージョン内の使用可能なストレージプールからGoogle Cloudストレージプールを選択します。

TridentでGoogle Cloudバックエンドを設定するには、バックエンドファイルで、、 `apiRegion`を `apiKey`指定する必要があります `projectNumber`。プロジェクト番号はGoogle Cloudコンソールで確認できます。APIキーは、Google CloudでCloud Volumes Service のAPIアクセスを設定するときに作成したサービスアカウントの秘密鍵ファイルから取得されます。

Google Cloudのサービスタイプとサービスレベルに関するCloud Volumes Serviceの詳細については、を参照してください"CVS for GCPでのTridentサポートの詳細"。

Cloud Volumes Service for Google Cloudドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
「 gcp-cvs 」	はい。	はい。	はい。	はい。	はい。	CVS -パフォーマンスサービスタイプでのみ利用できます。

レプリケーションに関する注意事項

- レプリケーションはTridentで管理されません。
- クローンは、ソースボリュームと同じストレージプールに作成されます。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは '`toragePools’additionalStoragePools’excludeStoragePools" の 3 セットのフィルタを設定できます

パラメータを使用 `storagePools`すると、指定した属性に一致するプールだけにストレージを制限できます。パラメータは、 `additionalStoragePools`属性とパラメータで選択された一連のプールとともに、Tridentがプロビジョニングに使用する一連のプールを拡張するために使用し `storagePools`ます。どちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

excludeStoragePools' パラメータを使用して ' 属性に一致するプールの一覧を除外します

QoSポリシーをエミュレートします

ストレージクラスを設計して Quality of Service ポリシーをエミュレートする場合は ' 「メディア」属性を「 hdd 」または「 sd 」として ' ストレージクラスを作成しますストレージクラスで言及されている「メディア」属性に基づいて、 Trident は「 hdd 」アグリゲートまたは「 sd 」アグリゲートにメディア属性と一致させる適切なバックエンドを選択し、ボリュームのプロビジョニングを特定のアグリゲートに誘導します。したがって、「メディア」属性が「 SD 」に設定されているストレージクラス Premium を作成して、プレミアム QoS ポリシーに分類できます。メディア属性を「 hdd 」に設定し、標準の QoS ポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「 IOPS 」属性を使用して、 QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用する

ストレージクラスは、シンプロビジョニングとシックプロビジョニング、 Snapshot 、クローン、暗号化などの機能が有効になっている特定のバックエンドでボリュームを直接プロビジョニングするように設計できます。使用するストレージを指定するには、必要な機能を有効にしてバックエンドに適したストレージクラスを作成します。

仮想プール

仮想プールは、すべてのTridentバックエンドで使用できます。Tridentが提供する任意のドライバを使用して、任意のバックエンドに仮想プールを定義できます。

仮想プールを使用すると、管理者はストレージクラスで参照可能なバックエンド上に抽象化レベルを作成して、バックエンドにボリュームを柔軟かつ効率的に配置できます。同じサービスクラスを使用して異なるバックエンドを定義できます。さらに、同じバックエンドに異なる特性を持つ複数のストレージプールを作成することもできます。ストレージクラスに特定のラベルを持つセレクタが設定されている場合、Tridentはボリュー

ムを配置するすべてのセレクタラベルに一致するバックエンドを選択します。ストレージクラスセレクタのラベルが複数のストレージプールに一致する場合、Tridentはそのうちの1つをボリュームのプロビジョニング元として選択します。

仮想プールの設計

バックエンドの作成時に、一般に一連のパラメータを指定できます。管理者が、同じストレージクレデンシャルと異なるパラメータセットを使用して別のバックエンドを作成することはできませんでした。仮想プールの導入により、この問題 は軽減されました。仮想プールは、バックエンドとKubernetesストレージクラスの間に導入されたレベル抽象化です。管理者は、Kubernetes Storage Classesでセレクターとして参照できるラベルとともにパラメータをバックエンドに依存しない方法で定義できます。仮想プールは、Tridentでサポートされるすべてのバックエンドに対して定義できます。リストには、 SolidFire / NetApp HCI 、 ONTAP 、 GCP 上の Cloud Volumes Service 、 Azure NetApp Files が含まれます。

> ⓘ 仮想プールを定義する場合は、バックエンド定義で既存の仮想プールの順序を変更しないことをお勧めします。また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

さまざまなサービスレベル/QoSのエミュレート

サービスクラスをエミュレートするための仮想プールを設計できます。Cloud Volume Service for Azure NetApp Files の仮想プール実装を使用して、さまざまなサービスクラスをセットアップする方法を見ていきましょう。Azure NetApp Filesバックエンドには、異なるパフォーマンスレベルを表す複数のラベルを設定します。設定 `servicelevel` 適切なパフォーマンスレベルを考慮し、各ラベルの下にその他の必要な側面を追加します。次に、異なる仮想プールにマッピングするさまざまなKubernetesストレージクラスを作成します。を使用する `parameters.selector` 各StorageClassは、ボリュームのホストに使用できる仮想プールを呼び出します。

特定の一連の側面を割り当てます

特定の側面を持つ複数の仮想プールは、単一のストレージバックエンドから設計できます。そのためには、バックエンドに複数のラベルを設定し、各ラベルに必要な側面を設定します。を使用して、さまざまなKubernetesストレージクラスを作成します `parameters.selector` 異なる仮想プールにマッピングされるフィールド。バックエンドでプロビジョニングされるボリュームには、選択した仮想プールに定義された設定が適用されます。

ストレージプロビジョニングに影響する **PVC** 特性

要求されたストレージクラスを超える一部のパラメータは、PVCの作成時にTridentプロビジョニングの決定プロセスに影響する可能性があります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つがアクセスモードです。必要なモードは、ストレージ要求をホストするために選択されたバックエンドに影響を与える可能性があります。

Trident は、以下のマトリックスに記載されているアクセス方法で使用されているストレージプロトコルと一致するかどうかを試みます。これは、基盤となるストレージプラットフォームに依存しません。

	ReadWriteOnce コマンドを使用します	**ReadOnlyMany**	**ReadWriteMany**
iSCSI	はい。	はい。	○（ Raw ブロック）
NFS	はい。	はい。	はい。

NFS バックエンドが設定されていない Trident 環境に送信された ReadWriteMany PVC が要求された場合、ボリュームはプロビジョニングされません。このため、リクエスタは、アプリケーションに適したアクセスモードを使用する必要があります。

ボリューム操作

永続ボリュームの変更

永続ボリュームとは、 Kubernetes で変更不可のオブジェクトを 2 つだけ除いてです。再利用ポリシーとサイズは、いったん作成されると変更できます。ただし、これにより、ボリュームの一部の要素がKubernetes以外で変更されることが防止されるわけではありません。特定のアプリケーション用にボリュームをカスタマイズしたり、誤って容量が消費されないようにしたり、何らかの理由でボリュームを別のストレージコントローラに移動したりする場合に便利です。

> Kubernetes のツリー内プロビジョニングツールは、現時点では NFS または iSCSI PVS のボリュームサイズ変更処理をサポートしていません。Trident では、 NFS ボリュームと iSCSI ボリュームの拡張がサポートされています。

作成後に PV の接続の詳細を変更することはできません。

オンデマンドのボリューム **Snapshot** を作成

Trident では、 CSI フレームワークを使用して、ボリュームスナップショットのオンデマンド作成とスナップショットからの PVC の作成がサポートされます。Snapshot は、データのポイントインタイムコピーを管理し、 Kubernetes のソース PV とは無関係にライフサイクルを管理する便利な方法です。これらの Snapshot を使用して、 PVC をクローニングできます。

Snapshot からボリュームを作成します

Trident では、ボリューム Snapshot から PersistentVolumes を作成することもできます。そのためには、PersistentVolumeClaimを作成し、ボリュームの作成元となるSnapshotとしてを指定します `datasource`。Trident は、 Snapshot にデータが存在するボリュームを作成することで、この PVC を処理します。この機能を使用すると、複数のリージョン間でデータを複製したり、テスト環境を作成したり、破損した本番ボリューム全体を交換したり、特定のファイルとディレクトリを取得して別の接続ボリュームに転送したりできます。

クラスタ内でボリュームを移動します

ストレージ管理者は、 ONTAP クラスタ内のアグリゲート間およびコントローラ間で、ストレージ利用者への無停止でボリュームを移動できます。この処理は、Tridentが使用しているSVMからアクセスできるデスティネーションアグリゲートであるかぎり、TridentまたはKubernetesクラスタには影響しません。重要なことは、アグリゲートがSVMに新しく追加されている場合は、バックエンドをTridentに再追加してリフレッシュする必要があることです。これにより、 Trident が SVM のインベントリを再設定し、新しいアグリゲートが認識されます。

ただし、バックエンド間でのボリュームの移動は Trident では自動でサポートされていません。これには、同じクラスタ内の SVM 間、クラスタ間、または別のストレージプラットフォームへの SVM の間も含まれます（ Trident に接続されているストレージシステムの場合も含む）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Trident にインポートできます。

ボリュームを展開します

Tridentでは、NFSおよびiSCSI PVSのサイズ変更がサポートされています。これにより、ユーザは Kubernetes レイヤを介してボリュームのサイズを直接変更できます。ボリュームを拡張できるのは、 ONTAP 、 SolidFire / NetApp HCI 、 Cloud Volumes Service バックエンドなど、主要なすべてのネットアップストレージプラットフォームです。あとで拡張できるようにするには、ボリュームに関連付けられているStorageClassでをに `true`設定し `allowVolumeExpansion`ます。永続的ボリュームのサイズを変更する必要がある場合は、永続的ボリューム要求で必要なボリュームサイズになるようにアノテーションを編集します `spec.resources.requests.storage`。Tridentによって、ストレージクラスタ上のボリュームのサイズが自動的に変更されます。

既存のボリュームを **Kubernetes** にインポートする

Volume Import では、既存のストレージボリュームを Kubernetes 環境にインポートできます。これは現在、「 ONTAP-NAS」 、「 ONTAP-NAS-flexgroup 」、「 solidfire-san-」 、「 azure-netapp-files 」、「 gcp-cvs` ドライバ」でサポートされています。この機能は、既存のアプリケーションを Kubernetes に移植する場合や、ディザスタリカバリシナリオで使用する場合に便利です。

ONTAPドライバとドライバを使用する場合 `solidfire-san`は、コマンドを使用し `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`て、Tridentで管理するKubernetesに既存のボリュームをインポートします。import volume コマンドで使用した PVC YAML または JSON ファイルは、 Trident をプロビジョニングツールとして識別するストレージクラスを指定します。NetApp HCI / SolidFire バックエンドを使用する場合は、ボリューム名が一意であることを確認してください。ボリューム名が重複している場合は、ボリュームインポート機能で区別できるように、ボリュームを一意の名前にクローニングします。

ドライバまたは `gcp-cvs`ドライバを使用している場合 `azure-netapp-files`は、コマンドを使用し `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`て、Tridentで管理するKubernetesにボリュームをインポートします。これにより、ボリューム参照が一意になります。

上記のコマンドが実行されると、 Trident はバックエンド上のボリュームを検出してサイズを確認します。設定されたPVCのボリュームサイズを自動的に追加（および必要に応じて上書き）します。Trident が新しい PV を作成し、 Kubernetes が PVC を PV にバインド

特定のインポートされた PVC を必要とするようにコンテナを導入した場合、ボリュームインポートプロセスによって PVC/PV ペアがバインドされるまで、コンテナは保留状態のままになります。PVC/PV ペアがバインドされると、他に問題がなければコンテナが起動します。

OpenShift サービスを導入します

OpenShift の付加価値クラスタサービスは、クラスタ管理者とホストされているアプリケーションに重要な機能を提供します。これらのサービスが使用するストレージはノードローカルリソースを使用してプロビジョニングできますが、これにより、サービスの容量、パフォーマンス、リカバリ性、持続可能性が制限されることがよくあります。エンタープライズストレージアレイを活用してこれらのサービスに容量を提供することで、劇的に向上したサービスを実現できます。ただし、すべてのアプリケーションと同様に、 OpenShift とストレージ管理者は、緊密に連携してそれぞれに最適なオプションを決定する必要があります。Red Hat のドキュメントは、要件を決定し、サイジングとパフォーマンスのニーズを確実に満たすために大きく活用する必要があ

ります。

レジストリサービス

レジストリのストレージの導入と管理については、に記載されています "netapp.io のコマンドです" を参照してください "ブログ"。

ロギングサービス

他の OpenShift サービスと同様に、ログ記録サービスは、 Ansible と、インベントリファイル（別名）で提供される構成パラメータを使用して導入されますホスト。プレイブックに含まれています。ここでは、 OpenShift の初期インストール時にロギングを導入し、 OpenShift のインストール後にロギングを導入するという、 2 つのインストール方法について説明します。

> Red Hat OpenShift バージョン 3.9 以降、データ破損に関する懸念があるため、記録サービスに NFS を使用しないことを公式のドキュメントで推奨しています。これは、 Red Hat 製品のテストに基づいています。ONTAP NFSサーバにはこのような問題がないため、ロギング環境を簡単にバックアップできます。ロギングサービスには最終的にどちらかのプロトコルを選択する必要がありますが、両方のプロトコルがネットアッププラットフォームを使用する場合に適していることと、 NFS を使用する理由がないことを確認してください。

ロギング・サービスで NFS を使用する場合は、インストーラが失敗しないように、 Ansible 変数「 OpenShift 」の「 OpenShift 」 enable_unsupported _configurations 」を「 true 」に設定する必要があります。

はじめに

ロギングサービスは、必要に応じて、両方のアプリケーションに導入することも、 OpenShift クラスタ自体のコア動作に導入することもできます。オペレーション・ログを配置する場合 ' 変数 OpenShift の logging_use_ops を true として指定すると ' サービスの 2 つのインスタンスが作成されます操作のロギングインスタンスを制御する変数には「 ops 」が含まれ、アプリケーションのインスタンスには含まれません。

基盤となるサービスで正しいストレージが使用されるようにするには、導入方法に応じてAnsible変数を設定することが重要です。それぞれの導入方法のオプションを見てみましょう。

> 次の表には、ロギングサービスに関連するストレージ構成に関連する変数のみを示します。その他のオプションは、で確認できます "Red Hat OpenShift のロギングに関するドキュメント" 導入環境に応じて、確認、設定、使用する必要があります。

次の表の変数では、入力した詳細を使用してロギングサービスの PV と PVC を作成する Ansible プレイブックが作成されます。この方法は、 OpenShift インストール後にコンポーネントインストールプレイブックを使用するよりもはるかに柔軟性に劣るが、既存のボリュームがある場合はオプションとなります。

変数（ **Variable** ）	詳細
「 OpenShift 」ロギング・ストレージ・タイプ	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift 」ロギング・ストレージ・ホスト	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。

変数（ **Variable** ）	詳細
「 OpenShift 」ロギング・ストレージ・ NFS_DIRECT'	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift _logging 」としてジャンクションされている場合、この変数にそのパスを使用します。
「 OpenShift 」ロギング・ストレージ・ボリューム名	作成する PV の名前（「 pv_ose_logs 」など）。
「 OpenShift 」ロギング・ストレージ・ボリューム・サイズ	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
'OpenShift の logging_es_vpc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
「 OpenShift logging _es_vpc_storage_class_name 」	PVC で使用されるストレージクラスの名前。
「 OpenShift logging _es_vpc_size 」を参照してください	PVC で要求されたボリュームのサイズ。
「 OpenShift logging _es_vpc_prefix 」を参照してください	ロギングサービスで使用される PVC のプレフィックス。
'OpenShift の logging_es_ops_pvc_dynamic	動的にプロビジョニングされたボリュームを ops ロギングインスタンスに使用するには、「 true 」に設定します。
「 OpenShift logging _es_ops_pvc_storage_class_name 」を参照してください	処理ロギングインスタンスのストレージクラスの名前。
'OpenShift logging _es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
「 OpenShift logging _es_ops_pvc_prefix 」を参照してください	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、 OpenShift のバージョンが異なるためわずかに変更される場合があるので、必ず読んで従うようにしてください "Red Hat OpenShift Container Platform 3.11 のドキュメント" 使用しているバージョンに対応した

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動拡張機能にも必要であり、多くの組織では、チャージバックやショーバックの

アプリケーションに指標サービスのデータを使用しています。

ロギングサービスや OpenShift 全体と同様に、 Ansible を使用して指標サービスを導入します。また、ロギングサービスと同様に、メトリクスサービスは、クラスタの初期セットアップ中、またはコンポーネントのインストール方法を使用して運用後に導入できます。次の表に、指標サービスに永続的ストレージを設定する際に重要となる変数を示します。

以下の表には、指標サービスに関連するストレージ構成に関連する変数のみが含まれています。このドキュメントには、他にも導入環境に応じて確認、設定、使用できるオプションが多数あります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ storage _kind 」	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift _ metrics _storage_host 」というようになります	NFS ホストのホスト名または IP アドレス。これは SVM のデータ LIF に設定されている必要があります。
「 OpenShift _ metrics _storage_nfs_directory 」というエラーが表示されます	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift メトリック」としてジャンクションされている場合は、この変数にそのパスを使用します。
「 OpenShift _ metrics _storage_volume_name 」という形式で指定します	作成する PV の名前（「 pv_ose_metrics 」など）。
「 OpenShift _ metrics _storage_volume_size 」というようになります	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ cassandra_vpc_prefix 」という形式で指定します	メトリック PVC に使用するプレフィックス。
「 OpenShift _ metrics _ cassandra_vp_size' 」のようになります	要求するボリュームのサイズ。
「 OpenShift _ metrics _ cassandra_storage_type 」のようになります	指標に使用するストレージのタイプ。適切なストレージクラスを使用して PVC を作成するには、 Ansible に対してこれを dynamic に設定する必要があります。
「 OpenShift _ metrics _cassanda_pvc_storage_class_name 」という形式で指定します	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、 Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、 PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合は、OpenShiftのインストール後にAnsibleによって必要なPVCが作成さ

れ、Tridentによってストレージがプロビジョニングされたらサービスが導入されます。

上記の変数と導入プロセスは、 OpenShift の各バージョンで変更される可能性があります。必ず見直しを行ってください "RedHat OpenShift 導入ガイド" をバージョンに合わせて設定し、環境に合わせて設定します。

データ保護とディザスタリカバリ

TridentとTridentを使用して作成されたボリュームの保護とリカバリのオプションについて説明します。永続性に関する要件があるアプリケーションごとに、データ保護とリカバリの戦略を用意しておく必要があります。

Tridentのレプリケーションとリカバリ

災害発生時にTridentをリストアするバックアップを作成できます。

Tridentレプリケーション

Tridentは、Kubernetes CRDを使用して独自の状態を格納および管理し、Kubernetesクラスタetcdを使用してメタデータを格納します。

手順

1. を使用してKubernetesクラスタetcdをバックアップします "Kubernetes：etcdクラスタのバックアップ"。
2. バックアップアーティファクトをFlexVolに配置します。

> FlexVolが配置されているSVMを別のSVMへのSnapMirror関係で保護することを推奨します。

Tridentリカバリ

Kubernetes CRDとKubernetesクラスタetcdスナップショットを使用して、Tridentをリカバリできます。

手順

1. デスティネーションSVMから、Kubernetes etcdデータファイルと証明書が格納されているボリュームを、マスターノードとしてセットアップするホストにマウントします。
2. Kubernetesクラスタに関連する必要な証明書をすべてのにコピーします `/etc/kubernetes/pki` およびの下のetcdメンバーファイル `/var/lib/etcd`。
3. を使用して、etcdバックアップからKubernetesクラスタをリストアします "Kubernetes：etcdクラスタをリストアします"。
4. を実行します `kubectl get crd` Tridentのカスタムリソースがすべて稼働していることを確認し、Tridentオブジェクトを読み出してすべてのデータが利用可能であることを確認します。

SVMのレプリケーションとリカバリ

Tridentではレプリケーション関係を設定できませんが、ストレージ管理者はを使用してSVMをレプリケートできます "ONTAP SnapMirrorの略"。

災害が発生した場合は、 SnapMirror デスティネーション SVM をアクティブ化してデータの提供を開始でき

ます。システムがリストアされたら、プライマリに戻すことができます。

このタスクについて

SnapMirror SVMレプリケーション機能を使用する場合は、次の点を考慮してください。

- SVM-DRを有効にしたSVMごとに、個別のバックエンドを作成する必要があります。
- SVM-DRをサポートするバックエンドにレプリケーション不要のボリュームをプロビジョニングしないように、必要な場合にのみレプリケートされたバックエンドを選択するようにストレージクラスを設定します。
- アプリケーション管理者は、レプリケーションに伴う追加コストと複雑さを理解し、このプロセスを開始する前にリカバリプランを慎重に検討する必要があります。

SVMレプリケーション

を使用できます "ONTAP：SnapMirrorレプリケーション" をクリックしてSVMレプリケーション関係を作成します。

SnapMirrorでは、レプリケートする対象を制御するオプションを設定できます。プリフォーム時に選択したオプションを知っておく必要がTridentを使用したSVMのリカバリあります。

- "-identity-preserve trueを指定します" SVMの設定全体をレプリケートします。
- "-discard-configs network" LIFと関連ネットワークの設定を除外します。
- "-identity-preserve false" ボリュームとセキュリティ設定のみをレプリケートします。

Tridentを使用した**SVM**のリカバリ

Tridentでは、SVMの障害は自動的に検出されません。災害が発生した場合、管理者は新しいSVMへのTridentフェイルオーバーを手動で開始できます。

手順

1. スケジュールされた実行中のSnapMirror転送をキャンセルし、レプリケーション関係を解除し、ソースSVMを停止してからSnapMirrorデスティネーションSVMをアクティブ化します。
2. を指定した場合 `-identity-preserve false` または `-discard-config network` SVMレプリケーションを設定する場合は、を更新します `managementLIF` および `dataLIF` をTridentバックエンド定義ファイルに追加します。
3. 確認します `storagePrefix` は、Tridentバックエンド定義ファイルに含まれています。このパラメータは変更できません。省略しています `storagePrefix` バックエンドの更新が失敗するように原因します。
4. 次のコマンドを使用して、必要なすべてのバックエンドを更新して新しいデスティネーションSVM名を反映します。

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. を指定した場合 `-identity-preserve false` または `discard-config network`、すべてのアプリケーションポッドをバウンスする必要があります。

を指定する `-identity-preserve true`と、デスティネーションSVMがアクティブ化されたときに、Tridentによってプロビジョニングされたすべてのボリュームからデータの提供が開始されます。

ボリュームのレプリケーションとリカバリ

TridentではSnapMirrorレプリケーション関係を設定できませんが、ストレージ管理者はを使用して、Tridentで作成されたボリュームをレプリケートできます"ONTAP SnapMirrorのレプリケーションとリカバリ"。

その後、を使用して、リカバリしたボリュームをTridentにインポートできます"tridentctlボリュームインポート"。

ではインポートはサポートされていません `ontap-nas-economy`、 `ontap-san-economy`または `ontap-flexgroup-economy` ドライバ。

Snapshotデータの保護

次のコマンドを使用してデータを保護およびリストアできます。

- 永続ボリューム（PV）のKubernetesボリュームSnapshotを作成するための外部のSnapshotコントローラとCRD。

  "ボリューム Snapshot"

- ONTAP Snapshot：ボリュームの内容全体のリストア、または個 々 のファイルまたはLUNのリカバリに使用します。

  "ONTAPスナップショット"

セキュリティ

セキュリティ

ここに記載されている推奨事項を使用して、Tridentのインストールが安全であることを確認します。

独自のネームスペースで**Trident**を実行

信頼性の高いストレージを確保し、潜在的な悪意のあるアクティビティをブロックするためには、アプリケーション、アプリケーション管理者、ユーザ、管理アプリケーションがTridentオブジェクト定義やポッドにアクセスできないようにすることが重要です。

他のアプリケーションとユーザをTridentから分離するには、必ずTridentを独自のKubernetesネームスペースにインストールし(`trident`ます）。Tridentを独自のネームスペースに配置すると、Kubernetes管理者のみがTridentポッドと、名前空間CRDオブジェクトに格納されているアーティファクト（該当する場合はバックエンドやCHAPシークレットなど）にアクセスできるようになります。Tridentネームスペースへのアクセスを管理者のみに許可し、アプリケーションへのアクセスを許可する必要があり `tridentctl`ます。

ONTAP SAN バックエンドで CHAP 認証を使用します

Tridentでは、ONTAP SANワークロードに対してCHAPベースの認証がサポートされます（ドライバと `ontap-san-economy` ドライバを使用 `ontap-san`）。NetAppでは、ホストとストレージバックエンド間の認証にTridentで双方向CHAPを使用することを推奨しています。

SANストレージドライバを使用するONTAPバックエンドの場合、Tridentは双方向CHAPを設定し、でCHAPユーザ名とシークレットを管理できます `tridentctl`。TridentがONTAPバックエンドでCHAPを構成する方法については、を参照してください"バックエンドにONTAP SANドライバを設定する準備をします"。

NetApp HCI および SolidFire バックエンドで CHAP 認証を使用します

ホストと NetApp HCI バックエンドと SolidFire バックエンドの間の認証を確保するために、双方向の CHAP を導入することを推奨します。Tridentは、テナントごとに2つのCHAPパスワードを含むシークレットオブジェクトを使用します。Tridentをインストールすると、CHAPシークレットが管理され、それぞれのPVのCRオブジェクトに格納され `tridentvolume` ます。PVを作成すると、TridentはCHAPシークレットを使用してiSCSIセッションを開始し、CHAPを介してNetApp HCIおよびSolidFireシステムと通信します。

> Tridentで作成されるボリュームは、どのボリュームアクセスグループにも関連付けられません。

NVEおよびNAEでのTridentの使用

NetApp ONTAP は、保管データの暗号化を提供し、ディスクが盗難、返却、転用された場合に機密データを保護します。詳細については、を参照してください "NetApp Volume Encryption の設定の概要"。

- 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すべてのボリュームでNAEが有効になります。
- バックエンドでNAEが有効になっていない場合、バックエンド構成でNVE暗号化フラグをに設定しないかぎり、TridentでプロビジョニングされたボリュームはNVE対応になり `false` ます。

> NAE対応バックエンドのTridentで作成されたボリュームは、NVEまたはNAEで暗号化する必要があります。
>
> - Tridentバックエンド構成でNVE暗号化フラグを「true」に設定すると、NAE暗号化をオーバーライドし、ボリューム単位で特定の暗号化キーを使用できます。
> - NAE対応バックエンドでNVE暗号化フラグをに設定する `false` と、NAE対応ボリュームが作成されます。NVE暗号化フラグをに設定してNAE暗号化を無効にすることはできません `false`。

- TridentでNVEボリュームを手動で作成するには、NVE暗号化フラグを明示的にに設定し `true` ます。

バックエンド構成オプションの詳細については、以下を参照してください。

- "ONTAP のSAN構成オプション"
- "ONTAP NASの構成オプション"

Linux Unified Key Setup（LUKS；統合キーセットアップ）

Linuxユニファイドキーセットアップ（LUKS）を有効にして、Trident上のONTAP SAN

およびONTAP SANエコノミーボリュームを暗号化できます。Tridentは、LUKSで暗号化されたボリュームのパスフレーズのローテーションとボリューム拡張をサポートしています。

Tridentでは、LUKSで暗号化されたボリュームでAES-XTS-plain64暗号化およびモードが使用されます（の推奨）"NIST"。

作業を開始する前に

- ワーカーノードにはcryptsetup 2.1以上（3.0よりも下位）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、を参照してください "Gitlab: cryptsetup"。
- パフォーマンス上の理由から、ワーカーノードでAdvanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨します。AES-NIサポートを確認するには、次のコマンドを実行します。

```
grep "aes" /proc/cpuinfo
```

何も返されない場合、お使いのプロセッサはAES-NIをサポートしていません。AES-NIの詳細については、以下を参照してください。 "Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にします

ONTAP SANおよびONTAP SANエコノミーボリュームでは、Linux Unified Key Setup（LUKS；Linux統合キーセットアップ）を使用して、ボリューム単位のホスト側暗号化を有効にできます。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。ONTAP SANのバックエンド構成オプションの詳細については、を参照してください "ONTAP のSAN構成オプション"。

```
"storage": [
    {
        "labels":{"luks": "true"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "true"
        }
    },
    {
        "labels":{"luks": "false"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "false"
        }
    },
]
```

2. 使用 `parameters.selector` LUKS暗号化を使用してストレージプールを定義する方法。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKSパスフレーズを含むシークレットを作成します。例：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

制限

LUKSで暗号化されたボリュームは、ONTAP の重複排除と圧縮を利用できません。

LUKSボリュームをインポートするためのバックエンド構成

LUKSボリュームをインポートするには、バックエンドでをに(`true`設定する必要があります `luksEncryption`。このオプションを指定する `luksEncryption`と、(`false`次の例に示すように、ボリュームがLUKS準拠である(`true`かどうかがTridentに通知されます。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

LUKSボリュームをインポートするためのPVC設定

LUKSボリュームを動的にインポートするには、 `trident.netapp.io/luksEncryption` `true`次の例に示すように、アノテーションをに設定し、LUKS対応のストレージクラスをPVCに含めます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

LUKSパスフレーズをローテーションします

LUKSのパスフレーズをローテーションしてローテーションを確認できます。

> パスフレーズは、ボリューム、Snapshot、シークレットで参照されなくなることを確認するまで忘れないでください。参照されているパスフレーズが失われた場合、ボリュームをマウントできず、データが暗号化されたままアクセスできなくなることがあります。

このタスクについて

LUKSパスフレーズのローテーションは、ボリュームをマウントするポッドが、新しいLUKSパスフレーズの指定後に作成されたときに行われます。新しいPODが作成されると、Tridentはボリューム上のLUKSパスフレーズをシークレット内のアクティブなパスフレーズと比較します。

- ボリュームのパスフレーズがシークレットでアクティブなパスフレーズと一致しない場合、ローテーションが実行されます。
- ボリュームのパスフレーズがシークレットのアクティブなパスフレーズと一致する場合は、を参照してください `previous-luks-passphrase` パラメータは無視されます。

手順

1. を追加します `node-publish-secret-name` および `node-publish-secret-namespace` StorageClassパラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. ボリュームまたはSnapshotの既存のパスフレーズを特定します。

   ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

   スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. ボリュームのLUKSシークレットを更新して、新しいパスフレーズと前のパスフレーズを指定します。確認します `previous-luke-passphrase-name` および `previous-luks-passphrase` 前のパスフレーズと同じにします。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. ボリュームをマウントする新しいポッドを作成します。これはローテーションを開始するために必要です。

5. パスフレーズがローテーションされたことを確認します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

パスフレーズは、ボリュームとSnapshotに新しいパスフレーズのみが返されたときにローテーションされました。

> たとえば、2つのパスフレーズが返された場合などです `luksPassphraseNames: ["B", "A"]`回転が不完全です。回転を完了するために、新しいポッドをトリガできます。

ボリュームの拡張を有効にします

LUKS暗号化ボリューム上でボリューム拡張を有効にできます。

手順

1. を有効にします `CSINodeExpandSecret` 機能ゲート（ベータ1.25+）。を参照してください "Kubernetes 1.25：CSIボリュームのノードベースの拡張にシークレットを使用します" を参照してください。
2. を追加します `node-expand-secret-name` および `node-expand-secret-namespace` StorageClass パラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

結果

ストレージのオンライン拡張を開始すると、ドライバに適切なクレデンシャルが渡されます。

Trident保護でアプリケーションを保護

Trident protectの詳細

NetApp Trident Protectは、NetApp ONTAPストレージシステムとNetApp Trident CSIストレージプロビジョニングツールを基盤とするステートフルなKubernetesアプリケーションの機能と可用性を強化する、高度なアプリケーションデータ管理機能を提供します。Trident Protectは、パブリッククラウドとオンプレミス環境にわたるコンテナ化されたワークロードの管理、保護、移動を簡易化します。また、APIとCLIを使用した自動化機能も提供します。

Trident保護を使用してアプリケーションを保護するには、カスタムリソース（CRS）を作成するか、Trident保護CLIを使用します。

次の手順

インストールする前に、Trident保護の要件について確認できます。

- "Trident保護の要件"

Tridentプロテクトのインストール

Trident保護の要件

まず、運用環境、アプリケーションクラスタ、アプリケーション、ライセンスの準備状況を確認します。Trident保護を導入して運用するには、環境がこれらの要件を満たしていることを確認してください。

TridentはKubernetesの互換性を保護

Trident Protectは、次のようなフルマネージドおよび自己管理型の幅広いKubernetes製品と互換性があります。

- Amazon Elastic Kubernetes Service（EKS）
- Google Kubernetes Engine （ GKE ）
- Microsoft Azure Kubernetes Service （ AKS ）
- Red Hat OpenShift のサービスです
- SUSE Rancher
- VMware Tanzuポートフォリオ
- アップストリームKubernetes

Tridentはストレージバックエンドの互換性を保護

Trident保護は、次のストレージバックエンドをサポートします。

- NetApp ONTAP 対応の Amazon FSX
- Cloud Volumes ONTAP
- ONTAPストレエシアレイ
- Google Cloud NetAppボリューム
- Azure NetApp Files の特長

ストレージバックエンドが次の要件を満たしていることを確認します。

- クラスタに接続されているNetAppストレージがAstra Trident 24.02以降を使用していることを確認します（Trident 24.10を推奨）。
  - Astra Tridentが24.06.1より前のバージョンで、NetApp SnapMirrorディザスタリカバリ機能を使用する場合は、Astra Control Provisionerを手動で有効にする必要があります。
- 最新のAstra Control Provisionerがインストールされていることを確認します（Astra Trident 24.06.1以降ではデフォルトでインストールおよび有効化されています）。
- NetApp ONTAPストレージバックエンドがあることを確認します。
- バックアップを格納するオブジェクトストレージバケットを設定しておきます。
- アプリケーションまたはアプリケーションデータの管理処理に使用するアプリケーションネームスペースを作成します。Trident保護では、これらのネームスペースは作成されません。カスタムリソースに存在しないネームスペースを指定すると、処理は失敗します。

NASエコノミーボリュームの要件

Trident Protectは、NASエコノミーボリュームへのバックアップおよびリストア処理をサポートします。Snapshot、クローン、NASエコノミーボリュームへのSnapMirrorレプリケーションは、現在サポートされていません。Trident保護で使用するNASエコノミーボリュームごとに、スナップショットディレクトリを有効にする必要があります。

> 一部のアプリケーションは、Snapshotディレクトリを使用するボリュームと互換性がありません。これらのアプリケーションでは、ONTAPストレージシステムで次のコマンドを実行して、snapshotディレクトリを非表示にする必要があります。
>
> ```
> nfs modify -vserver <svm> -v3-hide-snapshot enabled
> ```

snapshotディレクトリを有効にするには、NASエコノミーボリュームごとに次のコマンドを実行し、を変更するボリュームのUUIDに置き換え `<volume-UUID>`ます。

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

> 新しいボリュームに対してSnapshotディレクトリをデフォルトで有効にするには、Tridentバックエンド構成オプションをに `true`設定し `snapshotDir`ます。既存のボリュームには影響しません。

SnapMirrorレプリケーションの要件

NetApp SnapMirrorは、次のONTAPソリューションのTrident protectで使用できます。

- NetApp ASA
- NetApp AFF
- NetApp FAS
- NetApp ONTAP Select の略
- NetApp Cloud Volumes ONTAP の略
- NetApp ONTAP 対応の Amazon FSX

SnapMirrorレプリケーション用のONTAPクラスタの要件

SnapMirrorレプリケーションを使用する場合は、ONTAPクラスタが次の要件を満たしていることを確認します。

- * Astra Control ProvisionerまたはTrident *：Astra Control ProvisionerまたはTridentが、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クラスタに存在している必要があります。Trident保護では、次のドライバに基づくストレージクラスを使用したNetApp SnapMirrorテクノロジによるレプリケーションがサポートされます。
  - 「 ONTAP - NAS 」
  - 「 ontap - san 」
- ライセンス：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデスティネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

SnapMirrorレプリケーションのピアリングに関する考慮事項

ストレージバックエンドピアリングを使用する場合は、環境が次の要件を満たしていることを確認してください。

- *クラスタとSVM *：ONTAPストレージバックエンドにピア関係が設定されている必要があります。詳細については、を参照してください "クラスタと SVM のピアリングの概要" 。

  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。

- * Astra Control ProvisionerまたはTridentとSVM *：ピア関係にあるリモートSVMは、デスティネーションクラスタのAstra Control ProvisionerまたはTridentで使用できる必要があります。
- 管理バックエンド：レプリケーション関係を作成するには、Trident保護でONTAPストレージバックエンドを追加および管理する必要があります。
- * NVMe over TCP *：Trident保護では、NVMe over TCPプロトコルを使用するストレージバックエンドのNetApp SnapMirrorレプリケーションはサポートされません。

SnapMirrorレプリケーション用のTrident / ONTAPの設定

Trident保護を使用するには、ソースとデスティネーションの両方のクラスタのレプリケーションをサポートするストレージバックエンドを少なくとも1つ設定する必要があります。ソースクラスタとデスティネーション

クラスタが同じである場合は、耐障害性を最大限に高めるために、デスティネーションアプリケーションでソースアプリケーションとは別のストレージバックエンドを使用する必要があります。

KubeVirt使用時の考慮事項

SnapMirrorレプリケーションで仮想マシンを使用する場合 "KubeVirt"は、仮想化を設定して、SVMのフリーズおよびフリーズ解除を実行できるようにする必要があります。仮想化のセットアップが完了すると、導入するSVMに、フリーズおよびフリーズ解除に必要なツールが含まれるようになります。仮想化の設定の詳細については、を参照してください "OpenShift Virtualizationのインストール"。

Trident保護のインストールと設定

ご使用の環境がTrident保護の要件を満たしている場合は、次の手順に従ってクラスタにTrident保護をインストールします。NetAppからTrident protectを取得するか、独自のプライベートレジストリからインストールできます。プライベートレジストリからインストールすると、クラスタがインターネットにアクセスできない場合に役立ちます。

デフォルトでは、Trident protectは、クラスタと管理対象アプリケーションに関するログ、指標、トポロジ情報など、NetAppサポートケースをオープンする際に役立つサポート情報を収集します。Trident PROTECTは、これらのサポートバンドルを日次スケジュールでNetAppに送信します。Trident protectのインストール時に、必要に応じてこのサポートバンドル収集を無効にすることができます。いつでも手動で行うことができ"サポートバンドルの生成"ます。

Trident protect from NetAppのインストール

1. Trident Helmリポジトリを追加します。

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

2. TridentプロテクトCRDをインストールします。

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.0 --create-namespace --namespace
trident-protect
```

3. 次のいずれかのコマンドを使用して、Helmを使用してTrident protectをインストールします。をクラスタ名に置き換えます `<name_of_cluster>`。クラスタに割り当てられ、クラスタのバックアップとスナップショットの識別に使用されます。
   - Trident保護を通常どおりインストールします。

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.0
--create-namespace --namespace trident-protect
```

   - Trident protectをインストールし、Trident protect AutoSupportサポートバンドルのスケジュールされた毎日のアップロードを無効にします。

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.0 --create
-namespace --namespace trident-protect
```

4. 必要に応じて、VMをフリーズします。SnapMirrorでKubeVirtのサポートを使用している場合は、VMをフリーズすると効果的に管理できます。

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

フリーズ機能を動作させるには、仮想化を設定する必要があります。このセットアップ後に導入されたVMには、フリーズおよびフリーズ解除に必要なバイナリが含まれます。仮想化の設定の詳細については、を参照してください"OpenShift Virtualizationのインストール"。

Trident protectをプライベートレジストリからインストールする

Kubernetesクラスタがインターネットにアクセスできない場合は、プライベートイメージレジストリからTrident protectをインストールできます。次の例では、括弧内の値を環境の情報に置き換えます。

1. 次のイメージをローカルマシンにプルし、タグを更新して、プライベートレジストリにプッシュします。

```
netapp/controller:24.10.0
netapp/restic:24.10.0
netapp/kopia:24.10.0
netapp/trident-autosupport:24.10.0
netapp/exechook:24.10.0
netapp/resourcebackup:24.10.0
netapp/resourcerestore:24.10.0
netapp/resourcedelete:24.10.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例：

```
docker pull netapp/controller:24.10.0
```

```
docker tag netapp/controller:24.10.0 <private-registry-
url>/controller:24.10.0
```

```
docker push <private-registry-url>/controller:24.10.0
```

2. Trident protect systemネームスペースを作成します。

```
kubectl create ns trident-protect
```

3. レジストリにログインします。

```
helm registry login <private-registry-url> -u <account-id> -p <api-
token>
```

4. プライベートレジストリ認証に使用するプルシークレットを作成します。

```
kubectl create secret docker-registry regcred --docker
-username=<registry-username> --docker-password=<api-token> -n
trident-protect --docker-server=<private-registry-url>
```

5. Trident Helmリポジトリを追加します。

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

6. 次のTrident保護設定を含むという名前のファイルを作成します `protectValues.yaml`。

```
image:
  registry: <private-registry-url>
imagePullSecrets:
- name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
  - name: regcred
webhooksCleanup:
  imagePullSecrets:
  - name: regcred
```

7. TridentプロテクトCRDをインストールします。

```
helm install trident-protect-crds netapp-trident-protect/trident-
protect-crds --version 100.2410.0 --create-namespace --namespace
trident-protect
```

8. 次のいずれかのコマンドを使用して、Helmを使用してTrident protectをインストールします。をクラスタ名に置き換えます `<name_of_cluster>`。クラスタに割り当てられ、クラスタのバックアップとスナップショットの識別に使用されます。
   - Trident保護を通常どおりインストールします。

```
helm install trident-protect netapp-trident-protect/trident-
protect --set clusterName=<name_of_cluster> --version 100.2410.0
--create-namespace --namespace trident-protect -f
protectValues.yaml
```

- Trident protectをインストールし、Trident protect AutoSupportサポートバンドルのスケジュールされた毎日のアップロードを無効にします。

```
helm install trident-protect netapp-trident-protect/trident-
protect --set autoSupport.enabled=false --set
clusterName=<name_of_cluster> --version 100.2410.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

9. 必要に応じて、VMをフリーズします。SnapMirrorでKubeVirtのサポートを使用している場合は、VMをフリーズすると効果的に管理できます。

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

フリーズ機能を動作させるには、仮想化を設定する必要があります。このセットアップ後に導入されたVMには、フリーズおよびフリーズ解除に必要なバイナリが含まれます。仮想化の設定の詳細については、を参照してください"OpenShift Virtualizationのインストール"。

Trident保護CLIプラグインのインストール

Tridentユーティリティの拡張機能であるTrident protectコマンドラインプラグインを使用すると、Trident protectカスタムリソース（CRS）を作成して操作できます `tridentctl`。

Trident保護CLIプラグインのインストール

コマンドラインユーティリティを使用する前に、クラスタへのアクセスに使用するマシンにインストールする必要があります。マシンがx64またはARM CPUを使用しているかどうかに応じて、次の手順を実行します。

Linux AMD64 CPU用プラグインのダウンロード

1. Trident保護CLIプラグインをダウンロードします。

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.0/tridentctl-protect-linux-amd64
```

Linux ARM64 CPU用プラグインのダウンロード

1. Trident保護CLIプラグインをダウンロードします。

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.0/tridentctl-protect-linux-arm64
```

Mac AMD64 CPU用プラグインのダウンロード

1. Trident保護CLIプラグインをダウンロードします。

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.0/tridentctl-protect-macos-amd64
```

Mac ARM64 CPU用プラグインのダウンロード

1. Trident保護CLIプラグインをダウンロードします。

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/24.10.0/tridentctl-protect-macos-arm64
```

1. バイナリの実行権限を有効にします。

```
chmod +x tridentctl-protect
```

2. プラグインバイナリをPATH変数で定義されている場所にコピーします。たとえば、 `/usr/bin`または`/usr/local/bin`（昇格されたPrivilegesが必要な場合があります）。

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 必要に応じて、バイナリをホームディレクトリ内の場所にコピーできます。この場合は、PATH変数に場所を追加する必要があります。

```
cp ./tridentctl-protect ~/bin/
```

Trident CLIプラグインのヘルプを表示

組み込みプラグインヘルプ機能を使用して、プラグインの機能に関する詳細なヘルプを表示できます。

手順

1. ヘルプ機能を使用して、使用方法に関するガイダンスを表示します。

```
tridentctl protect help
```

コマンドの自動補完を有効にする

Trident保護CLIプラグインをインストールしたあとで、特定のコマンドの自動補完を有効にすることができます。

Bashシェルの自動補完を有効にする

1. 完了スクリプトをダウンロードします。

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.0/tridentctl-completion.bash
```

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

```
mkdir -p ~/.bash/completions
```

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.bash/completions`ます。

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. ホームディレクトリ内のファイルに次の行を追加し `~/.bashrc`ます。

```
source ~/.bash/completions/tridentctl-completion.bash
```

Zシェルの自動補完を有効にする

1. 完了スクリプトをダウンロードします。

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/24.10.0/tridentctl-completion.zsh
```

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

```
mkdir -p ~/.zsh/completions
```

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.zsh/completions`ます。

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. ホームディレクトリ内のファイルに次の行を追加し `~/.zprofile`ます。

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

結果

次のシェルログイン時に、tridentctl protectプラグインで自動補完コマンドを使用できます。

Trident保護を管理します。

許可とアクセス制御の管理

Trident保護では、KubernetesモデルのRole-Based Access Control（RBAC；ロールベースアクセス制御）が使用されます。デフォルトでは、Trident保護は単一のシステムネームスペースとそれに関連付けられたデフォルトのサービスアカウントを提供します。多数のユーザがいる組織や、特定のセキュリティニーズがある組織では、Trident保護のRBAC機能を使用して、リソースやネームスペースへのアクセスをより細かく制御できます。

クラスタ管理者は、常にデフォルトのネームスペース内のリソースにアクセスできます `trident-protect`。また、他のすべてのネームスペース内のリソースにもアクセスできます。リソースとアプリケーションへのアクセスを制御するには、追加の名前空間を作成し、それらの名前空間にリソースとアプリケーションを追加する必要があります。

デフォルトの名前空間にアプリケーションデータ管理CRSを作成することはできないことに注意して `trident-protect`ください。アプリケーションデータ管理CRSは、アプリケーションネームスペース内に作成する必要があります（ベストプラクティスとして、アプリケーションデータ管理CRSは、関連付けられているアプリケーションと同じネームスペースに作成します）。

> 管理者のみが、次のような特権Trident保護カスタムリソースオブジェクトへのアクセス権を持つ必要があります。
>
> - * AppVault *：バケット資格情報データが必要です。
> - * AutoSupportBundle *：指標、ログ、その他の機密性の高いTridentデータを収集します。
> - * AutoSupportBundleSchedule *：ログ収集スケジュールを管理します。
>
> RBACを使用して、権限付きオブジェクトへのアクセスを管理者に制限することを推奨します。

RBACでリソースおよびネームスペースへのアクセスを制御する方法の詳細については、を参照して"Kubernetes RBACのドキュメント"ください。

サービスアカウントの詳細については、を参照して "Kubernetesサービスアカウントのドキュメント"ください。

例：2つのユーザグループのアクセスを管理する

たとえば、ある組織に、クラスタ管理者、エンジニアリングユーザのグループ、およびマーケティングユーザのグループがあるとします。クラスタ管理者は次のタスクを実行して、engineeringグループとmarketingグループがそれぞれのネームスペースに割り当てられたリソースのみにアクセスできる環境を作成します。

手順1：各グループのリソースを含むネームスペースを作成する

ネームスペースを作成すると、リソースを論理的に分離し、それらのリソースにアクセスできるユーザをより

細かく制御できます。

手順

1. engineeringグループの名前空間を作成します。

```
kubectl create ns engineering-ns
```

2. marketingグループの名前空間を作成します。

```
kubectl create ns marketing-ns
```

ステップ**2**：各ネームスペースのリソースとやり取りするための新しいサービスアカウントを作成する

作成する新しい名前空間にはそれぞれデフォルトのサービスアカウントが付属していますが、将来必要に応じてPrivilegesをグループ間でさらに分割できるように、ユーザーのグループごとにサービスアカウントを作成する必要があります。

手順

1. engineeringグループのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. マーケティンググループのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

ステップ**3**：新しいサービスアカウントごとにシークレットを作成する

サービスアカウントシークレットは、サービスアカウントでの認証に使用され、侵害された場合は簡単に削除および再作成できます。

手順

1. エンジニアリングサービスアカウントのシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. マーケティングサービスアカウントのシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

手順4：**RoleBinding**オブジェクトを作成して、**ClusterRole**オブジェクトを新しい各サービスアカウントにバインドする

Trident保護をインストールすると、デフォルトのClusterRoleオブジェクトが作成されます。このClusterRoleをサービスアカウントにバインドするには、RoleBindingオブジェクトを作成して適用します。

手順

1. ClusterRoleをエンジニアリングサービスアカウントにバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. ClusterRoleをマーケティングサービスアカウントにバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

手順5：権限のテスト

権限が正しいことをテストします。

手順

1. エンジニアリングユーザーがエンジニアリングリソースにアクセスできることを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. エンジニアリングユーザーがマーケティングリソースにアクセスできないことを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

手順6：AppVaultオブジェクトへのアクセスを許可する

バックアップやスナップショットなどのデータ管理タスクを実行するには、クラスタ管理者が個 々 のユーザーにAppVaultオブジェクトへのアクセスを許可する必要があります。

手順

1. AppVaultへのユーザーアクセスを許可するAppVaultとシークレットの組み合わせYAMLファイルを作成して適用します。たとえば、次のCRは、AppVaultへのアクセスをユーザーに許可し `eng-user`ます。

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 役割CRを作成して適用し、クラスタ管理者がネームスペース内の特定のリソースへのアクセスを許可できるようにします。例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. RoleBinding CRを作成して適用し、権限をeng-userというユーザにバインドします。例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 権限が正しいことを確認します。

   a. すべての名前空間のAppVaultオブジェクト情報の取得を試みます。

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   次のような出力が表示されます。

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. ユーザがAppVault情報を取得できるかどうかをテストして、アクセス許可を得ているかどうかを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

次のような出力が表示されます。

```
yes
```

結果

AppVault権限を付与したユーザーは、アプリケーションデータ管理操作に承認されたAppVaultオブジェクトを使用できる必要があります。また、割り当てられた名前空間以外のリソースにアクセスしたり、アクセスできない新しいリソースを作成したりすることはできません。

サポートバンドルの生成

Trident protectを使用すると、管理者は、管理対象のクラスタとアプリケーションに関するログ、指標、トポロジ情報など、NetAppサポートに役立つ情報を含むバンドルを生成できます。インターネットに接続している場合は、カスタムリソース（CR）ファイルを使用してNetAppサポートサイト（NSS）にサポートバンドルをアップロードできます。

CRを使用したサポートバンドルの作成

1. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-support-bundle.yaml`）。
2. 次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.triggerType * :（*required*）サポートバンドルをすぐに生成するかスケジュールするかを指定します。スケジュールされたバンドル生成は12AM UTCに行われます。有効な値：
     - スケジュール済み
     - 手動
   - * spec.uploadEnabled * :（_Optional _）サポートバンドルの生成後にNetAppサポートサイトにアップロードするかどうかを制御します。指定しない場合、デフォルトはになります `false`。有効な値：
     - 正しいです
     - false（デフォルト）
   - **spec.dataWindowStart**:(*Optional*)サポートバンドルに含まれるデータのウィンドウを開始する日時を指定する、RFC 3339形式の日付文字列。指定しない場合は、デフォルトで24時間前になります。指定できる最も早い期間の日付は7日前です。

     YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. ファイルに正しい値を入力したら `astra-support-bundle.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-support-bundle.yaml
```

CLIを使用したサポートバンドルの作成

1. サポートバンドルを作成し、角かっこ内の値を環境からの情報に置き換えます。は `trigger-type`、バンドルをすぐに作成するか、スケジュールによって作成時間が指定されているかを決定し、または `Scheduled`を指定できます `Manual`。デフォルト設定はです `Manual`。

   例：

```
tridentctl protect create autosupportbundle <my_bundle_name>
--trigger-type <trigger_type>
```

アプリケーションの管理と保護

AppVaultオブジェクトを使用してバケットを管理する

Trident保護用のバケットカスタムリソース（CR）は、AppVaultと呼ばれます。AppVaultオブジェクトは、ストレージバケットの宣言型Kubernetesワークフロー表現です。AppVault CRには、バックアップ、Snapshot、リストア処理、SnapMirrorレプリケーションなど、保護処理でバケットを使用するために必要な設定が含まれています。AppVaultsを作成できるのは管理者のみです。

キー生成とAppVault定義の例

AppVault CRを定義するときは、プロバイダがホストするリソースにアクセスするための資格情報を含める必要があります。クレデンシャルのキーの生成方法は、プロバイダによって異なります。次に、いくつかのプロバイダのコマンドラインキー生成の例を示します。次に、各プロバイダのAppVault定義の例を示します。

Google Cloud

キー生成の例：

```
kubectl create secret generic gcp-creds --from-file=credentials=<mycreds
-file.json> -n trident-protect
```

次のAppVault定義の例は、使用および変更可能なCR、またはAppVault CRを生成するTrident protect CLIコマンドの例として提供されています。

AppVault CRの例

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Trident保護CLIを使用したAppVault CRの作成例

```
tridentctl protect create vault gcp my-new-vault --bucket mybucket
--project my-gcp-project --secret <gcp-creds>/<credentials>
```

Amazon S3

キー生成の例：

```
kubectl create secret generic -n trident-protect s3 --from
-literal=accessKeyID=<secret-name> --from-literal=secretAccessKey
=<generic-s3-trident-protect-src-bucket-secret>
```

次のAppVault定義の例は、使用および変更可能なCR、またはAppVault CRを生成するTrident protect CLIコマンドの例として提供されています。

AppVault CRの例

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3
```

CLIを使用したAppVaultの作成例

```
tridentctl protect create vault GenericS3 s3vault --bucket <bucket-
name> --secret  <secret-name>  --endpoint <s3-endpoint>
```

Microsoft Azure

キー生成の例：

```
kubectl create secret generic <secret-name> --from-literal=accountKey
=<secret-name> -n trident-protect
```

次のAppVault定義の例は、使用および変更可能なCR、またはAppVault CRを生成するTrident protect CLIコマンドの例として提供されています。

AppVault CRの例

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket-b643cc50-0429-4ad5-971f-
ac4a83621922
  namespace: trident-protect
spec:
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

CLIを使用したAppVaultの作成例

```
tridentctl protect create vault Azure <vault-name> --account <account-
name> --bucket <bucket-name> --secret <secret-name>
```

providerTypeおよびproviderConfigでサポートされる値

`providerType`AppVault CRのキーと`providerConfig`キーには、特定の値が必要です。次の表に、キーでサポートされている値と、各値で使用する必要がある関連`providerConfig`キーを`providerType`示し`providerType`ます。

サポートされる`providerType`値	関連付けられている`providerConfig`キー
AWS	S3
Azure	Azure
GCP	GCP
GenericS3	S3
OntapS3	S3
StorageGridS3	S3

AppVaultブラウザを使用してAppVault情報を表示する

Trident保護CLIプラグインを使用して、クラスタ上で作成されたAppVaultオブジェクトに関する情報を表示できます。

手順

1. AppVaultオブジェクトの内容を表示します。

```
tridentctl protect get appvaultcontent gcp-vault --show-resources all
```

出力例：

```
+-------------+-------+----------+----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+----------------------------
+---------------------------+
```

2. 必要に応じて、各リソースのAppVaultPathを表示するには、フラグを使用し `--show-paths`ます。

テーブルの最初の列に表示されるクラスタ名は、Trident protect helmのインストールでクラスタ名が指定されている場合にのみ使用できます。例： `--set clusterName=production1`。

AppVaultの削除

AppVaultオブジェクトはいつでも削除できます。

> AppVaultオブジェクトを削除する前に、AppVault CRのキーを削除しないで `finalizers`ください。これを行うと、AppVaultバケット内のデータが残り、クラスタ内のリソースが孤立する可能性があります。

作業を開始する前に

関連付けられているバケットに格納されているSnapshotとバックアップをすべて削除しておきます。

Kubernetes CLIを使用したAppVaultの削除

1. AppVaultオブジェクトを削除し、削除するAppVaultオブジェクトの名前に置き換え `appvault_name`ます。

```
kubectl delete appvault <appvault_name> -n trident-protect
```

Trident CLIを使用したAppVaultの削除

1. AppVaultオブジェクトを削除し、削除するAppVaultオブジェクトの名前に置き換え `appvault_name`ます。

```
tridentctl protect delete appvault <appvault_name> -n trident-protect
```

管理用アプリケーションの定義

Trident protectで管理するアプリケーションを定義するには、アプリケーションCRおよび関連するAppVault CRを作成します。

AppVault CRの作成

アプリケーションでデータ保護処理を実行するときに使用するAppVault CRを作成する必要があります。また、Trident保護がインストールされているクラスタにAppVault CRを配置する必要があります。AppVault CRはお使いの環境に固有です。AppVault CRSの例については、"AppVaultカスタムリソース。"

アプリケーションCRの作成

Trident保護を使用して管理するアプリケーションごとに、へのアプリケーションCRを作成する必要があります。管理用のアプリケーションを追加するには、アプリケーションCRを手動で作成するか、Trident保護CLIを使用してCRを作成します。

CRを使用したアプリケーションの追加

1. デスティネーションアプリケーションのCRファイルを作成します。

   a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `maria-app.yaml` ）。

   b. 次の属性を設定します。

      - * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。保護操作に必要な他のCRファイルがこの値を参照するため、選択した名前をメモします。
      - * spec.includedNamespaces*:(*required*)名前空間ラベルまたは名前空間名を使用して、アプリケーションリソースが存在する名前空間を指定します。アプリケーション名前空間は、このリストの一部である必要があります。

        YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: maria
  namespace: my-app-namespace
spec:
  includedNamespaces:
    labelSelector: {}
    namespace: my-app-namespace
```

CLIを使用したアプリケーションの追加

1. アプリケーション定義を作成して適用し、括弧内の値を環境からの情報に置き換えます。次の例に示す引数を持つカンマ区切りリストを使用して、名前空間とリソースをアプリケーション定義に含めることができます。

```
tridentctl protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include>
```

アプリケーションの保護

自動保護ポリシーまたはアドホックベースを使用して、スナップショットやバックアップを作成することで、すべてのアプリケーションを保護します。

オンデマンドスナップショットを作成します

オンデマンド Snapshot はいつでも作成できます。

CRを使用したスナップショットの作成

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.applicationRef *：スナップショットを作成するアプリケーションのKubernetes名。
   - * spec.appVaultRef *：（*required*）スナップショットの内容（メタデータ）を格納するAppVaultの名前。
   - * spec.reclaimPolicy *：（_Optional _）スナップショットCRが削除されたときのスナップショットのAppArchiveの動作を定義します。つまり、に設定しても `Retain`Snapshotは削除されます。有効なオプション：
     - `Retain`（デフォルト）
     - `Delete`

```
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  reclaimPolicy: Delete
```

3. ファイルに正しい値を入力したら `trident-protect-snapshot-cr.yaml`、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-cr.yaml
```

CLIを使用したスナップショットの作成

1. スナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot>
```

オンデマンドバックアップの作成

アプリはいつでもバックアップできます。

CRを使用したバックアップの作成

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.applicationRef * :（*required*）バックアップするアプリケーションのKubernetes名。
   - * spec.appVaultRef * :（*required*）バックアップ内容を格納するAppVaultの名前。
   - *spec.DataMover *:(*Optional*)バックアップ操作に使用するバックアップツールを示す文字列。有効な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia` （デフォルト）
   - * spec.reclaimPolicy * :（_Optional _）要求から解放されたバックアップの処理を定義します。有効な値：
     - `Delete`
     - `Retain` （デフォルト）
   - * Spec.snapshotRef * :（オプション）：バックアップのソースとして使用するSnapshotの名前。指定しない場合は、一時Snapshotが作成されてバックアップされます。

```
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. ファイルに正しい値を入力したら `trident-protect-backup-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-cr.yaml
```

CLIを使用したバックアップの作成

1. バックアップを作成します。角かっこ内の値は、使用している環境の情報に置き換えます。例：

```
tridentctl protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up>
```

データ保護スケジュールを作成

保護ポリシーは、定義されたスケジュールでスナップショット、バックアップ、またはその両方を作成することでアプリケーションを保護します。Snapshot とバックアップを毎時、日次、週次、および月単位で作成し、保持するコピーの数を指定できます。

CRを使用したスケジュールの作成

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-schedule-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.DataMover *:(*Optional*)バックアップ操作に使用するバックアップツールを示す文字列。有効な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia` （デフォルト）
   - * spec.applicationRef *：バックアップするアプリケーションのKubernetes名。
   - * spec.appVaultRef *：（*required*）バックアップ内容を格納するAppVaultの名前。
   - * spec.backupRetention *：保持するバックアップの数。ゼロは、バックアップを作成しないことを示します。
   - * spec.snapshotRetention *：保持するSnapshotの数。ゼロは、スナップショットを作成しないことを示します。
   - * spec.granularity*:スケジュールを実行する頻度。指定可能な値と必須の関連フィールドは次のとおりです。
     - `hourly` （を指定する必要があり `spec.minute`ます）
     - `daily` （とを指定する必要があり `spec.minute` `spec.hour`ます）。
     - `weekly`（および `spec.dayOfWeek`を指定する必要があり `spec.minute, spec.hour`ます）。
     - `monthly`（および `spec.dayOfMonth`を指定する必要があり `spec.minute, spec.hour`ます）。
   - * spec.dayOfMonth *：（_Optional _）スケジュールを実行する月の日（1〜31）。粒度がに設定されている場合、このフィールドは必須 `monthly`です。
   - * spec.DayOfWeek *：（_Optional _）スケジュールを実行する曜日（0 ~ 7）。0または7の値は日曜日を示します。粒度がに設定されている場合、このフィールドは必須 `weekly`です。
   - * spec.hour *：（_Optional _）スケジュールを実行する時刻（0 ~ 23）。粒度が、、またはに設定されている場合、このフィールドは必須 `daily` `weekly` `monthly`です。
   - * spec.minute *：（_Optional _）スケジュールを実行する分（0 ~ 59）。このフィールドは、粒度が、、、またはに設定されている場合は必須 `hourly` `daily` `weekly` `monthly`です。

```
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: <monthly>
  dayOfMonth: "1"
  dayOfWeek: "0"
  hour: "0"
  minute: "0"
```

3. ファイルに正しい値を入力したら `trident-protect-schedule-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

CLIを使用してスケジュールを作成する

1. 保護スケジュールを作成し、角かっこ内の値を環境からの情報に置き換えます。例：

> を使用すると、このコマンドの詳細なヘルプ情報を表示できます `tridentctl protect create schedule --help`。

```
tridentctl protect create schedule <my_schedule_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> --backup
-retention <how_many_backups_to_retain> --data-mover
<kopia_or_restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain>
```

Snapshot を削除します

不要になったスケジュール済みまたはオンデマンドの Snapshot を削除します。

手順

1. Snapshotに関連付けられているSnapshot CRを削除します。

```
kubectl delete snapshot <snapshot_name> -n my-app-namespace
```

バックアップを削除します

不要になったスケジュール済みまたはオンデマンドのバックアップを削除します。

手順

1. バックアップに関連付けられているバックアップCRを削除します。

```
kubectl delete backup <backup_name> -n my-app-namespace
```

バックアップ処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したバックアップ処理のステータスを確認できます。

手順

1. 次のコマンドを使用してバックアップ処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

azure-anf-files NetApp（ANF）処理のバックアップとリストアを実現

Trident protectをインストールしている場合はNetApp、Trident 24.06より前に作成されたazure-lun-filesストレージクラスを使用するストレージバックエンドに対して、スペース効率に優れたバックアップおよびリストア機能を有効にすることができます。この機能はNFSv4ボリュームで機能し、容量プールから追加のスペースを消費することはありません。

作業を開始する前に

次の点を確認します。

- Trident protectをインストールしておきます。
- Trident保護でアプリケーションを定義しました。この手順を完了するまで、このアプリケーションの保護機能は制限されます。
- ストレージバックエンドのデフォルトのストレージクラスとしてを選択しまし `azure-netapp-files` た。

構成手順用に展開

1. Trident 24.10にアップグレードする前にANFボリュームを作成した場合は、Tridentで次の手順を実行します。

   a. アプリケーションに関連付けられているNetAppファイルベースの各PVのSnapshotディレクトリを有効にします。

   ```
   tridentctl update volume <pv name> --snapshot-dir=true -n trident
   ```

   b. 関連付けられている各PVに対してSnapshotディレクトリが有効になっていることを確認します。

   ```
   tridentctl get volume <pv name> -n trident -o yaml | grep
   snapshotDir
   ```

   応答：

   ```
   snapshotDirectory: "true"
   ```

   +
   Snapshotディレクトリが有効になっていない場合、Trident保護は通常のバックアップ機能を選択します。この機能は、バックアッププロセス中に一時的に容量プールのスペースを消費します。この場合は、バックアップするボリュームと同じサイズの一時ボリュームを作成するための十分なスペースが容量プールに確保されていることを確認してください。

結果

これで、Trident保護を使用したアプリケーションのバックアップとリストアが可能になります。各PVCは、他のアプリケーションでバックアップおよびリストアに使用することもできます。

リストアアプリケーション

Trident保護を使用すると、Snapshotまたはバックアップからアプリケーションをリストアできます。同じクラスタにアプリケーションをリストアする場合、既存の Snapshot からのリストアは高速です。

> アプリケーションを復元すると、そのアプリケーションに設定されているすべての実行フックがアプリケーションとともに復元されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。

バックアップから別のネームスペースへのリストア

BackupRestore CRを使用して別のネームスペースにバックアップをリストアすると、Trident保護によってアプリケーションが新しいネームスペースにリストアされますが、リストアされたアプリケーションはTrident保

護によって自動的に保護されません。リストアされたアプリケーションを保護するには、Trident保護によって保護されるように、リストアされたアプリケーションのアプリケーションCRを作成する必要があります。

> 既存のリソースがある別のネームスペースにバックアップをリストアしても、バックアップ内のリソースと名前を共有するリソースは変更されません。バックアップ内のすべてのリソースをリストアするには、ターゲットネームスペースを削除して再作成するか、新しいネームスペースにバックアップをリストアします。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.appVaultRef * :（*required*）バックアップコンテンツが格納されているAppVaultの名前。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。
   - * spec.storageClassMapping * :リストア処理のソースストレージクラスからデスティネーションストレージクラスへのマッピング。および `sourceStorageClass`を、使用している環境の情報に置き換え `destinationStorageClass`ます。

     ```
     apiVersion: protect.trident.netapp.io/v1o
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。

- *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
- **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
- * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
- *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      group: my-resource-group
      kind: my-resource-kind
      version: my-resource-version
      names: ["my-resource-names"]
      namespaces: ["my-resource-namespaces"]
      labelSelectors: ["trident.netapp.io/os=linux"]
```

4. ファイルに正しい値を入力したら `trident-protect-backup-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

CLI を使用します

1. バックアップを別のネームスペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。 `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2`ます。例：

```
tridentctl protect create backuprestore <my_restore_name> --backup
<backup_namespace>/<backup_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

バックアップから元のネームスペースへのリストア

バックアップはいつでも元のネームスペースにリストアできます。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-ipr-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。

   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.appVaultRef *：（*required*）バックアップコンテンツが格納されているAppVaultの名前。

     例：

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appArchivePath: my-backup-path
  appVaultRef: appvault-name
```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。

   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
       - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
       - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
       - * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
       - *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
       - *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：

"trident.netapp.io/os=linux"。

例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      group: my-resource-group
      kind: my-resource-kind
      version: my-resource-version
      names: ["my-resource-names"]
      namespaces: ["my-resource-namespaces"]
      labelSelectors: ["trident.netapp.io/os=linux"]
```

4. ファイルに正しい値を入力したら `trident-protect-backup-ipr-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

CLI を使用します

1. バックアップを元のネームスペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。この `backup`引数では、という形式のネームスペースとバックアップ名を使用し `<namespace>/<name>`ます。例：

```
tridentctl protect create backupinplacerestore <my_restore_name>
--backup <namespace/backup_to_restore>
```

Snapshotから別のネームスペースへのリストア

カスタムリソース（CR）ファイルを使用して、スナップショットから別のネームスペースまたは元のソースネームスペースにデータをリストアできます。SnapshotRestore CRを使用して別のネームスペースにSnapshotをリストアすると、Trident保護によって新しいネームスペースにアプリケーションがリストアされますが、リストアされたアプリケーションはTrident保護によって自動的に保護されません。リストアされたアプリケーションを保護するには、Trident保護によって保護されるように、リストアされたアプリケーションのアプリケーションCRを作成する必要があります。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appVaultRef * :（*required*）スナップショットコンテンツが格納されているAppVaultの名前。
   - * spec.appArchivePath * :スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。
   - * spec.storageClassMapping * :リストア処理のソースストレージクラスからデスティネーションストレージクラスへのマッピング。および `sourceStorageClass`を、使用している環境の情報に置き換え `destinationStorageClass`ます。

     ```
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
       storageClassMapping:
         destination: "${destinationStorageClass}"
         source: "${sourceStorageClass}"
     ```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。

- *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
- **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
- * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
- *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      group: my-resource-group
      kind: my-resource-kind
      version: my-resource-version
      names: ["my-resource-names"]
      namespaces: ["my-resource-namespaces"]
      labelSelectors: ["trident.netapp.io/os=linux"]
```

4. ファイルに正しい値を入力したら `trident-protect-snapshot-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

CLI を使用します

1. スナップショットを別のネームスペースにリストアし、括弧内の値を環境の情報に置き換えます。

   ◦ `snapshot`引数では、という形式のネームスペースとSnapshot名を使用し `<namespace>/<name>`ます。

   ◦ `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2`ます。

   例：

```
tridentctl protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

Snapshotから元のネームスペースへのリストア

Snapshotはいつでも元のネームスペースにリストアできます。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-ipr-cr.yaml` ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appVaultRef * :（*required*）スナップショットコンテンツが格納されているAppVaultの名前。
   - * spec.appArchivePath * :スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

```
kubectl get snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appVaultRef: appvault-name
    appArchivePath: my-snapshot-path
```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
       - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
       - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
       - * resourceMatchers[].names * :（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
       - *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
       - *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      group: my-resource-group
      kind: my-resource-kind
      version: my-resource-version
      names: ["my-resource-names"]
      namespaces: ["my-resource-namespaces"]
      labelSelectors: ["trident.netapp.io/os=linux"]
```

4. ファイルに正しい値を入力したら `trident-protect-snapshot-ipr-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

CLI を使用します

1. Snapshotを元のネームスペースにリストアします。括弧内の値は、環境の情報に置き換えてください。例：

```
tridentctl protect create snapshotinplacerestore <my_restore_name>
--snapshot <snapshot_to_restore>
```

リストア処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したリストア処理のステータスを確認できます。

手順

1. 次のコマンドを使用してリストア処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

NetApp SnapMirrorでアプリケーションをレプリケート

Trident保護を使用すると、NetApp SnapMirrorテクノロジの非同期レプリケーション機能を使用して、ストレージバックエンド間、同じクラスタ上、または異なるクラスタ間でデータやアプリケーションの変更をレプリケートできます。

レプリケーション関係を設定

レプリケーション関係の設定には、次の作業が含まれます。

- Trident protectでアプリケーションスナップショットを作成する頻度を選択します（アプリケーションのKubernetesリソースと、アプリケーションの各ボリュームのボリュームスナップショットが含まれます）。
- レプリケーションスケジュールの選択（Kubernetesリソースと永続ボリュームデータを含む）
- Snapshotの作成時間の設定

手順

1. ソースクラスタにソースアプリケーション用のAppVaultを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します"AppVaultカスタムリソース"。

CRを使用した**AppVault**の作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-appvault-primary-source.yaml`）。

b. 次の属性を設定します。

- * metadata.name*:（*required*）AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
- * spec.providerConfig*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。bucketNameとプロバイダーに必要なその他の詳細を選択します。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"AppVaultカスタムリソース"。
- * spec.providerCredentials*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
  - * spec.providerCredentials.valueFromSecret*:（*required*）は、クレデンシャル値がシークレットから取得される必要があることを示します。
    - * key *：（*required*）選択するシークレットの有効なキー。
    - * name *：（*required*）このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
  - * spec.providerCredentials.secretAccessKey*:（*required*）プロバイダへのアクセスに使用するアクセスキー。name は spec.providerCredentials.valueFromSecret.name*と一致している必要があります。
- * spec.providerType*:（*required*）は、バックアップの提供元（NetApp ONTAP S3、汎用S3、Google Cloud、Microsoft Azureなど）を決定します。有効な値：
  - AWS
  - Azure
  - GCP
  - 汎用- s3
  - ONTAP - s3
  - StorageGRID - s3

c. ファイルに正しい値を入力したら `trident-protect-appvault-primary-source.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

CLIを使用した**AppVault**の作成

a. AppVaultを作成し、括弧内の値を環境からの情報に置き換えます。

```
tridentctl protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name>
```

2. ソースアプリケーションCRを作成します。

CRを使用したソースアプリケーションの作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-app-source.yaml`）。

b. 次の属性を設定します。

- * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
- * spec.includedNamespaces*:（*required*）名前空間と関連ラベルの配列。名前空間名を使用し、必要に応じてラベルを使用して名前空間のスコープを絞り込み、ここにリストされている名前空間に存在するリソースを指定します。アプリケーション名前空間は、この配列の一部である必要があります。
  - YAMLの例*：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: maria
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: maria
      labelSelector: {}
```

c. ファイルに正しい値を入力したら `trident-protect-app-source.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-namespace
```

CLIを使用したソースアプリケーションの作成

a. ソースアプリケーションを作成します。例：

```
tridentctl protect create app maria --namespaces maria -n my-app-namespace
```

3. 必要に応じて、ソースアプリケーションのスナップショットを作成します。このSnapshotは、デスティネーションクラスタのアプリケーションのベースとして使用されます。この手順を省略した場合は、スケジュールされた次のSnapshotが実行されて最新のSnapshotが作成されるまで待つ必要があります。

CRを使用したスナップショットの作成

a. ソースアプリケーションのレプリケーションスケジュールを作成します。

   i. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-schedule.yaml`）。

   ii. 次の属性を設定します。

      - * metadata.name*:（*required*）スケジュールカスタムリソースの名前。
      - *spec.AppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVault のmetadata.nameフィールドと一致する必要があります。
      - *spec.ApplicationRef *:(*required*)この値は、ソースアプリケーションCRのmetadata.name フィールドと一致する必要があります。
      - * spec.backupRetention *：（*required*）このフィールドは必須であり、値は0に設定する必要があります。
      - * spec.enabled *：trueに設定する必要があります。
      - * spec.granularity*:はに設定する必要があります `Custom`。
      - *spec.recurrenceRule *:開始日をUTC時間と繰り返し間隔で定義します。
      - * spec.snapshotRetention *：を2に設定する必要があります。

        YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule-0e1f88ab-f013-4bce-8ae9-6afed9df59a1
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-
46a3-420a-b351-45795e1b5e34
  applicationRef: maria
  backupRetention: "0"
  enabled: true
  granularity: custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

   i. ファイルに正しい値を入力したら `trident-protect-schedule.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-
namespace
```

CLIを使用したスナップショットの作成

a. スナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot>
```

4. ソースクラスタに適用したAppVault CRと同じソースアプリケーションAppVault CRをデスティネーションクラスタに作成し、という名前を付けます（例： `trident-protect-appvault-primary-destination.yaml`）。
5. CRを適用します。

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
my-app-namespace
```

6. デスティネーションクラスタにデスティネーションアプリケーション用のAppVaultを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します"AppVaultカスタムリソース"。
   a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-appvault-secondary-destination.yaml`）。
   b. 次の属性を設定します。
      - * metadata.name*:（*required*）AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
      - * spec.providerConfig*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。およびプロバイダに必要なその他の詳細情報を選択します `bucketName`。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"AppVaultカスタムリソース"。
      - * spec.providerCredentials*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
        - * spec.providerCredentials.valueFromSecret*:（*required*）は、クレデンシャル値がシークレットから取得される必要があることを示します。
          - * key * :（*required*）選択するシークレットの有効なキー。
          - * name * :（*required*）このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
        - * spec.providerCredentials.secretAccessKey*:（*required*）プロバイダへのアクセスに使用するアクセスキー。name は spec.providerCredentials.valueFromSecret.name*と一致している必要があります。

- * spec.providerType*:（*required*）は、バックアップの提供元（NetApp ONTAP S3、汎用S3、Google Cloud、Microsoft Azureなど）を決定します。有効な値：
  - AWS
  - Azure
  - GCP
  - 汎用- s3
  - ONTAP - s3
  - StorageGRID - s3

c. ファイルに正しい値を入力したら `trident-protect-appvault-secondary-destination.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n my-app-namespace
```

7. AppMirrorRelationship CRファイルを作成します。

CRを使用した**AppMirrorRelationship**の作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-relationship.yaml`）。

b. 次の属性を設定します。

- * metadata.name:*（必須）AppMirrorRelationshipカスタムリソースの名前。
- * spec.destinationAppVaultRef*:（*required*）この値は、デスティネーションクラスタ上のデスティネーションアプリケーションのAppVaultの名前と一致する必要があります。
- * spec.namespaceMapping*:(*required*)宛先およびソースの名前空間は、それぞれのアプリケーションCRで定義されているアプリケーション名前空間と一致している必要があります。
- *spec.sourceAppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVaultの名前と一致する必要があります。
- **spec.sourceApplicationName**:(*required*)この値は、ソースアプリケーションCRで定義したソースアプリケーションの名前と一致する必要があります。
- * spec.storageClassName *：（*required*）クラスタ上の有効なストレージクラスの名前を選択します。ストレージクラスは、ソースアプリケーションが導入されているソースクラスタで使用中のストレージクラスとピア関係にある必要があります。
- *spec.recurrenceRule *:開始日をUTC時間と繰り返し間隔で定義します。

  YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: maria
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. ファイルに正しい値を入力したら `trident-protect-relationship.yaml` 、CRを適用しま

す。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

CLIを使用した**AppMirrorRelationship**の作成

a. AppMirrorRelationshipオブジェクトを作成して適用し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --recurrence-rule <rule> --source-app
<my_source_app> --source-app-vault <my_source_app_vault>
```

8. （オプション）レプリケーション関係の状態とステータスを確認します。

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

デスティネーションクラスタへのフェイルオーバー

Trident保護を使用すると、レプリケートされたアプリケーションをデスティネーションクラスタにフェイルオーバーできます。この手順 はレプリケーション関係を停止し、デスティネーションクラスタでアプリケーションをオンラインにします。Trident protectが動作していた場合、ソースクラスタ上のアプリは停止しません。

手順

1. AppMirrorRelationship CRファイル（など）を開き `trident-protect-relationship.yaml`、* spec.desiredState*の値をに変更します `Promoted`。
2. CR ファイルを保存します。
3. CRを適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. （オプション）フェイルオーバーされたアプリケーションで必要な保護スケジュールを作成します。
5. （オプション）レプリケーション関係の状態とステータスを確認します。

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

フェイルオーバーされたレプリケーション関係を再同期します。

再同期処理によってレプリケーション関係が再確立されます。再同期処理を実行すると、元のソースアプリケーションが実行中のアプリケーションになり、デスティネーションクラスタで実行中のアプリケーションに加えた変更は破棄されます。

このプロセスは、レプリケーションを再確立する前に、デスティネーションクラスタ上のアプリケーションを停止します。

> フェイルオーバー中にデスティネーションアプリケーションに書き込まれたデータはすべて失われます。

手順

1. ソースアプリケーションのスナップショットを作成します。
2. AppMirrorRelationship CRファイル（など）を開き `trident-protect-relationship.yaml` 、spec.desiredStateの値をに変更します。 `Established`
3. CR ファイルを保存します。
4. CRを適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. フェイルオーバーされたアプリケーションを保護するためにデスティネーションクラスタで保護スケジュールを作成した場合は削除します。スケジュールが残っていると、ボリュームSnapshotが失敗します。

フェイルオーバーされたレプリケーション関係の逆再同期

フェイルオーバーされたレプリケーション関係を逆再同期すると、デスティネーションアプリケーションがソースアプリケーションになり、ソースがデスティネーションになります。フェイルオーバー中にデスティネーションアプリケーションに加えられた変更は保持されます。

手順

1. 元のデスティネーションクラスタでAppMirrorRelationship CRを削除します。これにより、デスティネーションがソースになります。新しいデスティネーションクラスタに保護スケジュールが残っている場合は削除します。
2. レプリケーション関係を設定するには、元 々 その関係を反対側のクラスタに設定するために使用したCRファイルを適用します。
3. 各クラスタでAppVault CRSの準備が完了していることを確認します。
4. 反対側のクラスタにレプリケーション関係を設定し、逆方向の値を設定します。

アプリケーションのレプリケーション方向を反転

レプリケーション方向を反転すると、Trident保護によってアプリケーションがデスティネーションストレージバックエンドに移動され、元のソースストレージバックエンドに引き続きレプリケートされます。Trident protectは、ソースアプリケーションを停止し、デスティネーションアプリケーションにフェイルオーバーする前にデータをデスティネーションにレプリケートします。

この状況では、ソースとデスティネーションを交換しようとしています。

手順

1. シャットダウンスナップショットを作成します。

CRを使用したシャットダウンスナップショットの作成

a. ソースアプリケーションの保護ポリシースケジュールを無効にします。

b. ShutdownSnapshot CRファイルを作成します。

i. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-shutdownsnapshot.yaml`）。

ii. 次の属性を設定します。

- * metadata.name*:（*required*）カスタムリソースの名前。
- *spec.AppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVaultのmetadata.nameフィールドと一致する必要があります。
- *spec.ApplicationRef *:(*required*)この値は、ソースアプリケーションCRファイルのmetadata.nameフィールドと一致する必要があります。

YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: maria
```

c. ファイルに正しい値を入力したら `trident-protect-shutdownsnapshot.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-namespace
```

CLIを使用したシャットダウンスナップショットの作成

a. シャットダウンスナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot>
```

2. Snapshotが完了したら、Snapshotのステータスを取得します。

```
kubectl get shutdownsnapshot -n my-app-namespace
<shutdown_snapshot_name> -o yaml
```

3. 次のコマンドを使用して* shutdownsnapshot.status.appArchivePath *の値を検索し、ファイルパスの最後の部分（basenameとも呼ばれます。これは最後のスラッシュのあとのすべてになります）を記録します。

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 次のように変更して、デスティネーションクラスタからソースクラスタへのフェイルオーバーを実行します。

> フェイルオーバー手順のステップ2では、AppMirrorRelationship CRファイルにフィールドを含め、 `spec.promotedSnapshot`その値を上記の手順3で記録したベースネームに設定します。

5. の逆再同期の手順を実行し[フェイルオーバーされたレプリケーション関係の逆再同期]ます。
6. 新しいソースクラスタで保護スケジュールを有効にします。

結果

リバースレプリケーションが実行されると、次の処理が実行されます。

- 元のソースアプリのKubernetesリソースのスナップショットが作成されます。
- 元のソースアプリケーションのポッドは、アプリケーションのKubernetesリソースを削除することで正常に停止されます（PVCとPVはそのまま維持されます）。
- ポッドがシャットダウンされると、アプリのボリュームのスナップショットが取得され、レプリケートされます。
- SnapMirror関係が解除され、デスティネーションボリュームが読み取り/書き込み可能な状態になります。
- アプリのKubernetesリソースは、元のソースアプリがシャットダウンされた後に複製されたボリュームデータを使用して、シャットダウン前のスナップショットから復元されます。
- 逆方向にレプリケーションが再確立されます。

アプリケーションを元のソースクラスタにフェイルバックします

Trident保護を使用すると、フェイルオーバー処理後に次の一連の処理を使用して「フェイルバック」を実現できます。このワークフローでは、元のレプリケーション方向を復元するために、Trident保護は、レプリケーション方向を反転する前に、アプリケーションの変更を元のソースアプリケーションに戻します（再同期）。

このプロセスは、デスティネーションへのフェイルオーバーが完了した関係から開始し、次の手順を実行します。

- フェイルオーバー状態から開始します。

- レプリケーション関係を逆再同期します。

> 通常の再同期操作は実行しないでください。フェイルオーバー中にデスティネーションクラスタに書き込まれたデータが破棄されます。

- レプリケーションの方向を逆にします。

手順

1. 手順を実行します[フェイルオーバーされたレプリケーション関係の逆再同期]。
2. 手順を実行します[アプリケーションのレプリケーション方向を反転]。

レプリケーション関係を削除する

レプリケーション関係はいつでも削除できます。アプリケーションレプリケーション関係を削除すると、2つの別 々 のアプリケーションが作成され、それらのアプリケーション間に関係がなくなります。

手順

1. AppMirrorRelationship CRを削除します。

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

アプリケーションの移行

バックアップデータまたはSnapshotデータを別のクラスタまたはストレージクラスにリストアすることで、クラスタ間またはストレージクラス間でアプリケーションを移行できます。

> アプリケーションを移行すると、そのアプリケーション用に構成されたすべての実行フックがアプリケーションとともに移行されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。

バックアップとリストアの処理

次のシナリオでバックアップとリストアの処理を実行するには、特定のバックアップとリストアのタスクを自動化します。

同じクラスタにクローニング

アプリケーションを同じクラスタにクローニングするには、Snapshotまたはバックアップを作成し、同じクラスタにデータをリストアします。

手順

1. 次のいずれかを実行します。
   a. "Snapshot を作成します"です。
   b. "バックアップを作成します"です。

2. Snapshotとバックアップのどちらを作成したかに応じて、同じクラスタで次のいずれかを実行します。
    a. "スナップショットからデータをリストア"です。
    b. "バックアップからデータをリストア"です。

別のクラスタにクローニング

アプリケーションを別のクラスタにクローニングする（クラスタ間クローニングを実行する）には、Snapshotまたはバックアップを作成し、データを別のクラスタにリストアします。デスティネーションクラスタにTrident protectがインストールされていることを確認します。

手順

1. 次のいずれかを実行します。
    a. "Snapshot を作成します"です。
    b. "バックアップを作成します"です。
2. バックアップまたはSnapshotを含むオブジェクトストレージバケットのAppVault CRがデスティネーションクラスタで設定されていることを確認します。
3. Snapshotとバックアップのどちらを作成したかに応じて、デスティネーションクラスタで次のいずれかを実行します。
    a. "スナップショットからデータをリストア"です。
    b. "バックアップからデータをリストア"です。

あるストレージクラスから別のストレージクラスへのアプリケーションの移行

スナップショットを別のデスティネーションストレージクラスにリストアすることで、あるストレージクラスから別のストレージクラスにアプリケーションを移行できます。

たとえば、次のようになります（リストアCRのシークレットを除く）。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    destination: "${destinationNamespace}"
    source: "${sourceNamespace}"
  storageClassMapping:
    destination: "${destinationStorageClass}"
    source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

CRを使用したスナップショットのリストア

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appArchivePath *：スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.appVaultRef *：（*required*）スナップショットコンテンツが格納されているAppVaultの名前。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     ```
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
       appArchivePath: my-snapshot-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。
   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
   - * resourceMatchers.group*:（_Optional _）フィルタリングするリソースのグループ。
   - *resourceMatchers.kind *:(_Optional _)フィルタリングするリソースの種類。
   - **resourceMatchers.version**:(*Optional*)フィルタリングするリソースのバージョン。
   - * resourceMatchers.names*:（_Optional _）フィルタリングするリソースのKubernetes metadata.nameフィールド内の名前。
   - * resourceMatchers.namespaces*:（_Optional _）フィルタリングするリソースのKubernetes metadata.nameフィールド内の名前空間。

- *resourceMatchers.labelSelectors *:(*optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      group: my-resource-group
      kind: my-resource-kind
      version: my-resource-version
      names: ["my-resource-names"]
      namespaces: ["my-resource-namespaces"]
      labelSelectors: ["trident.netapp.io/os=linux"]
```

4. ファイルに正しい値を入力したら `trident-protect-snapshot-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

CLIを使用したスナップショットのリストア

1. スナップショットを別のネームスペースにリストアし、括弧内の値を環境の情報に置き換えます。

   - `snapshot`引数では、という形式のネームスペースとSnapshot名を使用し`<namespace>/<name>`ます。
   - `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし`source1:dest1,source2:dest2`ます。

     例：

```
tridentctl protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

実行フックの管理

実行フックは、管理対象アプリケーションのデータ保護操作と組み合わせて実行するように構成できるカスタムアクションです。たとえば、データベースアプリケーションがある場合、実行フックを使用して、スナップショットの前にすべてのデータベーストランザクションを一時停止し、スナップショットの完了後にトランザクションを再開でき

ます。これにより、アプリケーションと整合性のある Snapshot を作成できます。

実行フックのタイプ

Trident PROTECTは、実行可能なタイミングに基づいて、次のタイプの実行フックをサポートします。

- Snapshot前
- Snapshot後
- バックアップ前
- バックアップ後
- リストア後のPOSTコマンドです
- フェイルオーバー後

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

1. 適用可能なカスタムプリオペレーション実行フックは、適切なコンテナで実行されます。カスタムのプリオペレーションフックは必要なだけ作成して実行できますが、操作前のこれらのフックの実行順序は保証も構成もされていません。
2. データ保護処理が実行されます。
3. 適用可能なカスタムポストオペレーション実行フックは、適切なコンテナで実行されます。必要な数のカスタムポストオペレーションフックを作成して実行できますが、操作後のこれらのフックの実行順序は保証されず、設定もできません。

同じ種類の実行フック（スナップショット前など）を複数作成する場合、これらのフックの実行順序は保証されません。ただし、異なるタイプのフックの実行順序は保証されています。たとえば'以下は'すべての異なるタイプのフックを持つ構成の実行順序です

1. スナップショット前フックが実行されます
2. スナップショット後フックが実行されます
3. 予備フックが実行されます
4. バックアップ後のフックが実行されます

上記の順序の例は、既存のSnapshotを使用しないバックアップを実行する場合にのみ該当します。

本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にしたら、作成されたSnapshotとバックアップをテストして整合性があることを確認します。これを行うには、アプリケーションを一時的なネームスペースにクローニングし、スナップショットまたはバックアップをリストアしてから、アプリケーションをテストします。

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

- 実行フックは、スクリプトを使用してアクションを実行する必要があります。多くの実行フックは、同じスクリプトを参照できます。
- Trident保護では、実行フックが使用するスクリプトを実行可能なシェルスクリプトの形式で記述する必要があります。
- スクリプトのサイズは96KBに制限されています。
- Trident保護では、実行フックの設定と一致基準を使用して、スナップショット、バックアップ、またはリストア処理に適用できるフックを決定します。

> ⓘ 実行フックは、実行中のアプリケーションの機能を低下させたり、完全に無効にしたりすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。実行フックが関連付けられている状態でバックアップまたはスナップショット操作を開始した後'キャンセルした場合でも'バックアップまたはスナップショット操作がすでに開始されていればフックは実行できますつまり、バックアップ後の実行フックで使用されるロジックは、バックアップが完了したとは見なされません。

実行フックフィルタ

アプリケーションの実行フックを追加または編集するときに、実行フックにフィルタを追加して、フックが一致するコンテナを管理できます。フィルタは、すべてのコンテナで同じコンテナイメージを使用し、各イメージを別の目的（Elasticsearchなど）に使用するアプリケーションに便利です。フィルタを使用すると、一部の同一コンテナで実行フックが実行されるシナリオを作成できます。1つの実行フックに対して複数のフィルタを作成すると、それらは論理AND演算子と結合されます。実行フックごとに最大10個のアクティブフィルタを使用できます。

実行フックに追加する各フィルタは、正規表現を使用してクラスタ内のコンテナを照合します。フックがコンテナと一致すると、そのコンテナに関連付けられたスクリプトがフックによって実行されます。フィルタの正規表現では、正規表現2（RE2）構文を使用します。この構文では、一致リストからコンテナを除外するフィルタの作成はサポートされていません。実行フックフィルタの正規表現に対してTrident保護がサポートする構文については、を参照してください "正規表現2（RE2）構文のサポート"。

> ⓘ リストアまたはクローン処理のあとに実行される実行フックにネームスペースフィルタを追加し、リストアまたはクローンのソースとデスティネーションが異なるネームスペースにある場合、ネームスペースフィルタはデスティネーションネームスペースにのみ適用されます。

実行フックの例

にアクセスして "NetApp Verda GitHubプロジェクト" 、Apache CassandraやElasticsearchなどの一般的なアプリケーションの実際の実行フックをダウンロードします。また、独自のカスタム実行フックを構築するための例やアイデアを得ることもできます。

実行フックの作成

Trident protectを使用して、アプリのカスタム実行フックを作成できます。実行フックを作成するには、Owner 、 Admin 、または Member のいずれかの権限が必要です。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-hook.yaml`ます。
2. Trident保護環境とクラスタ構成に合わせて、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.applicationRef *:(*required*)実行フックを実行するアプリケーションのKubernetes名。
   - *spec.stage *:(*required*)実行フックが実行されるアクションのステージを示す文字列。有効な値：
     - 前
     - 投稿
   - *spec.action *:(*required*)指定された実行フックフィルタが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値：
     - スナップショット
     - バックアップ
     - リストア
     - フェイルオーバー
   - *spec.enabled *:(*Optional*)この実行フックが有効か無効かを示します。指定しない場合、デフォルト値はtrueです。
   - **spec.hookSource**:(*required*) base64でエンコードされたフックスクリプトを含む文字列。
   - *spec.timeout *:(_Optional _)実行フックの実行を許可する時間を分単位で定義する数値。最小値は1分で、指定しない場合のデフォルト値は25分です。
   - * spec.arguments *:(_Optional _)実行フックに指定できる引数のYAMLリスト。
   - *spec.matchingCriteria *:(*Optional*)実行フックフィルタを構成する各ペアの基準キー値ペアのオプションリスト。実行フックごとに最大10個のフィルタを追加できます。
   - *spec.matchingCriteria.type *:(*Optional*)実行フックフィルタタイプを識別する文字列。有効な値：
     - コンテナイメージ
     - コンテナ名
     - ポッド名
     - PodLabel
     - ネームスペース名
   - *spec.matchingCriteria.value *:(*Optional*)実行フックフィルタ値を識別する文字列または正規表現。

     YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. CRファイルに正しい値を入力したら、CRを適用します。

```
kubectl apply -f trident-protect-hook.yaml
```

CLI を使用します

1. 実行フックを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file>
```

Tridentプロテクトのアンインストール

製品の試用版からフルバージョンにアップグレードする場合は、Trident保護コンポーネントの削除が必要になることがあります。

Trident保護を削除するには、次の手順を実行します。

手順

1. Trident保護CRファイルを削除します。

```
helm uninstall trident-protect-crds
```

2. Trident保護を削除します。

```
helm uninstall -n trident-protect trident-protect
```

3. Trident保護ネームスペースを削除します。

```
kubectl delete ns trident-protect
```

知識とサポート

よくある質問

ここでは、Tridentのインストール、設定、アップグレード、トラブルシューティングに関するFAQを紹介します。

一般的な質問

Tridentはどのくらいの頻度でリリースされますか。

24.02リリース以降、Tridentは2月、6月、10月の4カ月ごとにリリースされます。

Tridentは、特定のバージョンの**Kubernetes**でリリースされたすべての機能をサポートしていますか。

Tridentは通常、Kubernetesのアルファ機能をサポートしていません。Trident は、 Kubernetes ベータリリースに続く 2 つの Trident リリースでベータ機能をサポートしています。

Tridentの機能に関して、他の**NetApp**製品との依存関係はありますか。

Tridentは、他のNetAppソフトウェア製品との依存関係はなく、スタンドアロンアプリケーションとして機能します。ただし、ネットアップのバックエンドストレージデバイスが必要です。

Trident構成の完全な詳細情報を取得するにはどうすればよいですか。

Trident構成に関する詳細情報を取得するには、コマンドを使用し `tridentctl get`ます。

Tridentでのストレージのプロビジョニング方法に関する指標は取得できますか。

はい。管理対象のバックエンド数、プロビジョニングされたボリューム数、消費されたバイト数など、Trident処理に関する情報を収集するために使用できるPrometheusエンドポイント。を監視および分析に使用することもできます"Cloud Insights の機能です"。

Tridentを**CSI**プロビジョニングツールとして使用すると、ユーザエクスペリエンスは変化しますか？

いいえ。ユーザーエクスペリエンスと機能に関しては変更はありません。使用されるプロビジョニングツール名はです `csi.trident.netapp.io`。現在および将来のリリースで提供されるすべての新機能を使用する場合は、この方法でTridentをインストールすることを推奨します。

Kubernetesクラスタへの**Trident**のインストールと使用

Tridentはプライベートレジストリからのオフラインインストールをサポートしていますか。

はい、Tridentはオフラインでインストールできます。を参照してください "Tridentのインストールについて"。

Tridentをリモートでインストールできますか。

はい。Trident 18.10以降では、クラスタにアクセスできる任意のマシンからのリモートインストール機能がサポートされます `kubectl`。アクセスが確認されたら `kubectl`（たとえば、リモートマシンからコマンドを実行し `kubectl get nodes`て確認する）、インストール手順に従います。

Tridentでハイアベイラビリティを構成できますか。

Tridentはインスタンスが1つのKubernetesデプロイメント（ReplicaSet）としてインストールされるため、HAが組み込まれています。デプロイメント内のレプリカの数を増やすことはできません。Tridentがインストールされているノードが失われた場合やポッドにアクセスできない場合、Kubernetesはポッドをクラスタ内の正常なノードに自動的に再導入します。Tridentはコントロールプレーンのみであるため、現在マウントされているポッドはTridentを再導入しても影響を受けません。

Tridentは**kube-system**ネームスペースにアクセスする必要がありますか。

TridentはKubernetes APIサーバから読み取り、アプリケーションが新しいPVCを要求するタイミングを判断するため、kube-systemへのアクセスが必要になります。

Tridentで使用されるロールと**Privileges**を教えてください。

TridentインストーラによってKubernetes ClusterRoleが作成され、KubernetesクラスタのPersistentVolume、PersistentVolumeClaim、StorageClass、およびSecretリソースに特定のアクセス権が付与されます。を参照してください "tridentctlのインストールをカスタマイズします"。

Tridentがインストールに使用するマニフェストファイルをローカルで生成できますか。

必要に応じて、Tridentがインストールに使用するマニフェストファイルをローカルで生成および変更できます。を参照してください "tridentctlのインストールをカスタマイズします"。

2つの独立した**Kubernetes**クラスタで、**2**つの独立した**Trident**インスタンスで同じ**ONTAP**バックエンド**SVM**を共有できますか。

推奨されませんが、2つのTridentインスタンスに同じバックエンドSVMを使用できます。インストール時に各インスタンスに一意のボリューム名を指定するか、ファイルで一意のパラメータを `setup/backend.json`指定し `StoragePrefix`ます。これは、両方のインスタンスで同じ FlexVol を使用しないためです。

Tridentを**ContainerLinux(**旧**CoreOS)**にインストールすることはできますか**?**

Tridentは単なるKubernetesポッドであり、Kubernetesが実行されている場所にインストールできます。

NetApp Cloud Volumes ONTAPで**Trident**を使用できますか。

はい。Tridentは、AWS、Google Cloud、Azureでサポートされています。

Tridentは**Cloud Volumes Services**と連携しますか。

はい。Tridentは、AzureのAzure NetApp FilesサービスとGCPのCloud Volumes Serviceをサポートしています。

トラブルシューティングとサポート

NetAppは**Trident**をサポートしていますか。

Tridentはオープンソースで無料で提供されていますが、NetAppバックエンドがサポートされていれば、NetAppは完全にサポートしています。

サポートケースを作成するにはどうすればよいですか？

サポートケースを作成するには、次のいずれかを実行します。

1. サポートアカウントマネージャーに連絡して、チケットの発行に関するサポートを受けてください。
2. 連絡してサポートケースを作成します "ネットアップサポート"。

サポートログバンドルを生成するにはどうすればよいですか？

tridentctl logs-a を実行して ' サポートバンドルを作成できますバンドルでキャプチャされたログに加えて、kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

問題を作成し "Trident Github の利用"、問題の件名と説明に*RFE*を記載します。

不具合を発生させる場所

で問題を作成し "Trident Github の利用"ます。問題に関連する必要なすべての情報とログを記録しておいてください。

Tridentに関する簡単な質問があり、説明が必要な場合はどうなりますか？コミュニティやフォーラムはありますか？

ご質問、問題、ご要望がございましたら、TridentまたはGitHubからお問い合わせ"チャネルを外します"ください。

ストレージシステムのパスワードが変更され、**Trident**が機能しなくなりました。どうすれば回復できますか？

バックエンドのパスワードを `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。交換してください `myBackend` この例では、バックエンド名にとを指定しています `/path/to_new_backend.json` と入力します `backend.json` ファイル。

Tridentで**Kubernetes**ノードが見つかりません。この問題を解決するにはどうすればよいですか

TridentがKubernetesノードを検出できない可能性があるシナリオは2つあります。Kubernetes または DNS 問題内のネットワーク問題が原因の場合もあります。各 Kubernetes ノードで実行される Trident ノードのデーモンが Trident コントローラと通信し、 Trident にノードを登録できる必要があります。この問題は、Tridentのインストール後にネットワークの変更が発生した場合、クラスタに追加された新しいKubernetesノードでのみ発生します。

Trident ポッドが破損すると、データは失われますか？

Trident ポッドが削除されても、データは失われません。TridentのメタデータはCRDオブジェクトに格納されます。Trident によってプロビジョニングされた PVS はすべて正常に機能します。

Tridentのアップグレード

古いバージョンから新しいバージョンに直接アップグレードできますか（いくつかのバージョンはスキップします）？

NetAppでは、Tridentをあるメジャーリリースから次のメジャーリリースにアップグレードできます。バージョン 18.xx から 19.xx 、 19.xx から 20.xx にアップグレードできます。本番環境の導入前に、ラボでアップグレードをテストする必要があります。

Trident を以前のリリースにダウングレードできますか。

アップグレード、依存関係の問題、またはアップグレードの失敗または不完全な実行後に見つかったバグの修正が必要な場合は、そのバージョンに固有の手順を使用して以前のバージョンを再インストールする必要があります"Tridentのアンインストール"。これは、以前のバージョンにダウングレードするための唯一の推奨方法です。

バックエンドとボリュームを管理

ONTAP バックエンド定義ファイルに管理 **LIF** とデータ **LIF** の両方を定義する必要がありますか。

管理LIFは必須です。データLIFのタイプはさまざまです。

- ONTAP SAN：iSCSIには指定しないでください。Tridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCI LIFを検出します。が明示的に定義されている場合は、警告が生成され `dataLIF`ます。詳細については、を参照してください "ONTAP のSAN構成オプションと例" 。
- ONTAP NAS:を指定することを推奨します `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。詳細は、を参照してください。"ONTAP NASの設定オプションと例"

Tridentは**ONTAP**バックエンド用に**CHAP**を構成できますか。

はい。Tridentは、ONTAPバックエンドに対して双方向CHAPをサポートしています。これには、バックエンド構成での設定が必要です `useCHAP=true`。

Tridentを使用してエクスポートポリシーを管理するにはどうすればよいですか。

Tridentでは、バージョン20.04以降でエクスポートポリシーを動的に作成および管理できます。これにより、ストレージ管理者はバックエンド構成に 1 つ以上の CIDR ブロックを指定でき、 Trident では、その範囲に含まれるノード IP を作成したエクスポートポリシーに追加できます。このようにして、Tridentは、所定のCIDR内にIPを持つノードのルールの追加と削除を自動的に管理します。

管理 **LIF** とデータ **LIF** に **IPv6** アドレスを使用できますか。

Tridentは次のIPv6アドレスの定義をサポートします

- `managementLIF` および `dataLIF` ONTAP NASバックエンドの場合：
- `managementLIF` ONTAP SANバックエンドの場合：を指定することはできません `dataLIF` ONTAP SANバックエンドの場合：

TridentをIPv6で機能させるには、フラグ（インストール用 `tridentctl`）、（Tridentオペレータ用）、`IPv6`または（Helmインストール用） `tridentTPv6`を使用してインストールする必要があります `--use-ipv6`。

バックエンドの管理 **LIF** を更新できますか。

はい。 tridentctl update backend コマンドを使用してバックエンド管理 LIF を更新できます。

バックエンドのデータ **LIF** を更新できるか。

のデータLIFを更新できます `ontap-nas` および `ontap-nas-economy` のみ。

Trident for Kubernetesで複数のバックエンドを作成できますか。

Tridentは、同じドライバでも異なるドライバでも、多数のバックエンドを同時にサポートできます。

Tridentはバックエンドクレデンシャルをどのように保存しますか。

Tridentは、バックエンドクレデンシャルをKubernetesシークレットとして保存します。

Tridentはどのようにして特定のバックエンドを選択しますか。

バックエンド属性を使用してクラスに適切なプールを自動的に選択できない場合は 'toragePools' パラメータと additionalStoragePools' パラメータを使用して ' 特定のプールセットを選択します

Tridentが特定のバックエンドからプロビジョニングされないようにするにはどうすればよいですか。

パラメータを `excludeStoragePools`使用して、Tridentがプロビジョニングに使用する一連のプールをフィルタリングし、に一致するプールをすべて削除します。

同じ種類のバックエンドが複数ある場合、**Trident**はどのようにして使用するバックエンドを選択しますか。

同じタイプの設定済みバックエンドが複数ある場合、Tridentはおよび `PersistentVolumeClaim`のパラメータに基づいて適切なバックエンドを選択します `StorageClass`。たとえば、ONTAP - NASドライバのバックエンドが複数ある場合、Tridentは、およびの `PersistentVolumeClaim`パラメータを照合し、および `PersistentVolumeClaim`に記載されている要件を提供できるバックエンドを `StorageClass`照合し `StorageClass`ます。要求に一致するバックエンドが複数ある場合、Tridentはそのうちの1つをランダムに選択します。

Tridentは**Element / SolidFire**で双方向**CHAP**をサポートしていますか。

はい。

Tridentでは、どのようにして**ONTAP**ボリュームに**qtree**を導入しますか。**1** つのボリュームに配置できる **qtree** の数はいくつですか。

「 ONTAP-NAS-エコノミー 」ドライバは、同一の FlexVol （ 50 ～ 300 の範囲で設定可能）で最大 200 個の

qtree を作成し、クラスタ・ノードあたり 100,000 個の qtree を作成し、クラスタあたり 240 万個を作成します。エコノミー・ドライバーがサービスを提供する新しい「 PersistentVolumeClaim 」を入力すると、ドライバーは新しい qtree にサービスを提供できる FlexVol がすでに存在するかどうかを確認します。qtree を提供できる FlexVol が存在しない場合は、新しい FlexVol が作成されます。

ONTAP NAS でプロビジョニングされたボリュームに **UNIX** アクセス権を設定するにはどうすればよいですか。

Tridentによってプロビジョニングされるボリュームに対してUNIX権限を設定するには、バックエンド定義ファイルにパラメータを設定します。

ボリュームをプロビジョニングする際に、明示的な **ONTAP NFS** マウントオプションを設定するにはどうすればよいですか。

Tridentでは、Kubernetesではデフォルトでマウントオプションがどの値にも設定されません。Kubernetesストレージクラスでマウントオプションを指定するには、次の例を参照して"こちらをご覧ください"ください。

プロビジョニングしたボリュームを特定のエクスポートポリシーに設定するにはどうすればよいですか？

適切なホストにボリュームへのアクセスを許可するには、バックエンド定義ファイルに設定されている「 exportPolicy 」パラメータを使用します。

ONTAPを使用した**Trident**によるボリューム暗号化の設定方法を教えてください。

Trident によってプロビジョニングされたボリュームで暗号化を設定するには、バックエンド定義ファイルの暗号化パラメータを使用します。詳細については、以下を参照してください。"TridentとNVEおよびNAEとの連携"

Tridentを使用して**ONTAP**の**QoS**を実装する最良の方法はどれですか。

ONTAP の QoS を実装するには、「 torageClasses 」を使用します。

Tridentでシンプロビジョニングまたはシックプロビジョニングを指定するにはどうすればよいですか。

ONTAP ドライバは、シンプロビジョニングまたはシックプロビジョニングをサポートします。ONTAP ドライバはデフォルトでシンプロビジョニングに設定されています。シックプロビジョニングが必要な場合は、バックエンド定義ファイルまたは「 torageClass 」を設定する必要があります。両方が設定されている場合は、「 torageClass 」が優先されます。ONTAP で次の項目を設定します。

1. 'S torageClass' で 'provisioningType' 属性を thick に設定します
2. バックエンド定義ファイルで 'backend spaceReserve パラメータを volume に設定して ' シックボリュームを有効にします

誤って **PVC** を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes では、バージョン 1.10 以降、 PVC 保護が自動的に有効になります。

Tridentで作成された**NFS PVC**を拡張できますか。

はい。Tridentによって作成されたPVCを拡張できます。ボリュームの自動拡張は ONTAP の機能であり、Trident には適用されません。

ボリュームが **SnapMirror** データ保護（ **DP** ）モードまたはオフラインモードの間にインポートできますか。

外部ボリュームが DP モードになっているかオフラインになっている場合、ボリュームのインポートは失敗します。次のエラーメッセージが表示されます。

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

リソースクォータをネットアップクラスタに変換する方法

Kubernetes ストレージリソースクォータは、ネットアップストレージの容量があるかぎり機能します。容量不足が原因でNetAppストレージがKubernetesクォータ設定に対応できない場合、Tridentはプロビジョニングを試行しますがエラーが発生します。

Tridentを使用してボリューム**Snapshot**を作成できますか。

はい。Tridentでは、オンデマンドのボリュームSnapshotとSnapshotからの永続的ボリュームの作成がサポートされています。スナップショットからPVSを作成するには、フィーチャーゲートが有効になっていることを確認し `VolumeSnapshotDataSource`ます。

Tridentボリュームスナップショットをサポートするドライバを教えてください。

現在のところ ' オンデマンドスナップショットのサポートは 'ONTAP-NAS'`ONTAP-NAS-flexgroup 'ONTAP-SAN’ONTAP-SANエコノミー "solidfire-san-SAN""solidfire-san-""solidfire-san-'"solidfire-san-"" で利用できます「 gcp-cvs`」と「 azure-NetApp-files 」バックエンドドライバ。

Tridentで**ONTAP**を使用してプロビジョニングされたボリュームの**Snapshot**バックアップを作成する方法を教えてください。

これは 'ONTAP-NAS' 'ONTAP-SAN' および 'ONTAP-NAS-flexgroup ドライバで利用できますFlexVol レベルでは「 ONTAP-SAN-エコノミー 」ドライバに「スナップショットポリシー」を指定することもできます。

これはドライバでも使用できますが、qtreeレベルではなく、FlexVolレベルで使用でき `ontap-nas-economy`ます。Tridentでプロビジョニングされたボリュームの Snapshotを作成できるようにするには、backendパラメータオプションを、ONTAPバックエンドで定義されている目的のSnapshotポリシーに設定し `snapshotPolicy`ます。ストレージコントローラで作成されたSnapshotは、Tridentでは認識されません。

Tridentでプロビジョニングされたボリュームに**Snapshot**リザーブの割合を設定できますか。

はい。バックエンド定義ファイルで属性を設定することで、Tridentを使用してSnapshotコピーを格納するために特定の割合のディスクスペースをリザーブできます `snapshotReserve`。を設定し、`snapshotReserve`バックエンド定義ファイルでスナップショット予約の割合が設定されている場合は `snapshotPolicy`、バックエンドファイルで指定されている割合に従って設定され `snapshotReserve`ます。パーセンテージ番号が指定されていない場合 `snapshotReserve`、ONTAPはデフォルトでスナップショット予約のパーセンテージを5とします。この `snapshotPolicy`オプションをnoneに設定すると、Snapshotリザーブの割合は0に設定されます。

ボリュームの **Snapshot** ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。バックエンド定義ファイルで「 snapmirror directionDir 」パラメータを設定することで、 Trident によってプロビジョニングされたボリューム上のスナップショットディレクトリにアクセスできます。

Tridentを使用してボリューム用に**SnapMirror**を設定できますか。

現時点では、 SnapMirror は ONTAP CLI または OnCommand System Manager を使用して外部に設定する必要があります。

永続ボリュームを特定の **ONTAP Snapshot** にリストアするにはどうすればよいですか？

ボリュームを ONTAP Snapshot にリストアするには、次の手順を実行します。

1. 永続ボリュームを使用しているアプリケーションポッドを休止します。
2. ONTAP CLI または OnCommand システムマネージャを使用して、必要な Snapshot にリバートします。
3. アプリケーションポッドを再起動します。

Tridentは、負荷共有ミラーが設定されている**SVM**でボリュームをプロビジョニングできますか。

負荷共有ミラーは、NFS経由でデータを提供するSVMのルートボリューム用に作成できます。ONTAP は、Tridentによって作成されたボリュームの負荷共有ミラーを自動的に更新します。ボリュームのマウントが遅延する可能性があります。Tridentを使用して複数のボリュームを作成する場合、ボリュームをプロビジョニングする方法は、負荷共有ミラーを更新するONTAP によって異なります。

お客様 / テナントごとにストレージクラスの使用状況を分離するにはどうすればよいですか。

Kubernetes では、ネームスペース内のストレージクラスは使用できません。ただし、 Kubernetes を使用すると、ネームスペースごとにストレージリソースクォータを使用することで、ネームスペースごとに特定のストレージクラスの使用量を制限できます。特定のストレージへのネームスペースアクセスを拒否するには、そのストレージクラスのリソースクォータを 0 に設定します。

トラブルシューティング

Tridentのインストールおよび使用中に発生する可能性のある問題のトラブルシューティングには、ここに示すポインタを使用してください。

全般的なトラブルシューティング

- Trident ポッドが正常に起動しない場合（たとえば、 Trident ポッドが 2 つ未満の「 ContainerCreating 」フェーズで停止した場合）、「 kubectl-n trident 」を実行して、展開が trident 、「 kubectl-n trident 」が pod trident- を記述します -** 追加のインサイトを提供できます。kubelet ログの取得 (journalctl -xeu kubelet など) も役立ちます。
- Trident ログに十分な情報がない場合は、インストールオプションに基づいてインストールパラメータに「 -d 」フラグを渡して、 Trident のデバッグモードを有効にしてみてください。

  次に './tridentctl logs -n trident` を使用して debug が設定され ' ログ内で 'level=debug msg' を検索していることを確認します

オペレータとともにインストールされます

```
kubectl patch torc trident -n <namespace> --type=merge -p
'{"spec":{"debug":true}}'
```

すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには 'kubectl get pod -n trident' の出力の 'age' 列を確認します

Trident 20.07および20.10では、の代わりに `torc`使用して `tprov`ください。

Helm とともにインストールされます

```
helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
tridentDebug=true`
```

tridentctl を使用してインストールされます

```
./tridentctl uninstall -n trident
./tridentctl install -d -n trident
```

- バックエンド定義に「 debugTraceFlags 」を含めると、バックエンドごとにデバッグログを取得することもできます。たとえば、 Trident ログで API 呼び出しとメソッドの逆数を取得するには、「 `debugTraceFlags: {"API":true,"method" :true,}` を指定します。既存のバックエンドには 'tridentctl backend update で構成された 'ebugTraceFlags' を設定できます
- RedHat CoreOS を使用する場合は 'iscsid がワーカー・ノードで有効になっており ' デフォルトで起動されていることを確認しますこの設定には、 OpenShift MachineConfig を使用するか、イグニッションテンプレートを変更します。
- Trident をで使用する際によく発生する問題です "Azure NetApp Files の特長" テナントとクライアントのシークレットが、必要な権限がないアプリケーションの登録から取得された場合です。Tridentの要件の一覧については、 "Azure NetApp Files の特長" 設定
- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl status rpcbind' またはそれに相当する処理を実行することで確認できます
- Trident バックエンドが、以前に作業したことがあるにもかかわらず「 failed 」状態であると報告した場合は、バックエンドに関連付けられている SVM/admin クレデンシャルの変更が原因である可能性があります。「 tridentctl update backend 」または Trident ポッドのバウンスを使用してバックエンド情報を更新すると、この問題は修正されます。
- Docker をコンテナランタイムとして Trident をインストールするときに権限の問題が発生した場合は、「 --in cluster=false」 フラグを付けて Trident のインストールを試みてください。これはインストーラポッドを使用せず、「 trident-installer 」ユーザのために発生する許可の問題を回避します。
- 実行に失敗した後のクリーンアップには 'uninstall パラメータ <Uninstalling Trident > を使用しますデフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全にアンインストールしてインストールできます。
- 以前のバージョンのTridentにダウングレードする場合は、 `tridentctl uninstall` Tridentを削除するコマンド。必要なをダウンロードします "Trident のバージョン" を使用してをインストールします

`tridentctl install` コマンドを実行します

- インストールが成功した後、 PVC が「保留中」段階で停止した場合、「 kubectl 」を実行して PVC を記述すると、 Trident がこの PVC の PV のプロビジョニングに失敗した理由を追加情報に提供できます。

オペレータを使用した**Trident**の導入に失敗

オペレータを使用して Trident を導入する場合 'TridentOrchestrator のステータスは 'Installing から Installed に変わります'Failed' ステータスが表示され ' オペレータがそれ自体で回復できない場合は ' 次のコマンドを実行してオペレータのログを確認する必要があります

```
tridentctl logs -l trident-operator
```

trident-operator コンテナのログの末尾には、問題のある場所を示すことができます。たとえば、このような問題の 1 つは、エアーギャップ環境のアップストリームレジストリから必要なコンテナイメージをプルできないことです。

Trident のインストールが失敗した理由を理解するには、「 TridentOrchestrator 」のステータスを確認する必要があります。

```
kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                          Message
  ----     ------  ----               ----                          -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、 Trident のインストールに使用された「 TridentOrchestrator 」がすでに存在することを示します。各 Kubernetes クラスタは Trident のインスタンスを 1 つしか保持できないため、オペレータは任意の時点で作成可能なアクティブな TridentOrchestrator が 1 つだけ存在することを確認します。

また、 Trident ポッドのステータスを確認することで、適切でないものがあるかどうかを確認できます。

```
kubectl get pods -n trident

NAME                                READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                   1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw        4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                   1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                   1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv   1/1     Running            0
8m17s
```

1 つ以上のコンテナイメージがフェッチされなかったため、ポッドが完全に初期化できないことがわかります。

この問題に対処するには、「 TridentOrchestrator 」 CR を編集する必要があります。また、「 TridentOrchestrator 」を削除して、変更された正確な定義を持つ新しいものを作成することもできます。

Tridentの導入に失敗しました `tridentctl`

何が問題になったかを特定するために、インストーラをもう一度「 -d`` 」引数を使用して実行すると、デバッグモードが有効になり、問題の内容を理解するのに役立ちます。

```
./tridentctl install -n trident -d
```

問題を解決した後 ' 次のようにインストールをクリーンアップし 'tridentctl install コマンドを再度実行できます

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```

TridentとCRDを完全に取り外します。

Trident、作成されたCRD、および関連するカスタムリソースをすべて完全に削除できます。

この操作は元に戻すことはできません。Tridentを完全に新規にインストールする場合を除き、この操作は行わないでください。CRDを削除せずにTridentをアンインストールするには、を参照してください"Trident をアンインストールします"。

Trident オペレータ

Tridentオペレータを使用してTridentをアンインストールし、CRDを完全に削除するには、次の手順に従います。

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

Helm

Helmを使用してTridentをアンインストールし、CRDを完全に削除するには：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>tridentctl</code>

を使用してTridentをアンインストールした後にCRDを完全に削除するには `tridentctl`

```
tridentctl obliviate crd
```

RWX rawブロックネームスペースo Kubernetes 1.26でNVMeノードのステージング解除が失敗する

Kubernetes 1.26を実行している場合、RWX rawブロックネームスペースでNVMe/TCPを使用すると、ノードのステージング解除が失敗することがあります。次のシナリオは、障害に対する回避策を提供します。または、Kubernetesを1.27にアップグレードすることもできます。

ネームスペースとポッドが削除されました

Tridentで管理されるネームスペース（NVMeの永続的ボリューム）がポッドに接続されているシナリオを考えてみましょう。ネームスペースをONTAPバックエンドから直接削除すると、ポッドを削除しようとすると、ステージング解除プロセスが停止します。このシナリオは、Kubernetesクラスタやその他の機能には影響しません。

回避策

該当するノードから永続的ボリューム（そのネームスペースに対応するボリューム）をアンマウントして削除します。

ブロックされたデータ**LIF**

```
 If you block (or bring down) all the dataLIFs of the NVMe Trident
backend, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```
.回避策
すべての機能を復元するには、`dataLIFS`を起動します。

ネームスペースマッピングが削除され

```
 If you remove the `hostNQN` of the worker node from the corresponding
subsystem, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```
.回避策
を追加します `hostNQN` サブシステムに戻ります。

サポート

ネットアップでは、さまざまな方法で Trident をサポートしています。ナレッジベース（KB）記事やDiscordチャネルなど、24時間365日利用可能な無料のセルフサポートオプションをご用意しています。

Tridentのサポートライフサイクル

Tridentでは、バージョンに応じて3つのレベルのサポートを提供しています。を参照してください "定義に対するNetAppソフトウェアバージョンのサポート"。

フルサポート
: Tridentは、リリース日から12か月間フルサポートを提供します。

限定サポート
: Tridentでは、リリース日から13~24カ月目に限定的なサポートを提供しています。

セルフサポート
: Tridentのマニュアルは、リリース日から25～36カ月目に入手できます。

バージョン	フルサポート	限定サポート	セルフサポート
"24.10"	2025年10月	2026年10月	2027年10月
"24.06"	2025年6月	2026年6月	2027年6月
"24.02"	2025年2月	2026年2月	2027年2月

バージョン	フルサポート	限定サポート	セルフサポート
"23.10"	2024年10月	2025年10月	2026年10月
"23.07"	2024年7月	2025年7月	2026年7月
"23.04"	—	2025年4月	2026年4月
"23.01"	—	2025年1月	2026年1月
"22.10"	—	2024年10月	2025年10月
"22.07"	—	2024年7月	2025年7月
"22.04"	—	—	2025年4月
"22.01"	—	—	2025年1月

セルフサポート

トラブルシューティング記事の包括的なリストについては、を参照してください "ネットアップナレッジベース（ログインが必要）"。

コミュニティサポート

コンテナユーザ（Trident開発者を含む）の活発なパブリックコミュニティがネットアップにあります"チャネルを外します"。プロジェクトに関する一般的な質問をしたり、同じような気のある同僚と関連するトピックについて話し合うのには、この場所が最適です。

NetAppテクニカルサポート

Tridentのサポートが必要な場合は、を使用してサポートバンドルを作成し `tridentctl logs -a -n trident`、に送信してください `NetApp Support <Getting Help>`。

を参照してください。

- "Tridentのリソース"
- "Kubernetes ハブ"

参照

Tridentポート

Tridentが通信に使用するポートの詳細については、こちらを参照してください。

Tridentポート

Tridentは次のポートを介して通信します。

ポート	目的
8443	バックチャネル HTTPS
8001	Prometheus 指標エンドポイント
8000	Trident REST サーバ
17546	Trident デミ作用 / レディネスプローブポートは、 Trident デミ作用ポッドで使用されます

活性/レディネスプローブポートは、を使用して設置するときに変更できます `--probe-port` フラグ。このポートがワーカーノード上の別のプロセスで使用されていないことを確認することが重要です。

Trident REST API

"tridentctl コマンドとオプション"Trident REST APIを操作する最も簡単な方法ですが、必要に応じてRESTエンドポイントを直接使用することもできます。

REST APIを使用する状況

REST APIは、Kubernetes以外の環境でTridentをスタンドアロンバイナリとして使用する高度なインストールに役立ちます。

セキュリティを強化するため、ポッド内で実行する場合、Tridentは `REST API`デフォルトでlocalhostに制限されています。この動作を変更するには、ポッド構成でTridentの引数を設定する必要があり `-address`ます。

REST APIを使用する

これらのAPIの呼び出し方法の例については、debug)(`-d`フラグを渡します。詳細については、を参照してください "tridentctlを使用したTridentの管理"。

API は次のように機能します。

取得

`GET <trident-address>/trident/v1/<object-type>`

そのタイプのすべてのオブジェクトを一覧表示します。

GET <trident-address>/trident/v1/<object-type>/<object-name>

指定したオブジェクトの詳細を取得します。

投稿（ **Post** ）

POST <trident-address>/trident/v1/<object-type>

指定したタイプのオブジェクトを作成します。

- オブジェクトを作成するには JSON 構成が必要です。各オブジェクトタイプの仕様については、を参照してください"tridentctlを使用したTridentの管理"。
- オブジェクトがすでに存在する場合、動作は一定ではありません。バックエンドが既存のオブジェクトを更新しますが、それ以外のすべてのオブジェクトタイプで処理が失敗します。

削除

DELETE <trident-address>/trident/v1/<object-type>/<object-name>

指定したリソースを削除します。

バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、を参照してください "tridentctlを使用したTridentの管理"。

コマンドラインオプション

Tridentでは、Tridentオーケストレーションツールのコマンドラインオプションがいくつか公開されています。これらのオプションを使用して、導入環境を変更できます。

ロギング

-debug

デバッグ出力を有効にします。

-loglevel <level>

ロギングレベル（debug、info、warn、error、fatal）を設定します。デフォルトは info です。

Kubernetes

-k8s_pod

このオプションまたは `-k8s_api_server` をクリックしてKubernetesのサポートを有効にしこれを設定すると、 Trident はポッドの Kubernetes サービスアカウントのクレデンシャルを使用して API サーバに接続します。これは、サービスアカウントが有効になっている Kubernetes クラスタで Trident がポッドとして実行されている場合にのみ機能します。

-k8s_api_server <insecure-address:insecure-port>

このオプションまたはを使用し `-k8s_pod`て、Kubernetesのサポートを有効にします。Trident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、Tridentをポッドの外部に導入できますが、サポートされるのはAPIサーバへの安全でない接続のみです。安全に接続するには、オプションを使用してポッドにTridentを導入し `-k8s_pod`ます。

Docker です

-volume_driver <name>

Dockerプラグインの登録時に使用するドライバ名。デフォルトはです `netapp`。

-driver_port <port-number>

UNIXドメインソケットではなく、このポートでリッスンします。

-config <file>

必須。バックエンド構成ファイルへのパスを指定する必要があります。

REST

-address <ip-or-host>

TridentのRESTサーバがリスンするアドレスを指定します。デフォルトは localhost です。localhost で聞いて Kubernetes ポッド内で実行しているときに、 REST インターフェイスにポッド外から直接アクセスすることはできません。使用 `-address ""` RESTインターフェイスにポッドのIPアドレスからアクセスできるようにするため。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

-port <port-number>

TridentのRESTサーバがリスンするポートを指定します。デフォルトは 8000 です。

-rest

RESTインターフェイスを有効にします。デフォルトは true です。

Kubernetes オブジェクトと Trident オブジェクト

リソースオブジェクトの読み取りと書き込みを行うことで、 REST API を使用して Kubernetes や Trident を操作できます。Kubernetes と Trident 、 Trident とストレージ、 Kubernetes とストレージの関係を決定するリソースオブジェクトがいくつかあります。これらのオブジェクトの中には Kubernetes で管理されるものと Trident で管理されるものがあります。

オブジェクトは相互にどのように相互作用しますか。

おそらく、オブジェクト、その目的、操作方法を理解する最も簡単な方法は、 Kubernetes ユーザからのストレージ要求を 1 回だけ処理することです。

1. ユーザーは、「 PersistentVolumeClaim 」を作成して、特定のサイズの新しい「 PersistentVolume 」を、管理者が以前に設定した Kubernetes の「 torageClass 」から要求します。
2. Kubernetes の「 torageClass 」は、 Trident をプロビジョニングツールとして識別し、要求されたクラスのボリュームのプロビジョニング方法を Trident に指示するパラメータを含んでいます。
3. Trident は、対応する「 Backends 」と「 toragePools 」を識別する同じ名前の「 S torageClass 」を参照します。この名前は、このクラスのボリュームのプロビジョニングに使用できます。

4. Trident は、対応するバックエンドにストレージをプロビジョニングし、 2 つのオブジェクトを作成します。 Kubernetes では、「 PersistentVolume 」とは、ボリュームを検索、マウント、処理する方法を Kubernetes に伝える「 PersistentVolume 」と、「 PersistentVolume 」と実際のストレージの関係を保持する Trident 内のボリュームです。
5. Kubernetes は 'PersistentVolumeClaim を新しい 'PersistentVolume' にバインドしますPersistentVolume が実行される任意のホストに PersistentVolume をマウントする 'PersistentVolumeClaim を含むポッド。
6. ユーザーは、 Trident を指す「 VolumeSnapshotClass 」を使用して、既存の PVC の「 VolumeSnapshot 」を作成します。
7. Trident が PVC に関連付けられているボリュームを特定し、バックエンドにボリュームの Snapshot を作成します。また ' スナップショットの識別方法を Kubernetes に指示する 'VolumeSnapshotContent' も作成します
8. ユーザーは 'VolumeSnapshot' をソースとして使用して 'PersistentVolumeClaim を作成できます
9. Trident は必要なスナップショットを識別し、「 PersistentVolume 」と「 Volume 」の作成に関連する一連のステップを実行します。

Kubernetes オブジェクトの詳細については、を参照することを強く推奨します "永続ボリューム" Kubernetes のドキュメントのセクション。

Kubernetes `PersistentVolumeClaim` オブジェクト

Kubernetes の「 PersistentVolumeClaim 」オブジェクトは、 Kubernetes クラスタユーザが作成したストレージの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN、solidfire-san-エコノミー 構成、solidfire-san-SAN間にあるSolidFireを実現します
trident.netapp.io/cloneFromPVC	cloneSourceVolume の実行中です	ontap - NAS 、 ontap - san 、 solidfire-san-files 、 gcvs 、 ONTAP - SAN - 経済性
trident.netapp.io/splitOnClone	splitOnClone	ONTAP - NAS 、 ONTAP - SAN
trident.netapp.io/protocol	プロトコル	任意
trident.netapp.io/exportPolicy	エクスポートポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAP-SAN
trident.netapp.io/snapshotReserve	Snapshot リザーブ	ONTAP-NAS 、 ONTAP-NAS-flexgroup 、 ONTAP-SAN 、 GCP-cvs
trident.netapp.io/snapshotDirectory	snapshotDirectory の略	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/unixPermissions	unixPermissions	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/blockSize	ブロックサイズ	solidfire - SAN

作成された PV に「削除」再利用ポリシーがある場合、 Trident は PV が解放されると（つまり、ユーザーが PVC を削除したときに）、 PV と元のボリュームの両方を削除します。削除操作が失敗した場合、 Trident は PV をマークします。そのような状態で操作が成功するか、 PV が手動で削除されるまで、定期的に再試行します。PV が「 + Retain + 」ポリシーを使用している場合、 Trident はそのポリシーを無視し、管理者が Kubernetes とバックエンドからクリーンアップすると想定します。これにより、ボリュームを削除する前にバックアップまたは検査を行うことができます。PV を削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST API (tridentctl') を使用して削除してください。

Trident では CSI 仕様を使用したボリュームスナップショットの作成がサポートされています。ボリュームスナップショットを作成し、それをデータソースとして使用して既存の PVC のクローンを作成できます。これにより、 PVS のポイントインタイムコピーを Kubernetes にスナップショットの形で公開できます。作成した Snapshot を使用して新しい PVS を作成できます。「 +On-Demand Volume Snapshots + 」を見て、これがどのように機能するかを確認してください。

Tridentが提供するのも `cloneFromPVC` および `splitOnClone` クローンを作成するためのアノテーションこれらの注釈を使用して、CSI実装を使用せずにPVCのクローンを作成できます。

例 : ユーザがすでに「 m ysql 」という PVC を持っている場合、ユーザは「 trident.netapp.io/cloneFromPVC: mysql 」などの注釈を使用して「 m ysqlclone 」という新しい PVC を作成できます。このアノテーションセットを使用すると、 Trident はボリュームをゼロからプロビジョニングするのではなく、 MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- アイドルボリュームのクローンを作成することを推奨します。
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- また 'ONTAP-NAS' および 'ONTAP-SAN' ドライバを使用すると 'pvc 注釈 trident.netapp.io/splitOnClone` を trident.netapp.io/cloneFromPVC` と組み合わせて設定することが望ましい場合がありますTrident は 'trident.netapp.io/splitOnClone` を true に設定した場合 ' クローン・ボリュームを親ボリュームからスプリットするため ' 一部のストレージ効率を失うことなく ' クローン・ボリュームのライフサイクルを親ボリュームから完全に分離しますtrident.netapp.io/splitOnClone` を設定したり 'false に設定したりしないと ' 親ボリュームとクローンボリューム間の依存関係を作成する代わりに ' バックエンドでのスペース消費が削減されますこれにより ' クローンを最初に削除しない限り ' 親ボリュームを削除できなくなりますクローンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリュームのサイズが大きく異なっており、 ONTAP ではストレージ効率化のメリットはありません。

。 `sample-input` Directoryには、Tridentで使用するPVC定義の例が含まれています。を参照してください をクリックして、Tridentボリュームに関連付けられているパラメータと設定の完全な概要を確認します。

Kubernetes `PersistentVolume` オブジェクト

Kubernetes の 'PersistentVolume' オブジェクトは 'Kubernetes クラスタで利用できるようになったストレージの一部ですポッドに依存しないライフサイクルがあります。

Trident は 'PersistentVolume' オブジェクトを作成し ' プロビジョニングするボリュームに基づいて自動的に Kubernetes クラスタに登録します自分で管理することは想定されていません。

Trident をベースとする「 torageClass 」を参照する PVC を作成すると、 Trident は対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しい PV を登録します。プロビジョニングされたボリュームと対応する PV の構成では、 Trident は次のルールに従います。

- Trident は、 Kubernetes に PV 名を生成し、ストレージのプロビジョニングに使用する内部名を生成します。どちらの場合も、名前がスコープ内で一意であることが保証されます。
- ボリュームのサイズは、 PVC で要求されたサイズにできるだけ近いサイズに一致しますが、プラットフォームによっては、最も近い割り当て可能な数量に切り上げられる場合があります。

Kubernetes `StorageClass` オブジェクト

Kubernetes の「 torageClass 」オブジェクトは、「 PersistentVolumeClaims 」内の名前によって指定され、一連のプロパティを持つストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Trident を使用する Kubernetes の「 torageClass 」オブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、クラスのボリュームのプロビジョニング方法を Trident に指示します。

ストレージクラスのパラメータは次のとおりです。

属性	を入力します	必須	説明
属性（ Attributes ）	[string] 文字列をマップします	いいえ	後述の「属性」セクションを参照してください
ストレージプール	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング

属性	を入力します	必須	説明
AdditionalStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
excludeStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	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ています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、GCP-cvs、azure-NetApp-files 、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ

属性	を入力します	値	提供	リクエスト	でサポートされます
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、 ONTAP-NAS-エコノミー 、 ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

^1 ^ ： ONTAP Select システムではサポートされていません

ほとんどの場合、要求された値はプロビジョニングに直接影響します。たとえば、シックプロビジョニングを要求した場合、シックプロビジョニングボリュームが使用されます。ただし、 Element ストレージプールでは、提供されている IOPS の最小値と最大値を使用して、要求された値ではなく QoS 値を設定します。この場合、要求された値はストレージプールの選択のみに使用されます。

理想的には ' 属性だけを使用して ' 特定のクラスのニーズを満たすために必要なストレージの特性をモデル化できますTrident は ' 指定した属性の *ALL* に一致するストレージ・プールを自動的に検出して選択します

「 attributes 」を使用してクラスに適切なプールを自動的に選択できない場合は、「 toragePools 」および「 additionalStoragePools 」パラメータを使用してプールをさらに改良したり、特定のプールセットを選択したりできます。

'toragePools' パラメータを使用すると ' 指定した属性に一致するプールのセットをさらに制限できますつまり 'attributes' パラメータと 'toragePools' パラメータで指定されたプールの交点をプロビジョニングに使用しますどちらか一方のパラメータを単独で使用することも、両方を同時に使用することも

「 additionalStoragePools 」パラメータを使用すると、「 attributes 」パラメータと「 toragePools 」パラメータで選択されたプールに関係なく、 Trident がプロビジョニングに使用するプールのセットを拡張できます。

excludeStoragePools' パラメータを使用して、 Trident がプロビジョニングに使用するプールのセットをフィルタリングできます。このパラメータを使用すると、一致するプールがすべて削除されます。

'toragePools' パラメータと 'additionalStoragePools' パラメータでは ' 各エントリは '<backend>:<storagePoolList>' の形式で指定したバックエンドのストレージプールのカンマ区切りリストですたとえば、「 additionalStoragePools 」の値は「 ontapnas_192.168.1.100 ： aggr1 、 aggr2 ； solidfire_192.168.1.101 ： bronze 」のようになります。これらのリストでは、バックエンド値とリスト値の

両方に正規表現値を使用できます。tridentctl get backend を使用してバックエンドとそのプールのリストを取得できます

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に Trident が選択するストレージプール / バックエンドには影響しません。代わりに、 Kubernetes Persistent Volume でサポートされるパラメータを提供するだけです。ワーカーノードはファイルシステムの作成操作を担当し、 xfsprogs などのファイルシステムユーティリティを必要とする場合があります。

属性	を入力します	値	説明	関連するドライバ	**Kubernetes** のバージョン
FSstype （英語）	文字列	ext4、ext3、xfs	ブロックボリュームのファイルシステムのタイプ	solidfire-san-group、ontap/nas、ontap -nas-エコノミー、ontap -nas-flexgroup 、ontap -san 、ONTAP - SAN -経済性	すべて
allowVolumeExpansion の略	ブール値	true false	PVC サイズの拡張のサポートをイネーブルまたはディセーブルにします	ONTAPNAS 、ONTAPNAS エコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN 、ONTAP-SAN-エコノミー 、solidfire-san-, gcvs, azure-netapp-files	1.11 以上
volumeBindingMode のようになりました	文字列	即時、WaitForFirstConsumer	ボリュームバインドと動的プロビジョニングを実行するタイミングを選択します	すべて	1.19～1.26

- 。 `fsType` パラメータは、SAN LUNに必要なファイルシステムタイプを制御する場合に使用します。また、Kubernetesでは、の機能も使用されます `fsType` ファイルシステムが存在することを示すために、ストレージクラスに格納します。ボリューム所有権は、を使用して制御できます `fsGroup` ポッドのセキュリティコンテキスト（使用する場合のみ） `fsType` が設定されます。を参照してください "Kubernetes ：ポッドまたはコンテナのセキュリティコンテキストを設定します" を使用したボリューム所有権の設定の概要については、を参照してください `fsGroup` コンテキスト（Context）。Kubernetesでが適用されます `fsGroup` 次の場合のみ値を指定します

  - 「 fsType 」はストレージクラスで設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 NFS エクスポートにはファイルシステムがすでに存在します。fsGroup を使用するには ' ストレージ・クラスで fsType を指定する必要がありますこの値は 'NFS' に設定することも ' ヌル以外の任意の値に設定することもできます

- を参照してください "ボリュームを展開します" ボリューム拡張の詳細については、を参照してください。
- Trident インストーラバンドルには、「 `sample -input/storageclass-*.yaml 」で Trident で使用するストレージクラス定義の例がいくつか用意されています。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

Kubernetes `VolumeSnapshotClass` オブジェクト

Kubernetes 'VolumeSnapshotClass' オブジェクトは 'S torageClasses' に似ていますこの Snapshot コピーは、複数のストレージクラスの定義に役立ちます。また、ボリューム Snapshot によって参照され、Snapshot を必要な Snapshot クラスに関連付けます。各ボリューム Snapshot は、単一のボリューム Snapshot クラスに関連付けられます。

スナップショットを作成するには 'VolumeSnapshotClass' を管理者が定義する必要がありますボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

「 driver 」は、「 csi-snapclass 」クラスのボリュームスナップショットの要求が Trident によって処理される Kubernetes を指定します。「要素ポリシー」は、スナップショットを削除する必要がある場合に実行されるアクションを指定します。「削除ポリシー」が「削除」に設定されている場合、 Snapshot を削除すると、ボリューム Snapshot オブジェクトおよびストレージクラスタ上の基盤となる Snapshot は削除されます。または、「 Retain 」に設定すると、「 VolumeSnapshotContent 」と物理スナップショットが保持されます。

Kubernetes `VolumeSnapshot` オブジェクト

Kubernetes の VolumeSnapshot オブジェクトは ' ボリュームのスナップショットを作成する要求ですPVC が

ボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームスナップショット要求が受信されると、 Trident はバックエンドでのボリュームのスナップショット作成を自動的に管理し、ユニークな「 VolumeSnapshotContent 」オブジェクトを作成することによってスナップショットを公開します。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。

> VolumeSnapshot のライフサイクルはソース PVC とは無関係です。ソース PVC が削除されても、スナップショットは維持されます。スナップショットが関連付けられている PVC を削除すると、 Trident はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes `VolumeSnapshotContent` オブジェクト

Kubernetes の「 VolumeSnapshotContent 」オブジェクトは、すでにプロビジョニングされているボリュームから取得されたスナップショットを表します。これは「 PersistentVolume 」と似ており、ストレージ・クラスタ上でプロビジョニングされた Snapshot を表します。「 PersistentVolumeClaim 」および「 PersistentVolume 」オブジェクトと同様に、スナップショットが作成されると、「 VolumeContent Snapshot」 オブジェクトは「 VolumeSnapshot 」オブジェクトへの 1 対 1 のマッピングを保持します。これは、スナップショットの作成を要求しました。

「 VolumeSnapshotContent 」オブジェクトには、スナップショットを一意に識別する詳細（「 napshotHandle 」など）が含まれています。この「 napshotHandle 」は、 PV の名前と「 VolumeSnapshotContent 」オブジェクトの名前を組み合わせた一意のものです。

Trident では、スナップショット要求を受信すると、バックエンドにスナップショットが作成されます。スナップショットが作成されると、 Trident は「 VolumeSnapshotContent 」オブジェクトを構成し、そのスナップショットを Kubernetes API に公開します。

> 通常、オブジェクトを管理する必要はありませ `VolumeSnapshotContent`ん。ただし、Trident の外部でを作成する場合は例外です"ボリュームSnapshotのインポート"。

Kubernetes `CustomResourceDefinition` オブジェクト

Kubernetes カスタムリソースは、管理者が定義した Kubernetes API 内のエンドポイントであり、類似するオブジェクトのグループ化に使用されます。Kubernetes では、オブジェクトのコレクションを格納するためのカスタムリソースの作成をサポートしています。これらのリソース定義を取得するには 'kubectl get CRDs' を実行します

カスタムリソース定義（ CRD ）と関連するオブジェクトメタデータは、 Kubernetes によってメタデータストアに格納されます。これにより、 Trident の独立したストアが不要になります。

Tridentは、オブジェクトを使用し `CustomResourceDefinition`て、Tridentバックエンド、Tridentストレージクラス、TridentボリュームなどのTridentオブジェクトのIDを保持します。これらのオブジェクトは Trident によって管理されます。また、 CSI のボリュームスナップショットフレームワークには、ボリュームスナップショットの定義に必要ないくつかの SSD が導入されています。

CRD は Kubernetes の構成要素です。上記で定義したリソースのオブジェクトは Trident によって作成されます。簡単な例として 'tridentctl' を使用してバックエンドを作成すると ' 対応する tridentBackendsCRD オブジェクトが Kubernetes によって消費されるように作成されます

Trident の CRD については、次の点に注意してください。

- Trident をインストールすると、一連の CRD が作成され、他のリソースタイプと同様に使用できるようになります。
- Tridentをアンインストールするには、を使用します `tridentctl uninstall` コマンドであるTridentポッドが削除されましたが、作成されたSSDはクリーンアップされません。を参照してください "Trident をアンインストールします" Trident を完全に削除して再構成する方法を理解する。

Trident `StorageClass`オブジェクト

TridentではKubernetesに対応するストレージクラスが作成されます `StorageClass` を指定するオブジェクト `csi.trident.netapp.io` プロビジョニング担当者のフィールドに入力します。ストレージクラス名がKubernetesの名前と一致していること `StorageClass` 表すオブジェクト。

> Kubernetes では、 Trident をプロビジョニングツールとして使用する Kubernetes 「 torageClass 」が登録されると、これらのオブジェクトが自動的に作成されます。

ストレージクラスは、ボリュームの一連の要件で構成されます。Trident は、これらの要件と各ストレージプール内の属性を照合し、一致する場合は、そのストレージプールが、そのストレージクラスを使用するボリュームのプロビジョニングの有効なターゲットになります。

REST API を使用して、ストレージクラスを直接定義するストレージクラス設定を作成できます。ただし、Kubernetes の導入では、新しい Kubernetes の「 torageClass 」オブジェクトを登録するときに、これらのオブジェクトが作成されることを期待しています。

Trident バックエンドオブジェクト

バックエンドとは、 Trident がボリュームをプロビジョニングする際にストレージプロバイダを表します。 1 つの Trident インスタンスであらゆる数のバックエンドを管理できます。

> これは、自分で作成および管理する 2 つのオブジェクトタイプのうちの 1 つです。もう 1 つは、 Kubernetes の「 torageClass 」オブジェクトです。

これらのオブジェクトの作成方法の詳細については、を参照してください。 "バックエンドの設定"。

Trident `StoragePool`オブジェクト

ストレージプールは、各バックエンドでのプロビジョニングに使用できる個別の場所を表します。ONTAP の場合、これらは SVM 内のアグリゲートに対応します。NetApp HCI / SolidFire では、管理者が指定した QoS 帯域に対応します。Cloud Volumes Service の場合、これらはクラウドプロバイダのリージョンに対応します。各ストレージプールには、パフォーマンス特性とデータ保護特性を定義するストレージ属性があります。

他のオブジェクトとは異なり、ストレージプールの候補は常に自動的に検出されて管理されます。

Trident `Volume`オブジェクト

ボリュームは、 NFS 共有や iSCSI LUN などのバックエンドエンドエンドエンドポイントで構成される、プロビジョニングの基本単位です。Kubernetes では ' これらは 'PersistentVolumes' に直接対応しますボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

- Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。
- 関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP （ NAS 、 SAN ）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - * ：使用する Snapshot ポリシー
Snapshot リザーブ	文字列	いいえ	ONTAP - * ： Snapshot 用にリザーブされているボリュームの割合
エクスポートポリシー	文字列	いいえ	ONTAP-NAS* ：使用するエクスポートポリシー
snapshotDirectory の略	ブール値	いいえ	ONTAP-NAS* ： Snapshot ディレクトリが表示されているかどうか
unixPermissions	文字列	いいえ	ONTAP-NAS* ：最初の UNIX 権限

属性	を入力します	必須	説明
ブロックサイズ	文字列	いいえ	SolidFire - * ：ブロック / セクターサイズ
ファイルシステム	文字列	いいえ	ファイルシステムのタイプ

Trident は ' ボリュームの作成時に internalName を生成しますこの構成は 2 つのステップで構成されます。最初に、ストレージプレフィックス（デフォルトの「 trident 」またはバックエンド構成のプレフィックス）をボリューム名の前に付加し、「 <prefix> - <volume-name> 」という形式の名前を付けます。その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAP バックエンドでは、ハイフンをアンダースコアで置き換えます（つまり、内部名は「 <prefix>_<volume-name>` 」になります）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム設定を使用して、 REST API を使用してボリュームを直接プロビジョニングできますが、 Kubernetes 環境では、ほとんどのユーザが標準の Kubernetes の「 PersistentVolumeClaim 」メソッドを使用することを想定しています。Trident は、プロビジョニングプロセスの一環として、このボリュームオブジェクトを自動的に作成します。

Trident `Snapshot`オブジェクト

Snapshot はボリュームのポイントインタイムコピーで、新しいボリュームのプロビジョニングやリストア状態に使用できます。Kubernetes では ' これらは 'VolumeSnapshotContent' オブジェクトに直接対応します各 Snapshot には、 Snapshot のデータのソースであるボリュームが関連付けられます。

個々の「スナップショット」オブジェクトには、以下のプロパティが含まれています。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン（「 1 」）
名前	文字列	はい。	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetes の「 VolumeSnapshot 」オブジェクト要求が作成されると、 Trident は元のストレージシステム上にスナップショットオブジェクトを作成することによって動作します。このスナップショットオブジェクトの「 internalName 」は、プレフィックス「 snapshot-` と「 VolumeSnapshot 」オブジェクトの「 UID 」を組み合わせることによって生成されます（例：「 snapshot-e8d8d8a0ca-9826-11e9-9807-525400f3f660 」

）。「 volumeName 」と「 volumeInternalName 」には、バッキングボリュームの詳細を取得して値を設定します。

Trident `ResourceQuota`オブジェクト

Tridentデーモンセットは、Kubernetesで利用可能な最高の優先度クラスであるプライオリティクラスを使用して `system-node-critical`、ノードの正常なシャットダウン時にTridentがボリュームを識別してクリーンアップできるようにし、リソースへの負荷が高いクラスタでは、Tridentデーモンセットポッドが優先度の低いワークロードをプリエンプトできるようにします。

これを実現するために、Tridentはオブジェクトを使用し `ResourceQuota`て、`Trident`デーモンセットの「`system-node-critical`」優先クラスを確実に満たします。デプロイメントおよびデーモンセットの作成の前に、`Trident`はオブジェクトを検索し `ResourceQuota`、検出されていない場合は適用します。

デフォルトのリソース割り当ておよび優先クラスをより詳細に制御する必要がある場合は'custom.yamlを生成するか'Helmチャートを使用してResourceQuotaオブジェクトを構成します

次に示すのは'ResourceQuota'オブジェクトがTridentのデマ作用を優先する例です

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
     matchExpressions:
       - operator : In
         scopeName: PriorityClass
         values: ["system-node-critical"]
```

リソースクォータの詳細については、を参照してください。 "Kubernetes：リソースクォータ"。

クリーンアップ `ResourceQuota` インストールが失敗した場合

まれに'ResourceQuotaオブジェクトが作成された後にインストールが失敗する場合は'最初に実行します "アンインストール中です" を再インストールします。

それでも解決しない場合は'ResourceQuotaオブジェクトを手動で削除します

取り外します `ResourceQuota`

独自のリソース割り当てを制御する場合は、次のコマンドを使用してTridentオブジェクトを削除できます `ResourceQuota`。

```
kubectl delete quota trident-csi -n trident
```

PODセキュリティ標準（PSS）およびセキュリティコンテキストの制約（SCC）

Kubernetesポッドのセキュリティ標準（PSS）とポッドのセキュリティポリシー（PSP）によって、権限レベルが定義され、ポッドの動作が制限されます。また、OpenShift Security Context Constraints（SCC）でも、OpenShift Kubernetes Engine固有のポッド制限を定義します。このカスタマイズを行うために、Tridentはインストール時に特定の権限を有効にします。次のセクションでは、Tridentによって設定される権限について詳しく説明します。

PSSは、Podセキュリティポリシー（PSP）に代わるものです。PSPはKubernetes v1.21で廃止され、v1.25で削除されます。詳細については、を参照してください。 "Kubernetes：セキュリティ"。

必須の**Kubernetes Security Context**と関連フィールド

アクセス権	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、を参照してください "Kubernetes：マウントの伝播"。
ホストネットワーク	iSCSIデーモンに必要です。「iscsiadm」はiSCSIマウントを管理し、ホストネットワークを使用してiSCSIデーモンと通信します。
ホストIPC	NFSは'IPC（プロセス間通信）を使用して'nfsdと通信します
ホストPID	NFSで開始する必要があり `rpc-statd`ます。Tridentは、NFSボリュームをマウントする前にが実行されているかどうかをホストプロセスに照会して判断し`rpc-statd`ます。
機能	SYS_Admin'機能は'特権コンテナのデフォルト機能の一部として提供されますたとえば、Dockerは特権コンテナにこれらの機能を設定します。「CapPrm：0000003fffffffffffff`Capff：0000003fffffffff'」
Seccom	Seccompプロファイルは特権コンテナでは常に「制限されていない」ため、Tridentでは有効にできません。
SELinux	OpenShiftでは、特権コンテナは（「Super Privileged Container」）ドメインで実行され、非特権コンテナはドメインで実行さ `spc_t`れ `container_t`ます。では `containerd`、がインストールされていると `container-selinux`、すべてのコンテナがドメイン内で実行され `spc_t`、SELinuxが実質的に無効になります。そのため、Tridentはコンテナに追加しません `seLinuxOptions`。

アクセス権	説明
DAC	特権コンテナは、ルートとして実行する必要があります。CSIに必要なUNIXソケットにアクセスするために、非特権コンテナはrootとして実行されます。

PODセキュリティ標準（PSS）

ラベル	説明	デフォルト
'pod -security.esvus.io/enforce `po-security.Kubernetes io/enforce-version	Tridentコントローラとノードをインストールネームスペースに登録できるようにします。ネームスペースラベルは変更しないでください。	「enforce：特権」「enforce-version：」は、現在のクラスタのバージョンまたはテストされたPSSの最新バージョンです

名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating：…」または「Warning：trident-csi-…」が表示される場合があります。この場合はPrivilegedの名前空間ラベルが変更されていないかどうかを確認してくださいその場合は、Tridentを再インストールします。

PoDセキュリティポリシー（PSP）

フィールド	説明	デフォルト
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowedCSIDrivers	TridentはインラインCSIエフェメラルボリュームを使用しません。	空です
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
「allowedFlexVolumes」を参照してください	Tridentはを利用しません "FlexVolドライバ"そのため、これらのボリュームは許可されるボリュームのリストに含まれていません。	空です
「allowedHostPaths」を参照してください	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空です
allowedProcMountTypes	Tridentでは'ProcMountTypes'は使用しません	空です
"allowedUnsafeSysctls"	Tridentには安全でないsysctls'は必要ありません。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です

フィールド	説明	デフォルト
defaultAllowPrivilegeEscalation`	権限の昇格は、各Tridentポッドで処理されます。	「偽」
「forbiddenSysctls」と入力します	sysctlsは許可されません	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「hostIPC」	NFSボリュームをマウントするには'nfsdと通信するためにホストIPCが必要です	「真」
「ホストネットワーク」	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「hostPID」	ノードでRPC-statdが実行されているかどうかを確認するには'ホストPIDが必要です	「真」
「hostPorts」	Tridentはホストポートを使用しません。	空です
「特権」	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsGroup`	Tridentコンテナはrootとして実行されます。	「RunAsAny」
runAsUser	Tridentコンテナはrootとして実行されます。	runAsany`
runtimeClass'	Tridentは'RuntimeClasses'を使用しません	空です
SELinux	Tridentは'seLinuxOptions'を設定していませんこれは'コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」、「projected」、「emptyDir」

セキュリティコンテキストの制約（SCC）

ラベル	説明	デフォルト
allowHostDirVolumePlugin	Tridentノードのポッドは、ノードのルートファイルシステムをマウントします。	「真」
"allowHostIPC"	NFSボリュームをマウントするには’nfsdと通信するためにホストIPCが必要です	「真」
「allowHostNetwork」を参照してください	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「allowHostPID」を参照してください	ノードでRPC-statdが実行されているかどうかを確認するには’ホストPIDが必要です	「真」
"allowHostPorts`	Tridentはホストポートを使用しません。	「偽」
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowPrivilegeContainer]を参照してください	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
"allowedUnsafeSysctls"	Tridentには安全でないsysctls’は必要ありません。	「 NONE 」
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「グループ」	このSCCはTridentに固有で、ユーザにバインドされています。	空です
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsUser	Tridentコンテナはrootとして実行されます。	「RunAsAny」

ラベル	説明	デフォルト
「seLinuxContext」	Tridentは'seLinuxOptions'を設定していませんこれは'コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「seccompProfiles」	特権のあるコンテナは常に「閉鎖的」な状態で実行されます。	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ユーザー」	このSCCをTridentネームスペースのTridentユーザにバインドするエントリが1つあります。	該当なし
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」, downwardAPI, projected , emptyDir

法的通知

著作権に関する声明、商標、特許などにアクセスできます。

著作権

"https://www.netapp.com/company/legal/copyright/"

商標

NetApp 、 NetApp のロゴ、および NetApp の商標ページに記載されているマークは、 NetApp, Inc. の商標です。その他の会社名および製品名は、それぞれの所有者の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

ネットアップが所有する特許の最新リストは、次のサイトで入手できます。

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

プライバシーポリシー

"https://www.netapp.com/company/legal/privacy-policy/"

オープンソース

NetApp for Tridentで使用されているサードパーティの著作権およびライセンスは、の各リリースのNOTICESファイルで確認できます https://github.com/NetApp/trident/。