Trident 25.10 ドキュメント

Trident

NetApp
March 05, 2026

目次

Trident 25.10 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、NetApp Tridentの最新バージョンにおける新機能、機能拡張、およびバグ修正に関する情報が記載されています。

> インストーラーのzipファイルに含まれているLinux用 `tridentctl`バイナリは、テスト済みかつサポート対象のバージョンです。zipファイルの `/extras`部分に含まれている `macos`バイナリは、テストもサポートもされていないことに注意してください。

25.10の新機能

Trident および Trident Protect の新機能（機能強化、修正、廃止を含む）について説明します。

Trident

機能強化

- **Kubernetes**：
  - ONTAP-SAN（iSCSI および FC）に加えて、ONTAP-NAS NFS および ONTAP-SAN-Economy ドライバー用の v1beta1 Volume Group Snapshot Kubernetes API による CSI Volume Group Snapshot のサポートが追加されました。"ボリュームグループのスナップショットを操作する"を参照してください。
  - ONTAP-NAS および ONTAP-NAS-Economy（両方の NAS ドライバーの SMB を除く）、および ONTAP-SAN および ONTAP-SAN-Economy ドライバーの強制ボリュームデタッチによる自動ワークロードフェイルオーバーのサポートが追加されました。"Tridentを使用したステートフルアプリケーションのフェイルオーバーの自動化"を参照してください。
  - FCPボリュームのノード操作のスケーラビリティを向上させるため、Tridentノードの同時実行性が強化されました。
  - ONTAP NAS ドライバーに ONTAP AFX サポートを追加しました。"ONTAP NAS 構成オプションと例"を参照してください。
  - TridentOrchestrator CR および Helm チャートの値を使用して、Trident コンテナの CPU とメモリのリソース要求と制限を設定するためのサポートが追加されました。（"問題 #1000"、"問題 #927"、"問題 #853"、"問題 #592"、"問題 #110"）。
  - ASAr2 パーソナリティの FC サポートを追加しました。"ONTAP SAN 構成オプションと例"を参照してください。
  - HTTP ではなく HTTPS で Prometheus メトリックを提供するオプションが追加されました。"Tridentを監視"を参照してください。
  - ボリュームをインポートする際に元の名前を保持しながらTridentにボリュームのライフサイクルを管理させるオプション `--no-rename`を追加しました。"ボリュームをインポート"を参照してください。
  - Trident デプロイメントは、system-cluster-critical 優先度クラスで実行されるようになりました。
- Tridentコントローラーがhelm、operator、tridentctlを介してホストネットワークを使用するためのオプションを追加しました（"問題 #858"）。

- ANFドライバに手動QoSサポートを追加し、Trident 25.10で本番環境対応になりました。この実験的な機能強化はTrident 25.06で導入されました。

実験的な機能強化

> 本番環境では使用しないでください。

- **[テクニカルプレビュー]**：ONTAP-NAS（NFSのみ）およびONTAP-SAN（統合ONTAP 9のNVMe）の同時実行のサポートを追加しました。これは、既存のONTAP-SANドライバ（統合ONTAP 9のiSCSIおよびFCPプロトコル）のテクニカルプレビューに加えてのものです。

修正

- **Kubernetes**：
  - Linux DaemonSet を node-driver-registrar に標準化して Windows DaemonSet とコンテナイメージの命名に一致させることで、CSI node-driver-registrar コンテナ名の不一致を修正しました。
  - レガシー qtree のエクスポートポリシーが適切にアップグレードされない問題を修正しました。
- **Openshift**：
  - SCC の allowHostDirVolumePlugin が false に設定されているため、OpenShift の Windows ノードで Trident ノードポッドが起動しない問題を修正しました（"問題 #950"）。
- Kubernetes API QPS が Helm 経由で設定されない問題を修正（"問題 #975"）。
- 同じ Kubernetes ノード上の NVMe ベースの XFS ファイルシステム PVC のスナップショットに基づいて永続ボリューム要求（PVC）をマウントできない問題を修正しました。
- バックエンドごとに一意または共有のサブシステム名（例：netappdvp_subsystem）を追加することで、NDVP モードでホスト/Docker を再起動した後の UUID 変更の問題を修正しました。
- Tridentを23.10より前のバージョンから24.10以上にアップグレードする際のiSCSIボリュームのマウントエラーを修正し、「invalid SANType」の問題を解決しました。
- Tridentコントローラを再起動しないとTridentバックエンドの状態がオンライン/オフラインに移行しなかった問題を修正しました。
- PVC のサイズ変更が遅くなる原因となる断続的な競合状態を修正しました。
- ボリュームクローンの失敗時にスナップショットがクリーンアップされない問題を修正しました。
- カーネルによってデバイスパスが変更されたときにボリュームのステージ解除に失敗する問題を修正しました。
- LUKS デバイスがすでに閉じられているためにボリュームのステージ解除に失敗する問題を修正しました。
- ストレージ操作が遅いことでContextDeadlineエラーが発生する問題を修正しました。
- Trident オペレーターは、設定可能な k8s-timeout を待機して Trident バージョンを確認します。

Trident Protect

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ナーによってサポートされるステートフルKubernetesアプリケーションの機能性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。

機能強化

- スケジュール CR とバックアップ CR のスナップショット CR タイムアウトを制御するための注釈を追加しました：
  - `protect.trident.netapp.io/snapshot-completion-timeout`
  - `protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout`
  - `protect.trident.netapp.io/volume-snapshots-created-timeout`

  "サポートされているバックアップとスケジュールのアノテーション"を参照してください。
- バックアップCRによって使用されるPVCバインドタイムアウトを設定するためのアノテーションをスケジュールCRに追加しました `protect.trident.netapp.io/pvc-bind-timeout-sec`。"サポートされているバックアップとスケジュールのアノテーション"を参照してください。
- 実行フックの失敗を示す新しいフィールドを含む、改善された `tridentctl-protect`バックアップおよびスナップショットのリスト。

25.06.2 の変更点

Trident

修正

- **Kubernetes**：Kubernetes ノードからボリュームをデタッチするときに不正な iSCSI デバイスが検出される重大な問題を修正しました。

25.06.1 の変更点

Trident

SolidFireをご利用のお客様は、ボリュームの非公開時に既知の問題があるため、25.06.1にアップグレードしないでください。この問題に対処するため、25.06.2がまもなくリリースされる予定です。

修正

- **Kubernetes**：
  - サブシステムからマッピング解除される前にNQNがチェックされなかった問題を修正しました。
  - LUKS デバイスを複数回閉じようとするとボリュームのデタッチに失敗する問題を修正しました。
  - 作成以降にデバイスパスが変更された場合の iSCSI ボリュームのステージング解除を修正しました。
  - ストレージクラス間でのボリュームのブロッククローニング。
- **OpenShift**：OCP 4.19 で iSCSI ノードの準備が失敗する問題を修正しました。
- SolidFire バックエンドを使用してボリュームのクローンを作成する際のタイムアウトを増やしました（"問題#1008"）。

25.06の変更点

Trident

機能強化

- **Kubernetes**：
  - ONTAP-SAN iSCSIドライバー用の `v1beta1`ボリュームグループスナップショットKubernetes APIを使用したCSIボリュームグループスナップショットのサポートを追加しました。"ボリュームグループのスナップショットを操作する"を参照してください。

    VolumeGroupSnapshotは、ベータ API を備えた Kubernetes のベータ機能です。Kubernetes 1.32は、VolumeGroupSnapshotに必要な最小バージョンです。

  - iSCSI に加えて NVMe/TCP 用の ONTAP ASA r2 のサポートを追加しました。"ONTAP SAN 構成オプションと例"を参照してください。
  - ONTAP-NAS および ONTAP-NAS-Economy ボリュームに対する安全な SMB サポートが追加されました。セキュリティ強化のため、Active Directory ユーザーとグループを SMB ボリュームで使用できるようになりました。"セキュアSMBを有効にする"を参照してください。
  - iSCSI ボリュームのノード操作のスケーラビリティを向上させるために、Trident ノードの同時実行性が強化されました。
  - LUKS ボリュームを開くときに `--allow-discards`を追加し、スペース再利用のための破棄/TRIM コマンドを許可しました。
  - LUKS で暗号化されたボリュームをフォーマットする際のパフォーマンスが向上しました。
  - 障害が発生したが部分的にフォーマットされた LUKS デバイスの LUKS クリーンアップ機能が強化されました。
  - NVMe ボリュームの接続と切断に関する Trident ノードの冪等性が強化されました。
  - ONTAP-SAN-Economy ドライバーの Trident ボリューム構成に `internalID`フィールドを追加しました。
  - NVMe バックエンド用の SnapMirror によるボリュームレプリケーションのサポートが追加されました。"SnapMirrorを使用したボリュームのレプリケート"を参照してください。

実験的な機能強化

本番環境では使用しないでください。

- [テクニカルプレビュー] `--enable-concurrency`機能フラグを使用して、Tridentコントローラの同時操作を有効にしました。これにより、コントローラの操作を並行して実行できるようになり、ビジーな環境や大規模な環境でのパフォーマンスが向上します。

  この機能は実験段階であり、現在は ONTAP-SAN ドライバー（iSCSI および FCP プロトコル）を使用した限定的な並列ワークフローをサポートしています。

- [Tech Preview] ANF ドライバーによる手動 QOS サポートが追加されました。

修正

- **Kubernetes**：
  - CSI NodeExpandVolumeで、基盤となるSCSIディスクが利用できない場合にマルチパスデバイスのサイズが不一致のままになる可能性がある問題を修正しました。
  - ONTAP-NAS および ONTAP-NAS-Economy ドライバの重複するエクスポートポリシーのクリーンアップに失敗する問題を修正しました。
  - GCNV ボリュームが `nfsMountOptions`未設定時に NFSv3 にデフォルト設定される問題を修正しました。現在、NFSv3 プロトコルと NFSv4 プロトコルの両方がサポートされています。`nfsMountOptions`が指定されていない場合は、ホストのデフォルトの NFS バージョン（NFSv3 または NFSv4）が使用されます。
  - Trident を Kustomize を使用してインストールする際の展開の問題を修正しました（"問題 #831"）。
  - スナップショットから作成された PVC のエクスポート ポリシーが欠落していた問題を修正しました（"問題 #1016"）。
  - ANF ボリューム サイズが 1 GiB 単位に自動的に調整されない問題を修正しました。
  - Bottlerocket で NFSv3 を使用する際の問題を修正しました。
- サイズ変更の失敗にもかかわらず、ONTAP-NAS-Economy ボリュームが最大 300 TB まで拡張される問題を修正しました。
- ONTAP REST API の使用時にクローン スプリット処理が同期的に実行されていた問題を修正しました。

非推奨：

- **Kubernetes**：サポートされる最小 Kubernetes を v1.27 に更新しました。

Trident Protect

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ナーによってサポートされるステートフルKubernetesアプリケーションの機能性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。

機能強化

- 復元時間が改善され、より頻繁に完全バックアップを実行するオプションが提供されます。
- Group-Version-Kind（GVK）フィルタリングによるアプリケーション定義と選択的復元の粒度が向上しました。
- AppMirrorRelationship（AMR）とNetApp SnapMirrorを使用する場合の効率的な再同期と逆レプリケーションにより、完全な PVC レプリケーションを回避します。
- EKS Pod Identity を使用して AppVault バケットを作成する機能が追加され、EKS クラスターのバケット認証情報を含むシークレットを指定する必要がなくなりました。
- 必要に応じて、復元名前空間内のラベルと注釈の復元をスキップする機能を追加しました。
- AppMirrorRelationship（AMR）は、送信元 PVC の拡張をチェックし、必要に応じて宛先 PVC で適切な拡張を実行します。

修正

- 以前のスナップショットのスナップショット注釈値が新しいスナップショットに適用されていたバグを修

正しました。すべてのスナップショット注釈が正しく適用されるようになりました。

- 定義されていない場合は、デフォルトでデータ ムーバー暗号化（Kopia / Restic）のシークレットを定義します。
- S3 appvault作成の検証とエラー メッセージが改善されました。
- AppMirrorRelationship（AMR）は、失敗を回避するために、バインドされた状態の PV のみを複製するようになりました。
- 多数のバックアップを持つAppVaultでAppVaultContentを取得する際にエラーが表示される問題を修正しました。
- KubeVirt VMSnapshot は、障害を回避するために、リストアおよびフェイルオーバー処理から除外されます。
- Kopia のデフォルトの保持スケジュールが、ユーザーがスケジュールに設定した内容を上書きしたために、スナップショットが早期に削除されるという Kopia の問題を修正しました。

25.02.1 の変更点

Trident

修正

- **Kubernetes**：
  - trident-operator で、デフォルト以外のイメージレジストリを使用した場合に、サイドカーイメージ名とバージョンが正しく入力されない問題を修正しました（"問題 #983"）。
  - ONTAP フェイルオーバーギブバック中にマルチパス セッションの回復に失敗する問題を修正しました（"問題 #961"）。

25.02の変更点

Trident 25.02以降、新機能の概要では、TridentとTrident Protectの両方のリリースにおける機能強化、修正、および廃止予定について詳細が説明されています。

Trident

機能強化

- **Kubernetes**：
  - iSCSI 用の ONTAP ASA r2 のサポートを追加しました。
  - 非正常なノードシャットダウンのシナリオ中に ONTAP-NAS ボリュームを強制的に切断するサポートが追加されました。新しい ONTAP-NAS ボリュームは、Trident によって管理されるボリュームごとのエクスポート ポリシーを利用するようになります。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非公開時に新しいエクスポート ポリシー モデルに移行するためのアップグレード パスを提供しました。
  - cloneFromSnapshotアノテーションを追加しました。
  - 名前空間間のボリューム クローン作成のサポートが追加されました。
  - 拡張 iSCSI 自己修復スキャン修復により、正確なホスト、チャネル、ターゲット、LUN ID による再スキャンが開始されます。

- Kubernetes 1.32 のサポートが追加されました。

- **OpenShift**：
  - ROSA クラスタ上の RHCOS の自動 iSCSI ノード準備のサポートが追加されました。
  - OpenShift Virtualization for ONTAP ドライバーのサポートを追加しました。
- ONTAP-SAN ドライバーに Fibre Channel サポートが追加されました。
- NVMe LUKS サポートを追加しました。
- すべてのベースイメージをスクラッチイメージに切り替えました。
- iSCSI セッションがログインする必要があるのにログインされていない場合の iSCSI 接続状態の検出とログ記録を追加しました（"問題 #961"）。
- google-cloud-netapp-volumes ドライバーによる SMB ボリュームのサポートが追加されました。
- ONTAP ボリュームが削除時に回復キューをスキップできるようにするためのサポートを追加しました。
- タグの代わりに SHA を使用してデフォルトのイメージを上書きするサポートが追加されました。
- Trident インストーラーに image-pull-secrets フラグを追加しました。

修正

- **Kubernetes**：
  - 自動エクスポートポリシーから欠落しているノードIPアドレスを修正しました（"問題 #965"）。
  - ONTAP-NAS-Economy の自動エクスポート ポリシーがボリュームごとのポリシーに途中で切り替わる問題を修正しました。
  - 利用可能なすべての AWS ARN パーティションをサポートするようにバックエンド構成の認証情報を修正しました（"問題 #913"）。
  - Trident オペレータに自動コンフィギュレータ調整を無効にするオプションを追加しました（"問題 #924"）。
  - csi-resizer コンテナ用にsecurityContextを追加（"問題 #976"）。

Trident Protect

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ナーによってサポートされるステートフルKubernetesアプリケーションの機能性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。

機能強化

- KubeVirt / OpenShift Virtualization VM の volumeMode：File と volumeMode：Block（raw デバイス）ストレージの両方に対するバックアップとリストアのサポートを追加しました。このサポートはすべての Trident ドライバーと互換性があり、Trident Protect で NetApp SnapMirror を使用してストレージをレプリケートする際の既存の保護機能を強化します。
- Kubevirt 環境のアプリケーション レベルでフリーズ動作を制御する機能を追加しました。
- AutoSupport プロキシ接続の設定のサポートを追加しました。
- データ ムーバー暗号化（Kopia / Restic）のシークレットを定義する機能が追加されました。
- 実行フックを手動で実行する機能を追加しました。

- Trident Protect のインストール時にセキュリティ コンテキスト制約（SCC）を設定する機能を追加しました。
- Trident Protect のインストール時に nodeSelector を設定するためのサポートが追加されました。
- AppVaultオブジェクトのHTTP / HTTPS出力プロキシのサポートを追加しました。
- 拡張されたResourceFilterにより、クラスタースコープのリソースの除外が可能になりました。
- S3 AppVaultクレデンシャルでのAWSセッショントークンのサポートを追加しました。
- スナップショット前実行フック後のリソース収集のサポートが追加されました。

修正

- 一時ボリュームの管理を改善し、ONTAP ボリューム リカバリ キューをスキップするようにしました。
- SCC 注釈が元の値に復元されました。
- 並列操作のサポートにより復元効率が向上しました。
- 大規模アプリケーションの実行フックのタイムアウトのサポートが強化されました。

24.10.1 の変更点

機能強化

- **Kubernetes**：Kubernetes 1.32 のサポートが追加されました。
- iSCSI セッションがログインする必要があるのにログインされていない場合の iSCSI 接続状態の検出とログ記録を追加しました（"問題 #961"）。

修正

- 自動エクスポートポリシーから欠落しているノードIPアドレスを修正しました（"問題 #965"）。
- ONTAP-NAS-Economy の自動エクスポート ポリシーがボリュームごとのポリシーに途中で切り替わる問題を修正しました。
- CVE-2024-45337 および CVE-2024-45310 に対処するために Trident および Trident-ASUP の依存関係を更新しました。
- iSCSI 自己修復中に断続的に正常でない非 CHAP ポータルのログアウトを削除しました（"問題 #961"）。

24.10の変更点

機能強化

- 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、ゾーン対応のプロビジョニングをサポートするようになりました。
- GCP Workload Identity は、GKE を使用する Google Cloud NetApp Volumes の Cloud Identity として使用されます。
- `formatOptions`ONTAP-SAN および ONTAP-SAN-Economy ドライバーに構成パラメータを追加し、ユーザーが LUN フォーマット オプションを指定できるようにしました。
- Azure NetApp Files の最小ボリューム サイズを 50 GiB に削減しました。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。

- `denyNewVolumePools`構成パラメータを追加して、ONTAP-NAS-Economy および ONTAP-SAN-Economy ドライバーを既存の Flexvol プールに制限しました。
- すべてのONTAPドライバーで、SVMからのアグリゲートの追加、削除、または名前変更の検出が追加されました。
- 報告された PVC サイズが使用可能であることを保証するために、LUKS LUN に 18 MiB のオーバーヘッドを追加しました。
- ONTAP-SAN および ONTAP-SAN-Economy ノードのステージおよびアンステージの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ーを追加し、顧客がONTAPでTrident用の最小限のロールを作成できるようにしました。
- トラブルシューティングのための追加ログを追加しました `lsscsi`（"問題 #792"）。

Kubernetes

- Kubernetes ネイティブ ワークフローの新しい Trident 機能を追加：
  - データ保護
  - データ移行
  - ディザスタ リカバリ
  - アプリケーションモビリティ

    "Trident Protectの詳細はこちら"。
- Trident が Kubernetes API サーバーと通信する際に使用する QPS 値を設定するための新しいフラグ `--k8s-api-qps`をインストーラーに追加しました。
- `--node-prep`フラグをインストーラーに追加し、Kubernetesクラスター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できるようにしました。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しました。
- 非正常なノードシャットダウンのシナリオ中に ONTAP-NAS-Economy ボリュームを強制的にデタッチするサポートが追加されました。
- 新しいONTAP-NAS-Economy NFSボリュームは、 `autoExportPolicy`バックエンドオプションを使用する際にqtreeごとのエクスポートポリシーを使用します。アクセス制御とセキュリティを向上させるために、公開時にはqtreeはノード制限エクスポートポリシーにのみマップされます。既存のqtreeは、Tridentがアクティブなワークロードに影響を与えずにすべてのノードからボリュームを公開解除する際に、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 のサポートが追加されました。

実験的な機能強化

- ONTAP-SAN ドライバーの Fibre Channel サポートのテクニカル プレビューを追加しました。

修正

- **Kubernetes**：
  - Rancher アドミッション Webhook が Trident Helm インストールを妨げる問題を修正（"問題 #839"）。

- helmチャート値のAffinityキーを修正しました（"問題 #898"）。
- 修正済み tridentControllerPluginNodeSelector/tridentNodePluginNodeSelector は「true」値では動作しません（"問題 #899"）。
- クローン作成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リポジトリで`go mod tidy`を修正（"問題 #767"）。

廃止予定

- **Kubernetes**：
  - サポートされる最小 Kubernetes を 1.25 に更新しました。
  - POD セキュリティ ポリシーのサポートが削除されました。

製品のリブランディング

24.10リリース以降、Astra TridentはTrident（NetApp Trident）にリブランドされます。このリブランド変更は、Tridentの機能、サポートされるプラットフォーム、または相互運用性には影響しません。

24.06の変更点

機能強化

- 重要：`limitVolumeSize`パラメータは、ONTAP economyドライバのqtree/LUNサイズを制限するようになりました。新しい`limitVolumePoolSize`パラメータを使用して、これらのドライバのFlexvolサイズを制御します。（"問題 #341"）。
- iSCSI自己修復機能に、廃止されたigroupが使用されている場合に正確なLUN IDでSCSIスキャンを開始する機能を追加しました（"問題 #883"）。
- バックエンドがサスペンド モードの場合でも、ボリューム クローンおよびサイズ変更操作を許可するためのサポートが追加されました。
- Tridentコントローラのユーザー設定ログ設定をTridentノードポッドに伝播する機能を追加しました。
- Tridentに、ONTAPバージョン9.15.1以降でONTAPI（ZAPI）の代わりにRESTをデフォルトで使用するサポートを追加しました。
- 新しい永続ボリュームの ONTAP ストレージ バックエンドに、カスタム ボリューム名とメタデータのサポートを追加しました。
- `azure-netapp-files`（ANF）ドライバを強化し、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、デフォルトでスナップショットディレクトリを自動的に有効にするようにしました。
- NFSボリュームに対するBottlerocketサポートが追加されました。
- Google Cloud NetApp Volumes のテクニカル プレビュー サポートを追加しました。

Kubernetes

- Kubernetes 1.30 のサポートが追加されました。
- Trident DaemonSetが起動時にゾンビマウントと残留追□跡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 `trident.netapp.io/luksEncryption`を追加しました（"問題 #849"）。
- ANF ドライバーにトポロジ認識を追加しました。
- Windows Server 2022 ノードのサポートが追加されました。

修正

- 古いトランザクションが原因で発生していた Trident のインストール失敗を修正しました。
- Kubernetesからの警告メッセージを無視するようにtridentctlを修正しました（"問題 #892"）。
- Tridentコントローラ `SecurityContextConstraint`の優先度を `0`に変更しました（"問題 #887"）。
- ONTAPドライバーは20 MiB以下のボリュームサイズを受け入れるようになりました（"問題[#885"）。
- Tridentを修正し、ONTAP-SANドライバーのサイズ変更操作中にFlexVolボリュームが縮小するのを防ぎました。
- NFS v4.1 での ANF ボリュームのインポート失敗を修正しました。

24.02の変更点

機能強化

- Cloud Identity のサポートが追加されました。
  - ANF を使用した AKS - Azure Workload Identity がクラウド ID として使用されます。
  - FSxN を使用した EKS - AWS IAM ロールがクラウド ID として使用されます。
- EKS コンソールから EKS クラスターのアドオンとして Trident をインストールするためのサポートを追加しました。
- iSCSI self-healingを設定および無効化する機能を追加（"問題 #864"）。
- Amazon FSxパーソナリティをONTAPドライバーに追加し、AWS IAMおよびSecretsManagerとの統合を可能にし、Tridentがバックアップ付きのFSxボリュームを削除できるようにしました（"問題 #453"）。

Kubernetes

- Kubernetes 1.29 のサポートが追加されました。

修正

- ACP が有効になっていない場合の ACP 警告メッセージを修正しました（"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ドライバー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秒の遅延を追加しました。

廃止予定

- マルチプラットフォーム イメージ マニフェストから in-toto 構成証明フレームワークを削除しました。

23.10の変更点

修正

- ontap-nas および ontap-nas-flexgroup ストレージ ドライバーで、新しく要求されたサイズが合計ボリューム サイズより小さい場合のボリューム拡張を修正しました（"問題 #834"）。
- ontap-nas および ontap-nas-flexgroup ストレージ ドライバー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ようにボリューム サイズを修正しました（"問題 #722"）。
- ONTAP-NAS-Economy の FlexVol 名変換を修正しました。
- Windows ノードの再起動時に発生していた Trident 初期化の問題を修正しました。

機能強化

Kubernetes

Kubernetes 1.28 のサポートが追加されました。

Trident

- azure-netapp-files ストレージ ドライバーで 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ドライバのNVMe over TCPのサポートが追加されました。
- バックエンドがユーザーによって一時停止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を追加しました（"問題 #558"）。

23.07.1 の変更点

Kubernetes: ゼロダウンタイムアップグレードをサポートするためにデーモンセットの削除を修正しました（"問題 #740"）。

23.07の変更点

修正

Kubernetes

- Tridentのアップグレードで、終了状態のままになっている古いポッドを無視するように修正しました（"問題 #740"）。
- 「transient-trident-version-pod」定義に許容範囲を追加しました（"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に 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 ONTAPI（ZAPI）要求を修正しました。
- ストレージドライバコードのエラー処理を修正しました（"問題 #816"）。
- ONTAP ドライバーで use-rest=true を使用する場合のクォータサイズ変更を修正しました。
- ontap-san-economy での LUN クローン作成を修正しました。

- 公開情報フィールドを `rawDevicePath`から `devicePath`に戻す； `devicePath`フィールドを入力および回復（場合によっては）するためのロジックを追加。

機能強化

Kubernetes

- 事前にプロビジョニングされたスナップショットのインポートのサポートが追加されました。
- 最小限のデプロイメントとデーモンセットのLinux権限（"問題 #817"）。

Trident

- 「オンライン」ボリュームとスナップショットの状態フィールドは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（"問題 #801"、"#543"）、バックエンドの状態を更新します。
- LUNシリアル番号は常に取得され、ControllerVolumePublishワークフロー中に公開されます。
- 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ための追加のロジックを追加しました。
- 正しいマルチパス デバイスがアンステージングされていることを確認するための iSCSI ボリュームの追加検証。

実験的な機能強化

ONTAP-SAN ドライバーの NVMe over TCP のテクニカルプレビューサポートが追加されました。

ドキュメント

多くの構成とフォーマットの改善が行われました。

廃止予定

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。
- サポートされる最小 Kubernetes を 1.22 に更新しました。

23.04の変更点

> ONTAP-SAN-* ボリュームの強制ボリュームデタッチは、非正常ノードシャットダウン機能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。強制デタッチは、`--enable-force-detach` Trident インストーラフラグを使用してインストール時に有効にする必要があります。

修正

- 仕様で指定されている場合、インストールに IPv6 ローカルホストを使用するように Trident Operator を修正しました。
- Trident Operatorクラスタロール権限をバンドル権限と同期するように修正しました（"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際の問題を修正しました。
- SMB ボリュームのFlexGroupクローニングサポートとボリュームインポートを修正しました。
- Tridentコントローラがすぐにシャットダウンできなかった問題を修正しました（"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。
- 外部プロセスが完了まで実行できるように修正を追加しました。
- s390アーキテクチャのコンパイルエラーを修正しました（"問題#537"）。
- ボリュームマウント操作中の不正なログレベルを修正しました（"問題 #781"）。
- 潜在的な型アサーションエラーを修正しました（"問題 #802"）。

機能強化

- Kubernetes：
  - Kubernetes 1.27 のサポートが追加されました。
  - 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を追加しました。
  - 非正常なノードシャットダウンのシナリオ中に ONTAP-SAN-* ボリュームの強制デタッチをサポートするようになりました。
  - すべての ONTAP-SAN-* ボリュームはノードごとの igroup を使用するようになりました。セキュリティ体制を強化するために、LUN はそれらのノードにアクティブに公開されている間のみ igroup にマップされます。既存のボリュームは、Trident がアクティブなワークロードに影響を与えずに実行しても安全であると判断した場合、新しい igroup スキームに自動的に切り替えられます（"問題 #758"）。
  - ONTAP-SAN-* バックエンドから未使用の Trident 管理 igroup をクリーンアップすることで、Trident のセキュリティを強化しました。
- Amazon FSx の SMB ボリュームのサポートを ontap-nas-economy および ontap-nas-flexgroup ストレージドライバに追加しました。
- ontap-nas、ontap-nas-economy、ontap-nas-flexgroup ストレージドライバーによる 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（"問題 #732"）。
- 改良されたTridentシャットダウン手順は、まずAPIサーバーを非アクティブ化することによって行われます（"問題 #811"）。
- Makefileに、WindowsおよびArm64ホスト用のクロスプラットフォームビルドサポートを追加しました。BUILD.mdを参照してください。

廃止予定

Kubernetes: ontap-san および ontap-san-economy ドライバーを構成するときに、バックエンドスコープの igroup が作成されなくなりました（"問題 #758"）。

23.01.1 の変更点

修正

- 仕様で指定されている場合、インストールに IPv6 ローカルホストを使用するように Trident Operator を修正しました。
- Trident Operatorクラスタロール権限をバンドル権限と同期するように修正しました"問題 #799"。
- 外部プロセスが完了まで実行できるように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際の問題を修正しました。
- SMB ボリュームのFlexGroupクローニングサポートとボリュームインポートを修正しました。

23.01の変更点

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

修正

- Kubernetes：Helm経由のTridentインストールを修正するために、ポッドセキュリティポリシーの作成を除外するオプションを追加しました（"問題 #783、#794"）。

機能強化

Kubernetes

- Kubernetes 1.26 のサポートが追加されました。
- 全体的な Trident RBAC リソース使用率の向上（"問題#757"）。
- ホストノード上の壊れたまたは古くなった iSCSI セッションを検出して修正するための自動化を追加しました。
- 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サポートが追加されました。

Trident

- ontap-nas ストレージドライバに Amazon FSx for NetApp ONTAP を使用した SMB ボリュームのサポートを追加しました。
- SMB ボリュームを使用する際の NTFS アクセス許可のサポートが追加されました。
- CVS サービスレベルの 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。
- ontap-nas-flexgroupストレージドライバでFlexGroupsを作成する際に、flexgroupAggregateListのオプション使用がサポートされました。
- 複数のFlexVolボリュームを管理する際のontap-nas-economy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のデータ LIF 更新を有効にしました。
- Trident 展開と DaemonSet の命名規則を更新し、ホストノードの OS を反映するようにしました。

廃止予定

- Kubernetes：サポートされる最小 Kubernetes を 1.21 に更新しました。
- `ontap-san`または `ontap-san-economy`ドライバを設定する際、DataLIFを指定する必要がなくなりました。

22.10の変更点

Trident 22.10にアップグレードする前に、以下の重要な情報を必ずお読みください。

Trident 22.10に関する重要な情報

- Kubernetes 1.25 が Trident で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。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制するようになりました。推奨値は multipath.conf ファイル内の `find_multipaths: no`です。

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

修正

- `credentials`フィールドを使用して作成されたONTAPバックエンドが22.07.0アップグレード中にオンラインにならない問題を修正しました（"問題#759"）。
- **Docker**： 一部の環境でDockerボリュームプラグインが起動に失敗する問題を修正しました（"問題#548"および"問題 #760"）。
- ONTAP SAN バックエンドに固有の SLM の問題を修正し、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しました。
- ボリュームを接続するときに iSCSI LUN の不要なスキャンが発生するパフォーマンスの問題を修正しました。
- Trident iSCSI ワークフロー内の細分化された再試行を削除し、フェイルファーストを実現して外部再試行間隔を短縮しました。
- 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ときにiSCSIデバイスをフラッシュするとエラーが返される問題を修正しました。

機能強化

- Kubernetes：
  - Kubernetes 1.25 のサポートが追加されました。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。
  - 将来の権限拡張を可能にするために、Trident Deployment および DaemonSet 用に、個別の ServiceAccount、ClusterRole、ClusterRoleBinding を追加しました。
  - "クロスネームスペースボリューム共有"のサポートを追加。
- すべての Trident `ontap-*` ストレージドライバが ONTAP REST API で動作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`） `PodSecurityPolicy`なしで Kubernetes

1.25 をサポートします。

- "LUKS暗号化ボリュームのサポート"を `ontap-san`および `ontap-san-economy`ストレージドライバに追加しました。
- Windows Server 2019 ノードのサポートが追加されました。
- "Windows ノード上の SMB ボリュームのサポート"を `azure-netapp-files`ストレージドライバ経由で追加しました。
- ONTAP ドライバーの自動 MetroCluster スイッチオーバー検出が一般提供されるようになりました。

廃止予定

- **Kubernetes**：サポートされる最小 Kubernetes を 1.20 に更新しました。
- Astra Data Store（ADS）ドライバーを削除しました。
- `yes`および `smart`オプションのサポートを削除しました（ `find_multipaths`）。iSCSI のワーカーノードマルチパスを設定する場合に使用します。

22.07の変更点

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問題を修正しました。（"GitHub 問題 #700"）
- 非 CHAP パスからのエラー処理の問題を修正し、失敗した場合に kubelet が再試行するようにしました。"GitHub 問題 #736")

機能強化

- 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 に移行する
- ONTAP-SAN ボリュームは、ノードごとの igroup を使用するようになり、セキュリティ体制を強化するために、それらのノードにアクティブに公開されている間のみ LUN を igroup にマッピングします。既存のボリュームは、Trident がアクティブなワークロードに影響を与えずに実行しても安全であると判断した場合、新しい igroup スキームに自動的に切り替えられます。
- Trident のインストール時に ResourceQuota を含め、デフォルトで PriorityClass の消費が制限されている場合でも Trident の DaemonSet がスケジュールされるようにしました。
- Azure NetApp Files ドライバーにネットワーク機能のサポートを追加しました。("GitHub 問題 #717")
- ONTAPドライバーにテクニカルプレビューの自動MetroClusterスイッチオーバー検出を追加しました。（"GitHub 問題 #228"）

廃止予定

- **Kubernetes**：サポートされる最小 Kubernetes を 1.19 に更新しました。
- バックエンド構成では、単一の構成で複数の認証タイプが許可されなくなりました。

削除

- AWS CVS ドライバー（22.04 以降非推奨）は削除されました。
- Kubernetes
  - ノードポッドから不要な SYS_ADMIN 機能を削除しました。
  - ノード準備（nodeprep）を単純なホスト情報とアクティブなサービス検出にまで削減し、ワーカーノードでNFS/iSCSIサービスが利用可能である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、Tridentのインストール時に有効になる権限について詳しく説明しています。

22.04の変更点

NetAppは、製品とサービスを継続的に改善し、強化しています。Tridentの最新機能の一部を以下に示します。以前のリリースについては、 "以前のバージョンのドキュメント"を参照してください。

> 以前の Trident リリースからアップグレードし、Azure NetApp Files を使用している場合、`location` 設定パラメータは必須のシングルトンフィールドになりました。

修正

- iSCSI イニシエーター名の解析が改善されました。（"GitHub issue ＃681"）
- CSI ストレージ クラス パラメータが許可されない問題を修正しました。（"GitHub 問題 #598"）
- Trident CRDで重複キー宣言を修正しました。（"GitHub issue #671"）
- 不正確な CSI スナップショット ログを修正しました。（"GitHub イシュー＃629"））
- 削除されたノード上のボリュームの非公開に関する問題を修正しました。（"GitHub issue #691"）
- ブロックデバイス上のファイルシステムの不整合の処理を追加しました。（"GitHub issue ＃656"）
- インストール中に `imageRegistry`フラグを設定する際の自動サポートイメージのプル問題を修正しました。（"GitHub issue ＃715"）
- 複数のエクスポートルールを持つボリュームのクローニングに Azure NetApp Files ドライバーが失敗する問題を修正しました。

機能強化

- 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、最低でもTLS 1.3が必要になりました。("GitHub issue #698")
- Tridentは、セキュアエンドポイントからの応答にHSTSヘッダーを追加するようになりました。
- Tridentは、Azure NetApp FilesのUNIX権限機能を自動的に有効にしようとします。
- **Kubernetes**：Trident daemonset は system-node-critical 優先度クラスで実行されるようになりました。（"GitHub イシュー＃694"）

削除

E-Series ドライバー（20.07 以降無効）が削除されました。

22.01.1 の変更点

修正

- 削除されたノード上のボリュームの非公開に関する問題を修正しました。（"GitHub issue #691"）
- ONTAP API応答で集計スペースのnilフィールドにアクセスする際に発生するパニックを修正しました。

22.01.0 の変更点

修正

- **Kubernetes:** 大規模クラスタのノード登録バックオフ再試行時間を増やします。
- 同じ名前の複数のリソースによって azure-netapp-files ドライバーが混乱する可能性がある問題を修正しました。
- ONTAP SAN IPv6 DataLIF は、括弧で指定した場合に機能するようになりました。
- すでにインポート済みのボリュームをインポートしようとすると EOF が返され、PVC が保留状態になる問題を修正しました。（"GitHub 問題 #489"）
- SolidFireボリューム上で32個を超えるスナップショットが作成された場合にTridentのパフォーマンスが低下する問題を修正しました。
- SSL 証明書の作成で SHA-1 を SHA-256 に置き換えました。
- Azure NetApp Files ドライバを修正し、重複するリソース名を許可し、操作を単一の場所に制限するようにしました。
- Azure NetApp Files ドライバを修正し、重複するリソース名を許可し、操作を単一の場所に制限するようにしました。

機能強化

- Kubernetes の機能強化：
  - Kubernetes 1.23 のサポートが追加されました。
  - Trident OperatorまたはHelmを使用してインストールする場合のTridentポッドのスケジュールオプションを追加します。（"GitHub issue #651"）
- GCP ドライバーでクロスリージョン ボリュームを許可します。（"GitHub issue #633"）
- Azure NetApp Files ボリュームに「unixPermissions」オプションのサポートを追加しました。（"GitHub issue #666"）

廃止予定

Trident RESTインターフェースは127.0.0.1または[::1]アドレスでのみリッスンおよびサービスできます

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コントローラがCrashLoopBackOff状態になる問題があります。この問題はv21.10.1で修正されました（GitHub問題669）。

修正

- GCP CVS バックエンドでボリュームをインポートするときに発生する可能性のある競合状態を修正し、インポートの失敗を解消しました。
- ノードが削除され、その後Kubernetesクラスタに再度追加されたときに、TridentコントローラーがCrashLoopBackOff状態になる問題を修正しました（GitHub問題669）。
- SVM名が指定されていない場合にSVMが検出されなくなる問題を修正しました（GitHub問題612）。

21.10.0 の変更点

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問題を修正しました（GitHub issue 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しました（GitHub issue 597）。
- Kubernetes 関連の修正：
  - `ontap-nas`および `ontap-nas-flexgroup`ドライバーでスナップショットを作成する際に、ボリュームの使用済みスペースを最小restoreSizeとして返します（GitHub問題645）。
  - ボリュームのサイズ変更後に `Failed to expand filesystem`エラーが記録される問題を修正しました（GitHub issue 560）。
  - ポッドが `Terminating`状態でスタックする問題を修正しました（GitHub issue 572）。
  - `ontap-san-economy` FlexVolがスナップショットLUNでいっぱいになる可能性があるケースを修正しました（GitHub問題533）。
  - 異なるイメージでのカスタム YAML インストーラの問題を修正しました（GitHub issue 613）。
  - スナップショットサイズの計算を修正（GitHub issue 611）。
  - すべてのTridentインストーラーがプレーンKubernetesをOpenShiftとして識別できる問題を修正しました（GitHub問題639）。
  - Kubernetes APIサーバーにアクセスできない場合に調整を停止するようにTridentオペレーターを修正しました（GitHub問題599）。

機能強化

- GCP-CVS パフォーマンスボリュームの `unixPermissions`オプションのサポートを追加しました。
- GCP の 600 GiB ～ 1 TiB の範囲のスケール最適化 CVS ボリュームのサポートが追加されました。
- Kubernetes 関連の機能強化：
  - Kubernetes 1.22 のサポートが追加されました。
  - Kubernetes 1.22で動作するTridentオペレーターとHelmチャートを有効にしました（GitHub issue 628

）。

- オペレータイメージを `tridentctl`imagesコマンドに追加しました（GitHub issue 570）。

実験的な機能強化

- `ontap-san`ドライバで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`、 `ontap-nas-economy`ドライバーに*テクニカルプレビュー* RESTサポートを追加しました。

既知の問題

既知の問題では、製品の正常な使用を妨げる可能性のある問題が特定されます。

- 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、クラスタをアップグレードする前に、values.yamlを更新して `excludePodSecurityPolicy`を `true`に設定するか、 `--set excludePodSecurityPolicy=true`を `helm upgrade`コマンドに追加する必要があります。
- Tridentは、StorageClassで `fsType`が指定されていないボリュームに対して、空白の `fsType` (`fsType=""`を強制します。Kubernetes 1.17以降で作業する場合、TridentはNFSボリュームに対して空白の `fsType`を指定することをサポートしています。iSCSIボリュームの場合、Security Contextを使用して `fsGroup`を強制する際には、StorageClassで `fsType`を設定する必要があります。
- 複数のTridentインスタンスでバックエンドを使用する場合、各バックエンド設定ファイルには、ONTAPバックエンドの場合は異なる `storagePrefix`値を、SolidFireバックエンドの場合は異なる `TenantName`を指定する必要があります。Tridentは、他のTridentインスタンスが作成したボリュームを検出できません。ONTAPまたはSolidFireバックエンドで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がボリューム作成をべき等操作として扱うためです。 `storagePrefix`または `TenantName`が異なっていない場合、同じバックエンドで作成されたボリュームの名前が競合する可能性があります。
- Tridentをインストールする場合（ `tridentctl`またはTrident Operatorを使用）、 `tridentctl`を使用してTridentを管理する場合は、 `KUBECONFIG`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。これは、 `tridentctl`が動作する対象のKubernetesクラスタを示すために必要です。複数のKubernetes環境で作業する場合は、 `KUBECONFIG`ファイルが正確にソース化されていることを確認する必要があります。
- iSCSI PVのオンラインスペース再利用を実行するには、ワーカーノード上の基盤となるOSで、ボリュームにマウントオプションを渡す必要がある場合があります。これはRHEL/Red Hat Enterprise Linux CoreOS（RHCOS）インスタンスにも当てはまり、 `discard` "マウントオプション"；discard mountOptionが[`StorageClass`^]に含まれていることを確認して、オンラインブロック破棄をサポートします。
- Kubernetesクラスタごとに複数のTridentインスタンスがある場合、Tridentは他のインスタンスと通信できず、それらが作成した他のボリュームを検出できないため、クラスタ内で複数のインスタンスが実行されると予期しない不正な動作が発生します。Kubernetesクラスタごとに1つのTridentインスタンスのみが存在する必要があります。
- Tridentがオフラインの間にTridentベースの `StorageClass`オブジェクトがKubernetesから削除された場合、Tridentがオンラインに戻ったときに、データベースから対応するストレージクラスを削除しません。これらのストレージクラスは、 `tridentctl`またはREST APIを使用して削除する必要があります。
- ユーザーが対応するPVCを削除する前にTridentによってプロビジョニングされたPVを削除した場合、Tridentはバッキングボリュームを自動的に削除しません。ボリュームは `tridentctl`またはREST APIを使用して削除する必要があります。
- ONTAPでは、アグリゲートのセットが各プロビジョニング要求に対して一意でない限り、複数のFlexGroupを同時にプロビジョニングすることはできません。

- IPv6でTridentを使用する場合は、 `managementLIF`と `dataLIF`をバックエンド定義内で角括弧で囲んで指定する必要があります。例：`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。

> ONTAP SAN バックエンドでは `dataLIF`を指定できません。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。

- `solidfire-san`ドライバーをOpenShift 4.5で使用する場合は、基盤となるワーカーノードがCHAP認証アルゴリズムとしてMD5を使用していることを確認してください。Element 12.7では、安全なFIPS準拠のCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256が利用できます。

詳細情報の参照

- "Trident GitHub"
- "Tridentブログ"

以前のバージョンのドキュメント

Trident 25.10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"Tridentサポートライフサイクル"に基づいて利用できます。

- "Trident 25.06"
- "Trident 25.02"
- "Trident 24.10"
- "Trident 24.06"
- "Trident 24.02"
- "Trident 23.10"
- "Trident 23.07"
- "Trident 23.04"
- "Trident 23.01"

ONTAP ASA r2ストレージシステムに対するNetApp Tridentのサポート

NetApp Trident 25.02以降では、NetApp ASA r2システムをストレージバックエンドとしてサポートしています。詳細については、"ASA r2システム"を参照してください。

ASA r2システムには `ontap-san`ドライバが必要です。TridentはASA r2システムの `ontap-san-economy`ドライバをサポートしていません。

バックエンド構成で `ontap-san`を `storageDriverName`として指定すると、Tridentは自動的にASA r2ストレージシステムを検出します。

Tridentは、Trident protectを使用してASA r2システムに限定的なデータ保護を提供します。

サポートされているSANプロトコルは、Tridentのバージョンによって異なります：

- Trident 25.02以降はiSCSIをサポートします。
- Trident 25.06以降では、iSCSIに加えてNVMe/TCPもサポートされます。

ONTAPバックエンドストレージ用のStorage Virtual Machine（SVM）に少なくとも1つのアグリゲートを割り当てる必要があります。手順については"ASA r2システムでのSVMへのアグリゲートの割り当て"を参照してください。

サポートされている操作

- 永続ボリューム（PV）のプロビジョニング
- 動的ボリュームプロビジョニング
- ボリュームの作成と削除
- ボリュームのクローニング
- ボリュームの拡張
- ストレージクラスの管理

サポートされていない操作

- LUKS暗号化
- SnapMirrorボリューム レプリケーション
- アグリゲートの使用量の制限
- スペース予約モード
- Snapshot 数
- 階層化

詳細については、"ONTAP SAN 構成オプションと例"を参照してください。

既知の問題

今回のリリースの動作に悪影響を及ぼす可能性がある既知の問題が記載されています。

現在のリリースには次の既知の問題が影響します。

大きなファイルの**Restic**バックアップのリストアが失敗する可能性がある

Resticを使用して作成されたAmazon S3バックアップから30GB以上のファイルを復元する場合、復元操作が失敗する可能性があります。回避策として、データムーバーとしてKopiaを使用してデータをバックアップします（Kopiaはバックアップのデフォルトのデータムーバーです）。手順については "Trident Protectを使用してアプリケーションを保護"を参照してください。

はじめに

Tridentの詳細

Tridentの詳細

Tridentは、NetAppが管理する完全にサポートされた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ースを使用して、コンテナ化されたアプリケーションの永続性の要求を満たすように設計されています。

Tridentとは

NetApp Trident は、パブリック クラウドまたはオンプレミスにあるすべての一般的な NetApp ストレージ プラットフォーム（オンプレミス ONTAP クラスタ（AFF、FAS、ASA）、ONTAP Select、Cloud Volumes ONTAP、Element ソフトウェア（NetApp HCI、SolidFire）、Azure NetApp Files、Amazon FSx for NetApp ONTAP を含む）全体でストレージ リソースの消費と管理を可能にします。

Tridentは、Container Storage Interface（CSI）に準拠した動的ストレージオーケストレーターであり、"Kubernetes"とネイティブに統合されています。Tridentは、クラスター内の各ワーカーノード上で、単一のControllerポッドとNodeポッドとして実行されます。詳細については、"Tridentアーキテクチャ"を参照してください。

Tridentは、NetAppストレージプラットフォーム向けのDockerエコシステムとの直接統合も提供しています。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、"Docker用Tridentを導入する"を参照してください。

> Kubernetesを初めて使用する場合は、"Kubernetesの概念とツール"に慣れておく必要があります。

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側面と統合され、高度なデータ管理機能を提供することで、Kubernetes デプロイメントの機能性、能力、パフォーマンス、可用性を強化します。

Amazon FSx for NetApp ONTAP

"Amazon FSx for NetApp ONTAP"は、NetApp ONTAPストレージオペレーティングシステムを基盤とするファイルシステムを起動して実行できる、完全に管理されたAWSサービスです。

Azure NetApp Files

"Azure NetApp Files"は、NetAppを基盤と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。NetAppに期待されるパフォーマンスと豊富なデータ管理機能により、最も要求の厳しいファイルベースのワークロードを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のストレージ アプライアンスです。

Google Cloud NetApp Volumes

"Google Cloud NetApp Volumes"は、Google Cloudのフルマネージドファイルストレージサービスであり、ハイパフォーマンスで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"を使用すると、ストレージ管理者は、パフォーマンスを保証し、簡素化され合理化されたストレージ フットプリントを実現することで、ワークロードを統合できます。

NetApp HCI

"NetApp HCI"は、日常的なタスクを自動化し、インフラ管理者がより重要な機能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素化します。

Trident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対して直接プロビジョニングおよび管理できます。

NetApp ONTAP

"NetApp ONTAP"はNetAppのマルチプロトコルの統合ストレージ 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、またはオール HDD 構成があり、さまざまな導入モデルが提供されています（オンプレミス FAS 、 AFF 、 ASA クラスタ、 ONTAP Select 、 Cloud Volumes ONTAP ）。Trident は、これらの ONTAP 導入モデルをサポートしています。

Tridentアーキテクチャ

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

コントローラポッドとノードポッドについて

Tridentは、Kubernetesクラスタ上で単一のTridentコントローラポッドと1つ以上のTrident ノードポッドとして展開され、標準のKubernetes _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"は、Kubernetes Storageコミュニティによってメンテナンスされています。

Kubernetes "ノードセレクター" および "tolerations と taints" は、ポッドを特定のノードまたは優先ノードで実行するように制限するために使用されます。Trident のインストール時に、コントローラーとノードポッド

のノードセレクターと許容値を設定できます。

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

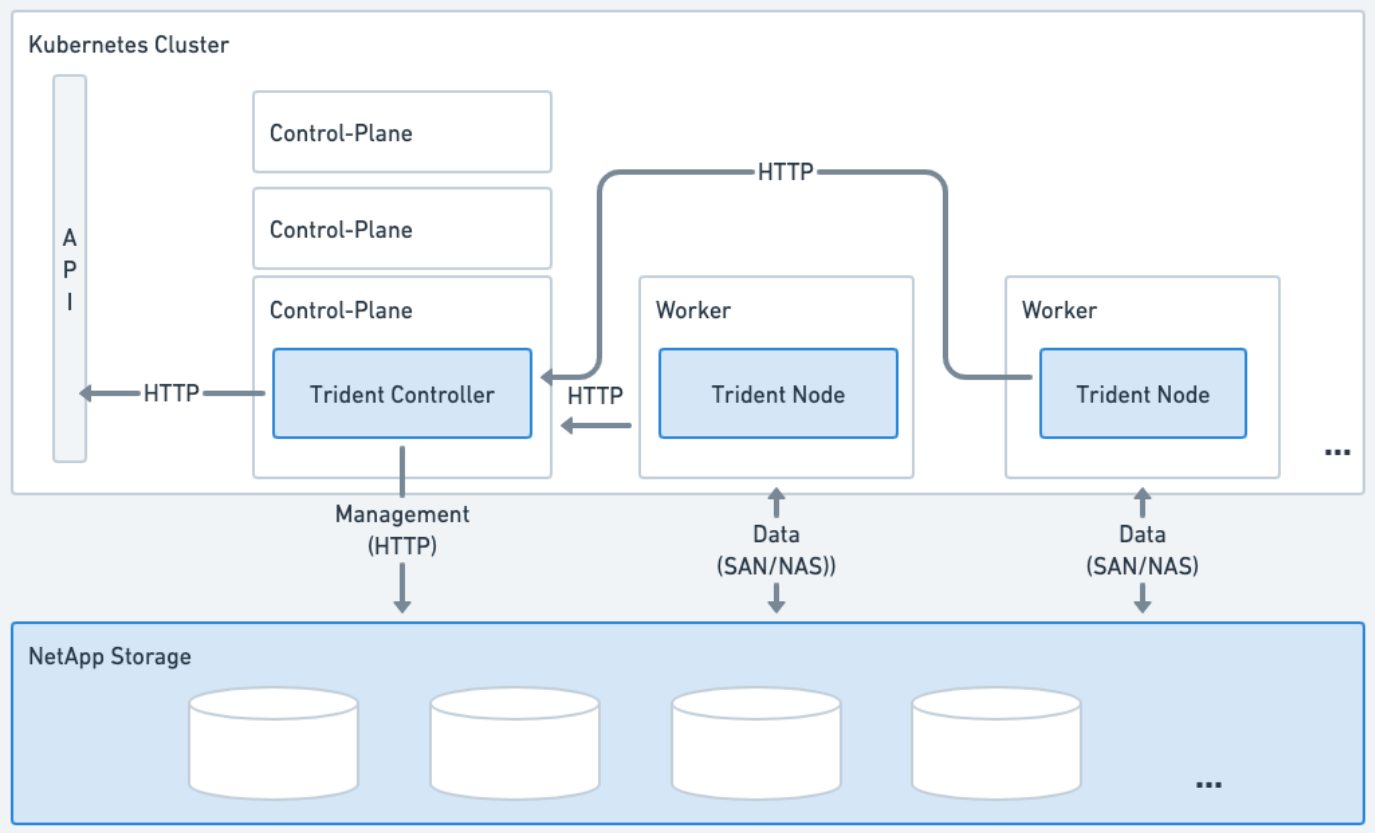


図 **1. Kubernetes** クラスタに **Trident** を導入済み

Tridentコントローラポッド

Trident Controller Pod は、CSI Controller プラグインを実行する単一のポッドです。

- NetApp ストレージでのボリュームのプロビジョニングと管理を担当
- Kubernetes Deployment によって管理
- 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。

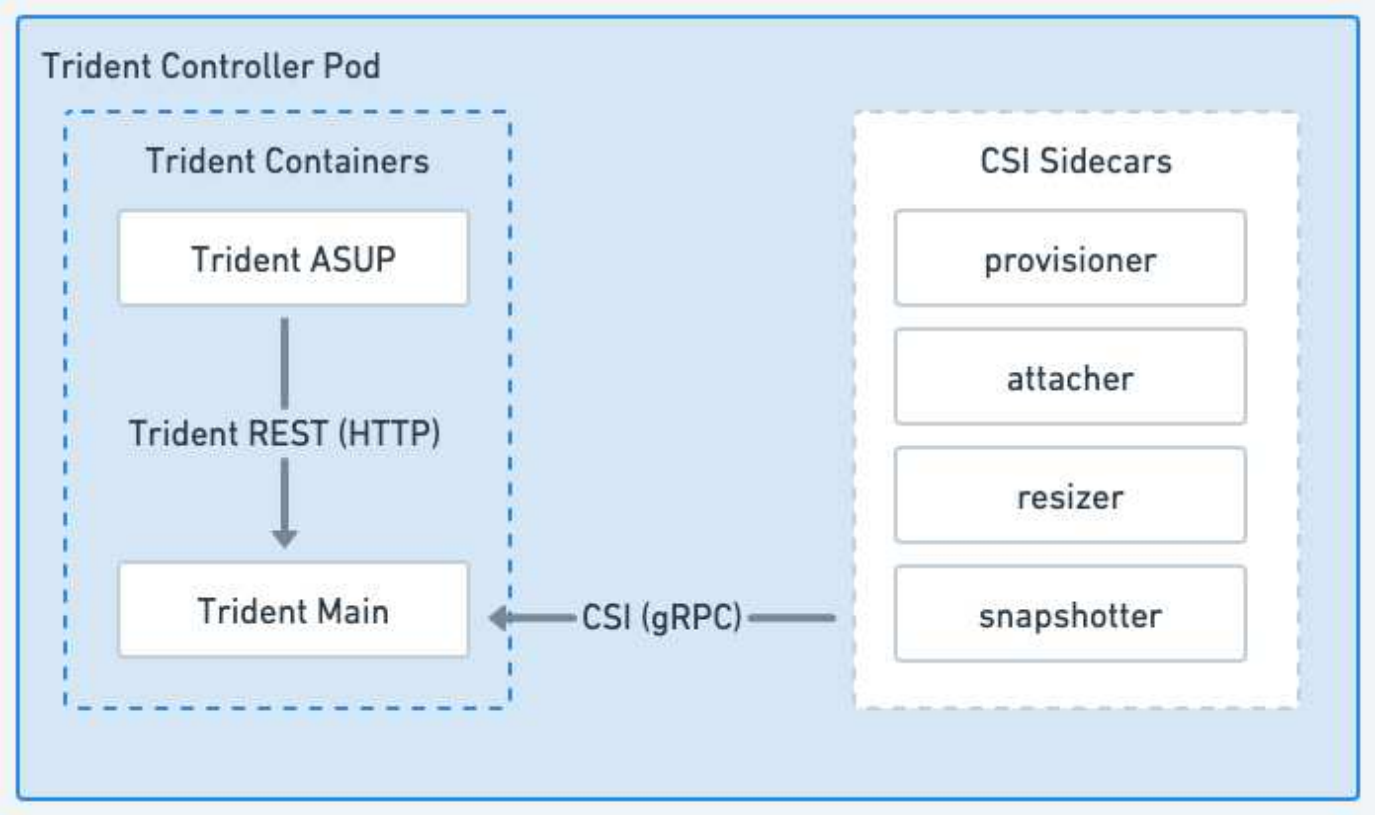


図 **2. Trident** コントローラポッドの図

Trident ノードポッド

Trident Node Pod は、CSI Node プラグインを実行する特権 Pod です。

- ホスト上で実行されている 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 DaemonSet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ります

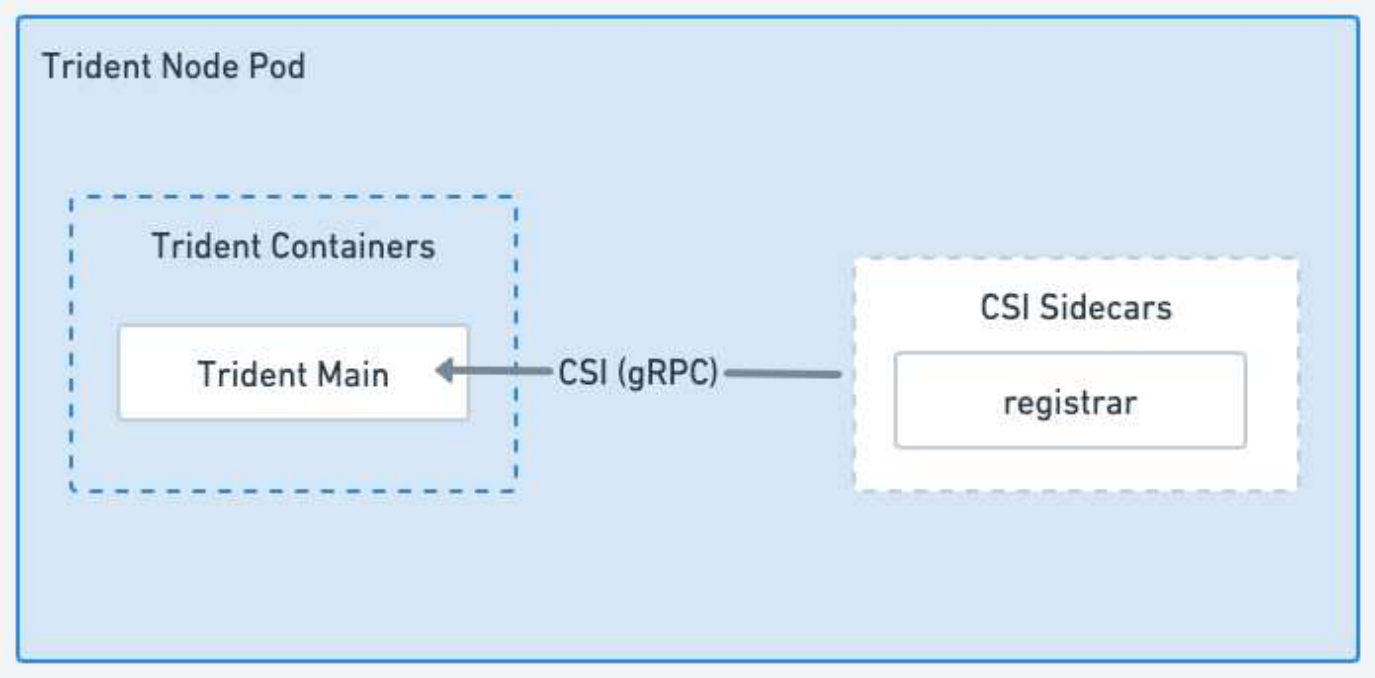


図 **3. Trident** ノードポッド図

サポートされている **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ています：

Kubernetes クラスタアーキテクチャ	サポート	デフォルトインストール
シングルマスター、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`、計算	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 つの主要な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ます。これは、プロビジョニング前の必要な準備として実行されます。2 番目のフェーズにはボリュームの作成自体が含まれ、保留中のボリュームのストレージクラスに関連付けられているストレージプールから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

storagePools、 additionalStoragePools、 `excludeStoragePools`リストの両方に依存します。ストレージクラスを作成すると、Tridentは各バックエンドが提供する属性とプールを、ストレージクラスによって要求されたものと比較します。ストレージプールの属性と名前が要求されたすべての属性とプール名と一致する場合、Tridentはそのストレージプールを、そのストレージクラスに適したストレージプールのセットに追加します。さらに、Tridentは `additionalStoragePools`リストに記載されているすべてのストレージプールを、属性がストレージクラスの要求された属性のすべてまたは一部を満たしていない場合でも、そのセットに追加します。 `excludeStoragePools`リストを使用して、ストレージクラスで使用するストレージプールを上書きして削除する必要があります。Trident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を実行し、そのストレージプールが既存のストレージクラスの要件を満たしているかどうかを確認し、除外としてマークされているものを削除します。

ボリュームの作成

Tridentは、ストレージクラスとストレージプール間の関連付けを使用して、ボリュームをプロビジョニングする場所を決定します。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のストレージプールのセットを取得し、ボリュームのプロトコルを指定すると、Tridentは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を削除します（たとえば、NetApp HCI/SolidFireバックエンドはファイルベースのボリュームを提供できませんが、ONTAP NASバックエンドはブロックベースのボリュームを提供できません）。Tridentは、ボリュームの均等な分散を容易にするために、この結果セットの順序をランダム化し、それを反復して、各ストレージプールにボリュームを順番にプロビジョニングしようとします。1つでも成功すると、正常に戻り、プロセス中に発生したすべての失敗がログに記録されます。Tridentは、要求されたストレージクラスとプロトコルで利用可能な*すべての*ストレージプールのプロビジョニングに失敗した場合*にのみ*、失敗を返します。

ボリュームスナップショット

Tridentがドライバーのボリュームスナップショットの作成を処理する方法の詳細については、こちらをご覧ください。

ボリューム **Snapshot** の作成について学ぶ

- ontap-nas、 ontap-san、および `azure-netapp-files`ドライバの場合、各永続ボリューム（PV）はFlexVolボリュームにマッピングされます。その結果、ボリュームスナップショットはNetAppスナップショットとして作成されます。NetAppスナップショットテクノロジは、競合するスナップショットテクノロジよりも優れた安定性、スケーラビリティ、リカバリ性、パフォーマンスを実現します。これらのスナップショットコピーは、作成に必要な時間とストレージスペースの両方において非常に効率的です。
- `ontap-nas-flexgroup`ドライバの場合、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、ボリュームスナップショットはNetApp FlexGroupスナップショットとして作成されます。NetAppスナップショットテクノロジは、競合するスナップショットテクノロジよりも優れた安定性、スケーラビリティ、リカバリ性、パフォーマンスを提供します。これらのスナップショットコピーは、作成に必要な時間とストレージスペースの両方において非常に効率的です。
- `ontap-san-economy`ドライバの場合、PVは共有FlexVolボリューム上に作成されたLUNにマップされます。PVのVolumeSnapshotsは、関連付けられたLUNのFlexClonesを実行することで実現されます。ONTAP FlexCloneテクノロジにより、最大規模のデータセットであってもほぼ瞬時にコピーを作成できます。コピーは親とデータブロックを共有し、メタデータに必要なストレージ以外は消費しません。
- `solidfire-san`ドライバの場合、各PVはNetApp Element ソフトウェア/NetApp HCI クラスタで作成されたLUNにマッピングされます。VolumeSnapshotsは、基盤となるLUNのElementスナップショットで表されます。これらのスナップショットはポイントインタイムコピーであり、システムリソースとスペースをわずかしか消費しません。
- `ontap-nas`および `ontap-san`ドライバーを使用する場合、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。これにより、スナップショットが作成/

スケジュールされるにつれて、ボリューム内の書き込み可能な領域の量が時間の経過とともに減少する可能性があります。これを解決する簡単な方法の1つは、Kubernetesを使用してサイズを変更し、ボリュームを増やすことです。もう1つのオプションは、不要になったスナップショットを削除することです。Kubernetesを通じて作成されたVolumeSnapshotが削除されると、Tridentは関連するONTAPスナップショットを削除します。Kubernetes経由で作成されなかったONTAPスナップショットも削除できます。

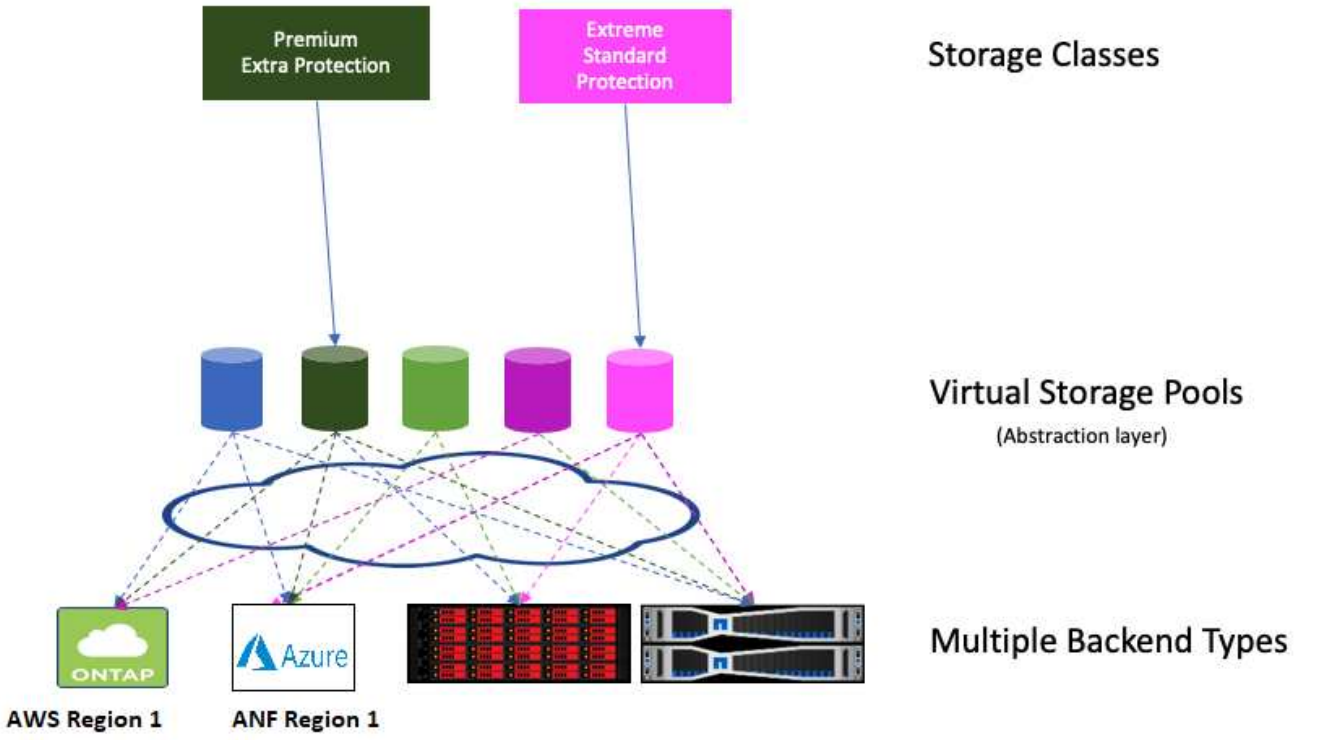
Tridentでは、VolumeSnapshotsを使用して、それらから新しいPVを作成できます。これらのスナップショットからPVを作成するには、サポートされているONTAPバックエンドに対してFlexCloneテクノロジを使用します。スナップショットからPVを作成する場合、バッキングボリュームはスナップショットの親ボリュームのFlexCloneになります。 `solidfire-san`ドライバは、Element ソフトウェアボリュームクローンを使用してスナップショットからPVを作成します。ここでは、Element スナップショットからクローンを作成します。

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 `StorageClasses`の間に抽象化レイヤーを提供します。管理者は、 `StorageClass`が希望する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、バックエンドごとに場所、パフォーマンス、保護などの側面を、バックエンドに依存しない共通の方法で定義できます。

仮想プールについて

ストレージ管理者は、JSON または YAML 定義ファイル内の任意の Trident バックエンドで仮想プールを定義できます。



仮想プールリストの外部で指定されたすべてのアスペクトはバックエンドに対してグローバルであり、すべての仮想プールに適用されますが、各仮想プールは1つ以上のアスペクトを個別に指定できます（バックエンド

グローバルアスペクトを上書きします）。

- 仮想プールを定義するときは、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することはお勧めしません。変更を加えるには、新しい仮想プールを定義する必要があります。

ほとんどの側面は、バックエンド固有の用語で指定されます。重要なのは、アスペクト値はバックエンドのドライバーの外部には公開されておらず、 `StorageClasses`でのマッチングには利用できないことです。代わりに、管理者は仮想プールごとに1つ以上のラベルを定義します。各ラベルはキー：値のペアであり、ラベルは固有のバックエンド間で共通である可能性があります。アスペクトと同様に、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。事前定義された名前と値を持つアスペクトとは異なり、管理者は必要に応じてラベルキーと値を完全に定義できます。便宜上、ストレージ管理者は仮想プールごとにラベルを定義し、ラベルごとにボリュームをグループ化できます。

仮想プールのラベルは、次の文字を使用して定義できます。

- 大文字 `A-Z`
- 小文字 `a-z`
- 数字 `0-9`
- アンダースコア `_`
- ハイフン `-`

A `StorageClass`は、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識別します。仮想プールセレクターは次の演算子をサポートします：

オペレーター	例	プールのラベル値は次の条件を満たす必要があります：
`=`	パフォーマンス＝プレミアム	一致
`!=`	パフォーマンス!=極限	一致しない
`in`	（east、west）の場所	値のセットに含まれる
`notin`	performance notin（silver、bronze）	値のセットに含まれない
`<key>`	保護	任意の値で存在する
`!<key>`	！保護	存在しない

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、 "ボリュームアクセスグループ"を参照してください。

CHAP を使用している場合は、このセクションを無視してください。CHAP を使用すると、管理が簡素化され、以下で説明するスケーリング制限を回避できるため、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できます。Trident は、拡張 CSI プロビジョナーとしてインストールされると CHAP を使用します。

Trident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が無効になっている場合は、 `trident`というアクセスグループが見つかると予想されます（構成で1つ以上のアクセス グループIDを指定しない限り）。

Tridentは新しいボリュームを構成された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は作成または管理しません。アクセスグループは、ストレージバックエンドがTridentに追加される前に存在している必要があり、そのバックエンドによってプロビジョニングされたボリュームをマウントする可能性のあるKubernetesクラスター内のすべてのノードからのiSCSI IQNを含める必要があります。ほとんどのインストールでは、クラスター内のすべてのワーカーノードが含まれます。

64 を超えるノードを持つ Kubernetes クラスターの場合は、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 グループを構成すると、最大 256 ノードのクラスター内の任意のノードが任意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、 "ここをクリックしてください。"を参照してください。

デフォルトの `trident`アクセスグループを使用している設定を、他のアクセスグループも使用する設定に変更する場合は、 `trident`アクセスグループのIDをリストに含めてください。

Trident のクイックスタート

Tridentをインストールして、数ステップでストレージリソースの管理を開始できます。開始する前に、"Tridentの要件"を確認してください。

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

1 ワーカーノードを準備する

Kubernetes クラスタ内のすべてのワーカーノードは、ポッド用にプロビジョニングしたボリュームをマウントできる必要があります。

"ワーカーノードを準備する"

2 **Trident** をインストール

Trident は、さまざまな環境や組織に最適化された複数のインストール方法とモードを提供します。

"Tridentをインストール"

3 バックエンドを作成する

バックエンドは、Trident とストレージ システムの関係を定義します。バックエンドは、Trident がそのストレージ システムとどのように通信するか、および Trident がそこからボリュームをどのようにプロビジョニングするかを指定します。

"バックエンドを設定する"ストレージ システム用

4 Kubernetes StorageClassを作成する

Kubernetes StorageClassオブジェクトはTridentをプロビジョナーとして指定し、カスタマイズ可能な属性を持つ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プロビジョナーを指定するKubernetesオブジェクト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

PersistentVolume (PV) は、Kubernetes クラスター上でクラスター管理者によってプロビジョニングされる物理ストレージ リソースです。*PersistentVolumeClaim* (PVC) は、クラスター上の PersistentVolume へのアクセス要求です。

PersistentVolume（PV）とPersistentVolumeClaim（PVC）を作成します。これらは、設定された Kubernetes StorageClassを使用して PV へのアクセスを要求します。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

追加のバックエンドを追加したり、ストレージ クラスを管理したり、バックエンドを管理したり、ボリューム操作を実行したりできるようになりました。

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する必要があります。特定のバックエンドには追加の要件がある場合があります。

Tridentに関する重要な情報

Trident に関する以下の重要な情報を必ずお読みください。

Tridentに関する重要な情報

- Kubernetes 1.34 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Trident は SAN 環境でのマルチパス構成の使用を厳格に強制し、multipath.conf ファイル内の推奨値は `find_multipaths: no`です。

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

サポートされているフロントエンド（オーケストレータ）

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ターをサポートします：

- Anthos On-Prem（VMware）およびベアメタル版 Anthos 1.16
- Kubernetes 1.27 - 1.34
- OpenShift 4.12、4.14 - 4.20（iSCSIノードの準備をOpenShift 4.19で使用する場合、サポートされる最小Tridentバージョンは25.06.1です。）

> Tridentは、"Red Hat Extended Update Support（EUS）リリースライフサイクル"に準拠して古いOpenShiftバージョンを引き続きサポートしています。これは、アップストリームで公式にサポートされなくなったKubernetesバージョンに依存している場合でも同様です。このような場合にTridentをインストールする際は、Kubernetesバージョンに関する警告メッセージを無視しても問題ありません。

- Rancher Kubernetes Engine 2（RKE2）v1.28.x - 1.34.x

> *Trident* は *Rancher Kubernetes Engine 2* （*RKE2*） バージョン *1.27.x - 1.34.x* でサポートされていますが、*Trident* は現在 *RKE2 v1.28.5+rke2r1* でのみ認定されています。

Trident は、 Google Kubernetes Engine （ GKE ）、 Amazon Elastic Kubernetes Services （ EKS ）、 Azure Kubernetes Service （ AKS ）、 Mirantis Kubernetes Engine （ MKE ）、 VMWare Tanzu Portfolio など、他の多くの完全マネージド型およびセルフマネージド型の Kubernetes サービスとも連携します。

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

> Tridentがインストールされている Kubernetes クラスターを1.25から1.26以降にアップグレードする前に、"Helmインストールのアップグレード"を参照してください。

サポートされているバックエンド（ストレージ）

Tridentを使用するには、以下のサポートされているバックエンドの1つ以上が必要です：

- Amazon FSx for NetApp ONTAP
- Azure NetApp Files
- Cloud Volumes ONTAP
- Google Cloud NetApp Volumes
- NetApp All SAN Array（ASA）
- オンプレミスFAS、AFF、またはASA r2（iSCSI、NVMe/TCP、FC）で、NetAppの完全サポートまたは限定サポート対象のONTAPバージョンを実行しているもの。"ソフトウェア バージョンのサポート"を参照してください。
- NetApp HCI/Element ソフトウェア 11 以上

KubeVirtおよびOpenShift VirtualizationのTridentサポート

サポートされているストレージドライバー：

Tridentは、KubeVirtおよびOpenShift VirtualizationについてONTAPドライバをサポートしています：

- ontap-nas
- ontap-nas-economy
- ontap-san（iSCSI、FCP、NVMe over TCP）
- ontap-san-economy（iSCSI のみ）

考慮すべき点：

- OpenShift Virtualization環境で、ストレージクラスを更新し、 `fsType`パラメータ（例： `fsType: "ext4"`）を追加してください。必要に応じて、 `volumeMode=Block`パラメータを `dataVolumeTemplates`で明示的に設定し、CDIにBlockデータボリュームの作成を通知してください。
- ブロックストレージドライバの*RWX*アクセスモード: ontap-san (iSCSI、NVMe/TCP、FC) および ontap-san-economy (iSCSI) ドライバは、"volumeMode: Block" (raw デバイス) でのみサポートされます。これらのドライバでは、ボリュームが raw デバイスモードで提供されるため、 `fstype`パラメータは使用できません。
- RWX アクセスモードが必要なライブ移行ワークフローでは、次の組み合わせがサポートされています：
  - NFS＋ `volumeMode=Filesystem`
  - iSCSI + `volumeMode=Block` （raw デバイス）
  - NVMe/TCP + `volumeMode=Block` （raw デバイス）
  - FC + `volumeMode=Block` （raw device）

機能の要件

以下の表は、このリリースの Trident で利用可能な機能と、サポートされる Kubernetes のバージョンをまとめたものです。

機能	**Kubernetes**バージョン	機能ゲートが必要ですか？
Trident	1.27 - 1.34	いいえ
ボリューム Snapshot	1.27 - 1.34	いいえ
ボリュームスナップショットからのPVC	1.27 - 1.34	いいえ
iSCSI PV のサイズ変更	1.27 - 1.34	いいえ
ONTAP 双方向 CHAP	1.27 - 1.34	いいえ
動的エクスポートポリシー	1.27 - 1.34	いいえ
Trident Operator	1.27 - 1.34	いいえ
CSIトポロジー	1.27 - 1.34	いいえ

テスト済みのホストオペレーティングシステム

ただし、Tridentは特定のオペレーティングシステムを公式にサポートしていませんが、以下のオペレーティングシステムは動作することが確認されています：

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

  NVMe/TCP には RHEL 9 以降が必要です。

- AMD64およびARM64上のUbuntu 22.04 LTS以降
- Windows Server 2022
- SUSE Linux Enterprise Server（SLES）15以降

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、これらのワーカーは、使用しているバックエンドに応じて、標準のNFSクライアントまたはiSCSIイニシエーターを使用してTridentが提供するボリュームをマウントできる必要があります。

`tridentctl`ユーティリティは、これらの Linux ディストリビューションのいずれでも実行できます。

ホスト構成

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

ストレージシステムの構成

Tridentでは、バックエンド構成でストレージシステムを使用する前に、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が通信するには、特定のポートへのアクセスが必要です。

"Tridentポート"

コンテナイメージと対応する Kubernetes バージョン

エアギャップインストールの場合、次のリストはTridentのインストールに必要なコンテナイメージのリファレンスです。 `tridentctl images`コマンドを使用して、必要なコンテナイメージのリストを確認します。

Trident 25.10に必要なコンテナイメージ

Kubernetesのバージョン	コンテナイメージ
v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0、v1.32.0、v1.33.0、v1.34.0	• docker.io/netapp/trident:25.10.0 • docker.io/netapp/trident-autosupport:25.10 • registry.k8s.io/sig-storage/csi-provisioner:v5.3.0 • registry.k8s.io/sig-storage/csi-attacher:v4.10.0 • registry.k8s.io/sig-storage/csi-resizer:v1.14.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.15.0 • docker.io/netapp/trident-operator:25.10.0（オプション）

Tridentをインストール

Trident オペレータを使用したインストール

tridentctlを使用してインストールする

OpenShift 認定オペレーターを使用したインストール

Tridentを使用

ワーカーノードを準備する

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

適切なツールの選択

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

NFSツール

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

iSCSIツール

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

NVMeツール

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

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

SCSI over FCツール

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

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

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

SMBツール

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

ノードサービス検出

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

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

イベントを確認する

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

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

検出されたサービスを確認する

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

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

NFSボリューム

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

RHEL 8以降

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

Ubuntu

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

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

iSCSIボリューム

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

iSCSI自己修復機能

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

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

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

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

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

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

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

- iSCSIセッションが見つかりません
- 不足しているLUN

Tridentをアップグレードする前に考慮すべき点

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

iSCSIツールをインストールする

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

開始する前に

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

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

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

を指定してください。 "Red Hat ドキュメント"を参照してください。

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

RHEL 8以降

1. 次のシステムパッケージをインストールします：

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

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

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

3. スキャンを手動に設定します：

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

4. マルチパスを有効にする：

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

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

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

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

6. 有効化して起動 `iscsi`：

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

Ubuntu

1. 次のシステムパッケージをインストールします：

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

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

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

3. スキャンを手動に設定します：

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

4. マルチパスを有効にする：

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

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

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

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

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

iSCSI self-healingを設定または無効にする

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

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

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

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

Helm

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

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

```
helm install trident trident-operator-100.2506.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

tridentctl

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

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

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

NVMe/TCPボリューム

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

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

RHEL 9

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

Ubuntu

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

インストールの検証

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

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

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

SCSI over FCボリューム

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

前提条件

FC に必要なネットワークとノードの設定を構成します。

ネットワーク設定

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

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

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

   詳細については、次のONTAPドキュメントを参照してください：

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

FCツールをインストールする

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

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

RHEL 8以降

1. 次のシステムパッケージをインストールします：

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

2. マルチパスを有効にする：

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

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

3. `multipathd`が実行されていることを確認します：

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

Ubuntu

1. 次のシステムパッケージをインストールします：

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

2. マルチパスを有効にする：

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

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

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

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

SMB ボリュームのプロビジョニングの準備

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

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

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

開始する前に

SMB ボリュームをプロビジョニングする前に、次のものが必要です。

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

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

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

手順

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

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

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

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

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

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

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

c. 共有が作成されたことを確認します：

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

詳細については、"SMB共有を作成する"を参照してください。

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

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

バックエンドの設定と管理

バックエンドを設定

バックエンドは、Trident とストレージ システムの関係を定義します。バックエンドは、Trident がそのストレージ システムとどのように通信するか、および Trident がそこからボリュームをどのようにプロビジョニングするかを指定します。

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

- "Azure NetApp Files バックエンドを構成する"

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

Azure NetApp Files

Azure NetApp Files バックエンドを構成する

Azure NetApp Files を Trident のバックエンドとして構成できます。Azure NetApp Files バックエンドを使用して NFS および SMB ボリュームを接続できます。Trident は、Azure Kubernetes Services（AKS）クラスターのマネージド ID を使用した資格情報管理もサポートしています。

Azure NetApp Files ドライバの詳細

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

考慮事項

- Azure NetApp Files サービスは 50 GiB 未満のボリュームをサポートしていません。Trident は、より小さいボリュームが要求された場合、50 GiB のボリュームを自動的に作成します。
- Trident は、 Windows ノード上で実行されているポッドにマウントされた SMB ボリュームのみをサポートします。

AKS のマネージド ID

Trident は Azure Kubernetes Services クラスタの"マネージド ID"をサポートしています。マネージド ID によって提供される合理化されたクレデンシャル管理を活用するには、次のものが必要です：

- AKS を使用してデプロイされた Kubernetes クラスタ
- AKS Kubernetes クラスタで設定された管理対象 ID
- Tridentがインストールされており、 `cloudProvider`を指定するための `"Azure"`が含まれています。

Trident オペレータ

Trident オペレータを使用して Trident をインストールするには、 `tridentorchestrator_cr.yaml`を編集して `cloudProvider`を `"Azure"`に設定します。例：

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

Helm

次の例は、環境変数 `$CP`を使用して、Trident セット `cloudProvider`を Azure にインストールします：

```
helm install trident trident-operator-100.2506.0.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、 `cloudProvider` フラグを `Azure`に設定します：

```
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 クレデンシャルを提供する代わりに、ワークロード ID として認証することで Azure リソースにアクセスできるようになります。

Azure でクラウド ID を活用するには、次のものが必要です：

- AKS を使用してデプロイされた Kubernetes クラスタ
- AKS Kubernetes クラスタで設定されたワークロード ID と oidc-issuer
- Trident がインストールされており、 `cloudProvider`を指定し、 `"Azure"`および `cloudIdentity`でワークロード ID を指定します

Trident オペレータ

Trident オペレータを使用して Trident をインストールするには、 `tridentorchestrator_cr.yaml`を編集して `cloudProvider`を `"Azure"`に設定し、 `cloudIdentity`を `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`に設定します。

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' # Edit
```

Helm

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

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

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

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

<code>tridentctl</code>

次の環境変数を使用して、*クラウド プロバイダ*および*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-
xxxxxxxxxxx"
```

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

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

Azure NetApp Files バックエンドを設定するための準備

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

NFSおよび**SMB**ボリュームの前提条件

Azure NetApp Files を初めて使用する場合、または新しい場所で使用する場合は、Azure NetApp Files をセットアップして NFS ボリュームを作成するために、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 のセットアップと NFS ボリュームの作成"。

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

- `subscriptionID`、 `tenantID`、 `clientID`、 `location`、および `clientSecret`は、AKS クラスターでマネージド ID を使用する場合はオプションです。
- `tenantID`、 `clientID`、および `clientSecret`は、AKSクラスターでクラウドIDを使用する場合はオプションです。

- 容量プール。"Microsoft：Azure NetApp Files の容量プールを作成する"を参照してください。
- Azure NetApp Files に委任されたサブネット。"Microsoft：サブネットを Azure NetApp Files に委任する"を参照してください。
- `subscriptionID`Azure NetApp Files が有効になっている Azure サブスクリプションから。
- `tenantID`、 `clientID`、および `clientSecret`は、十分な権限を持つAzure Active Directoryの"アプリ登録"から、Azure NetApp Filesサービスに対して使用されます。アプリ登録は次のいずれかを使用する必要があります：
  - 所有者または貢献者の役割"Azure によって事前定義済み"。
  - "カスタム Contributor ロール"サブスクリプションレベルで(`assignableScopes`）Tridentが必要とするものだけに制限された以下の権限を持ちます。カスタムロールを作成したら、"Azure ポータルを使用してロールを割り当てる"。

カスタム貢献者ロール

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

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

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

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

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

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

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

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

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

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

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
```

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

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

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

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

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

SMB ボリュームの追加要件

SMB ボリュームを作成するには、次のものが必要です：

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

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

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

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

バックエンド構成オプション

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

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

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

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

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

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

必要な権限とリソース

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

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

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

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

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

ボリュームのプロビジョニング

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

パラメータ	概要	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。`exportRule`は、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たリストにする必要があります。SMBボリュームの場合は無視されます。	"0.0.0.0/0"
`snapshotDir`	.snapshot ディレクトリの可視性を制御します	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（4 桁の 8 進数）。SMB ボリュームの場合は無視されます。	「」（プレビュー機能、サブスクリプションでホワイトリストへの登録が必要）

構成例

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

最小限の構成

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

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

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

AKS のマネージド ID

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

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

AKS のクラウド ID

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

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

容量プールフィルタを使用した特定のサービスレベル設定

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

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

手動 **QoS** 容量プールを使用したバックエンドの例

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

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

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

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

仮想プールの構成

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

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

サポートされているトポロジ構成

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

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

ストレージクラスの定義

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

`parameter.selector`フィールドを使用した定義例

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

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

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

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

SMB ボリュームの定義例

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

デフォルトの名前空間での基本設定

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

名前空間ごとに異なるシークレットを使用する

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

ボリュームごとに異なるシークレットを使用する

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

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

バックエンドを作成する

バックエンド構成ファイルを作成したら、次のコマンドを実行します：

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

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs
```

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

Google Cloud NetApp Volumes

Google Cloud NetApp Volumes バックエンドを設定する

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 Volumes バックエンドを使用して NFS および SMB 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、 `google-cloud-netapp-volumes`クラスタと通信するための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	**volumeMode**	サポートされているアクセスモード	サポートされているファイルシステム
`google-cloud-netapp-volumes`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	`nfs`, `smb`

GKE のクラウド ID

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

Google Cloud でクラウド ID を活用するには、次のものが必要です：

- GKE を使用してデプロイされた Kubernetes クラスタ。
- GKE クラスタで設定されたワークロード ID と、ノードプールで設定された GKE MetaData Server。
- Google Cloud NetApp Volumes管理者（roles/netapp.admin）ロールまたはカスタムロールを持つGCPサ

ービスアカウント。

- cloudProviderに「GCP」を指定し、cloudIdentityに新しいGCPサービスアカウントを指定するTridentがインストールされています。以下に例を示します。

Trident オペレータ

Trident オペレーターを使用して Trident をインストールするには、 `tridentorchestrator_cr.yaml`を編集して `cloudProvider`を `"GCP"`に設定し、 `cloudIdentity`を `iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com`に設定します。

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)** および **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、Trident をインストールし、環境変数 `$CP`を使用して `cloudProvider`を GCP に設定し、環境変数 `$ANNOTATION`を使用して `cloudIdentity`を設定します：

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

<code>tridentctl</code>

次の環境変数を使用して、*クラウド プロバイダ*および*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、 `cloud-provider`フラグを `$CP`に、 `cloud-identity`を `$ANNOTATION`に設定します：

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

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

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

NFSボリュームの前提条件

Google Cloud NetApp Volumes を初めて使用する場合、または新しい場所で使用する場合は、Google Cloud NetApp Volumes をセットアップして NFS ボリュームを作成するために初期設定が必要です。"開始する前に"を参照してください。

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

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

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

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

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

バックエンド構成オプション

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

パラメータ	概要	デフォルト
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	`storageDriverName`の値は「google-cloud-netapp-volumes」として指定する必要があります。

パラメータ	概要	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	ドライバー名 + "_" + API キーの一部
storagePools	ボリューム作成用のストレージプールを指定するために使用されるオプションのパラメータ。	
projectNumber	Google Cloud アカウントのプロジェクト番号。値は Google Cloud ポータルのホームページにあります。	
location	Google Cloud の所在地。Trident が GCNV ボリュームを作成します。リージョンをまたがる Kubernetes クラスタを作成する場合、 `location`で作成されたボリュームは、複数の Google Cloud リージョンにまたがるノードでスケジュールされたワークロードで使用できます。リージョン間のトラフィックには追加コストが発生します。	
apiKey	netapp.admin`ロールを持つGoogle Cloudサービスアカウントの`API`キー。これには、Google Cloudサービスアカウントの秘密鍵ファイルの`JSON`形式の内容（バックエンド構成ファイルにそのままコピーされます）が含まれます。 `apiKey`には、次のキーのキーと値のペアを含める必要があります： `type、 project_id、 client_email、 client_id、 auth_uri、 token_uri、 auth_provider_x509_cert_url、および client_x509_cert_url。	
nfsMountOptions	NFS マウントオプションのきめ細かな制御。	"nfsvers=3"
limitVolumeSize	要求されたボリュームサイズがこの値を超える場合、プロビジョニングは失敗します。	""（デフォルトでは強制されません）
serviceLevel	ストレージ プールとそのボリュームのサービス レベル。値は flex、 standard、 premium、または `extreme`です。	
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	""
network	GCNV ボリュームに使用される Google Cloud ネットワーク。	
debugTraceFlags	トラブルシューティング時に使用するデバッグフラグ。例： {"api":false, "method":true}。トラブルシューティングを行っており、詳細なログダンプが必要な場合を除き、これを使用しないでください。	null
nasType	NFS または SMB ボリュームの作成を構成します。オプションは nfs、 `smb`または null です。null に設定すると、デフォルトで NFS ボリュームになります。	nfs

パラメータ	概要	デフォルト
supportedTopologies	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、"CSI トポロジを使用する"を参照してください。例： supportedTopologies: - topology.kubernetes.io/region: asia-east1 topology.kubernetes.io/zone: asia-east1-a	

ボリュームのプロビジョニング オプション

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

パラメータ	概要	デフォルト
exportRule	新しいボリュームのエクスポートルール。任意の IPv4 アドレスの組み合わせをコンマで区切ったリストにする必要があります。	"0.0.0.0/0"
snapshotDir	`.snapshot`ディレクトリへのアクセス	NFSv4の場合は"true"、NFSv3の場合は"false"
snapshotReserve	Snapshot 用に予約されているボリュームの割合	""（デフォルトの0を受け入れます）
unixPermissions	新しいボリュームの UNIX 権限（4 桁の 8 進数）。	""

構成例

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

最小限の構成

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

この構成は、Google Cloud NetApp Volumes を使い始めたばかりで試している場合に最適ですが、実際には、プロビジョニングするボリュームに対して追加のスコープを指定する必要がある可能性が高くなります。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec
  private_key: |
    -----BEGIN PRIVATE KEY-----
    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
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

SMB ボリュームの設定

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

StoragePoolsフィルターを使用した構成

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

仮想プールの構成

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

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

詳細については、"仮想プール"を参照してください。

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

サポートされているトポロジ構成

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

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

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します：

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

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

```
kubectl get tridentbackendconfig

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

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

```
tridentctl logs
```

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

ストレージクラスの定義

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

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

- `parameter.selector`フィールドを使用した定義例：*

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

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

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

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

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

SMB ボリュームの定義例

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

デフォルトの名前空間での基本設定

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

名前空間ごとに異なるシークレットを使用する

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

ボリュームごとに異なるシークレットを使用する

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

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

PVC定義の例

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

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

```
kubectl get pvc gcnv-nfs-pvc

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

NetApp HCI または SolidFire バックエンドを設定する

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

要素ドライバの詳細

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

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

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

開始する前に

Element バックエンドを作成する前に、次のものが必要です。

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

バックエンド構成オプション

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	デフォルト
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	常に「solidfire-san」
`backendName`	カスタム名またはストレージバックエンド	「solidfire_」+ ストレージ（iSCSI）IP アドレス
`Endpoint`	テナント資格情報を持つSolidFireクラスターのMVIP	
`SVIP`	ストレージ（iSCSI）IP アドレスとポート	
`labels`	ボリュームに適用する任意のJSON 形式のラベルのセット。	""
`TenantName`	使用するテナント名（見つからない場合は作成されます）	
`InitiatorIFace`	iSCSIトラフィックを特定のホストインターフェイスに制限する	"default"
`UseCHAP`	CHAP を使用して iSCSI を認証します。Trident は CHAP を使用します。	true
`AccessGroups`	使用するアクセスグループIDのリスト	「trident」という名前のアクセスグループのIDを検索します
`Types`	QoS仕様	

パラメータ	概要	デフォルト
`limitVolumeSize`	要求されたボリュームサイズがこの値を超える場合、プロビジョニングに失敗します	""（デフォルトでは強制されません）
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例：{"api":false, "method":true}	null

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

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

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

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

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

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

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

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

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

```
  - labels:
      performance: silver
      cost: "3"
    zone: us-east-1b
    type: Silver
  - labels:
      performance: bronze
      cost: "2"
    zone: us-east-1c
    type: Bronze
  - labels:
      performance: silver
      cost: "1"
    zone: us-east-1d
```

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

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

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

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

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

```
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze; cost=2
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=1
  fsType: ext4

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

詳細情報の参照

- "ボリュームアクセスグループ"

ONTAP SANドライバー

ONTAP SAN ドライバの概要

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

ONTAP SAN ドライバーの詳細

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

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

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

ユーザー権限

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

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

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

NVMe/TCPに関する追加の考慮事項

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

- IPv6
- NVMe ボリュームの Snapshot とクローン
- NVMe ボリュームのサイズ変更
- Tridentの外部で作成されたNVMeボリュームをインポートして、そのライフサイクルをTridentで管理できるようにする
- NVMe ネイティブマルチパス
- K8sノードの正常または異常シャットダウン（24.06）

Tridentでサポートされていない機能：

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

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

ONTAP SANドライバを使用してバックエンドを設定する準備をします

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

要件

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

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

複数のドライバを同時に実行し、それぞれに対応するストレージクラスを作成できることも覚えておいてください。たとえば、 `san-dev`ドライバを使用する `ontap-san`クラスと、 `san-default`ドライバを使用する `ontap-san-economy`クラスを設定することができます。

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

ONTAP バックエンドを認証します

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

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

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

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

クレデンシャルベースの認証を有効にする

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

サンプルのバックエンド定義は次のようになります：

YAML

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

JSON

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

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

証明書ベースの認証の有効化

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

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

一般的なワークフローには次の手順が含まれます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で取得した値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションする

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

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

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

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

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

Trident 用のカスタム ONTAP ロールを作成します

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

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

ONTAPコマンドラインの使用

1. 次のコマンドを使用して新しいロールを作成します：

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

2. Tridentユーザーのユーザー名を作成します：

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

3. ロールをユーザーにマップします：

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

System Managerを使用

ONTAP System Managerで次の手順を実行します。

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

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

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

詳細については、次のページを参照してください：

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

双方向**CHAP**による接続の認証

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

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

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

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

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

仕組み

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

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

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

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

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

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

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

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

ONTAP SAN 構成オプションと例

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

す。このセクションでは、バックエンドの設定例と、バックエンドを StorageClasses にマッピングするための詳細について説明します。

"ASA r2システム"は、ストレージ レイヤの実装において他のONTAPシステム（ASA、AFF、FAS）とは異なります。これらの違いは、記載されている特定のパラメータの使用に影響します。"ASA r2システムとその他のONTAPシステムの違いについて詳しくは、こちらをご覧ください"。

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

Trident バックエンド構成では、システムが ASA r2 であることを指定する必要はありません。 `ontap-san`を`storageDriverName`として選択すると、Trident は ASA r2 またはその他の ONTAP システムを自動的に検出します。以下の表に記載されているように、一部のバックエンド構成パラメータは ASA r2 システムには適用されません。

バックエンド構成オプション

バックエンド構成オプションについては、次の表を参照してください：

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

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

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

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

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

formatOptionsの使用に関する推奨事項

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

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

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

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

手順

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

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

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

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

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

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

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

ボリュームのプロビジョニング用のバックエンド設定オプション

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

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

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

ボリュームプロビジョニングの例

デフォルトを定義した例を次に示します：

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

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

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

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

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

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

最小限の構成例

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

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

ONTAP SANの例

これは、 `ontap-san`ドライバを使用した基本的な設定です。

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

MetroCluster の例

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

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

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

ONTAP SANエコノミーの例

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

証明書ベースの認証の例

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

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

双方向CHAPの例

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

ONTAP SAN CHAPの例

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

ONTAP SAN economy CHAPの例

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

NVMe/TCPの例

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

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

SCSI over FC（FCP）の例

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

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

nameTemplateを使用したバックエンド構成の例

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

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

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

仮想プールを使用したバックエンドの例

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

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

グループ化できます。

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

ONTAP SANの例

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

ONTAP SANエコノミーの例

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

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

NVMe/TCPの例

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

バックエンドを**StorageClasses**にマッピングする

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ONTAP NAS ドライバー

ONTAP NAS ドライバの概要

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

ONTAP NAS ドライバーの詳細

Tridentは、ONTAPクラスタと通信するための次の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	volumeMode	サポートされているアクセスモード	サポートされているファイルシステム
`ontap-nas`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	""、`nfs`、`smb`
`ontap-nas-economy`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	""、`nfs`、`smb`
`ontap-nas-flexgroup`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	""、`nfs`、`smb`

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

ユーザー権限

Tridentは、通常、 `admin`クラスタユーザまたは `vsadmin`SVMユーザ、あるいは同じロールを持つ別の名前のユーザを使用して、ONTAPまたはSVM管理者として実行されることが想定されています。

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

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

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

ONTAP NAS ドライバーを使用した ONTAP バックエンドの設定に関する要件、認証オプション、エクスポートポリシーを理解します。

25.10リリース以降、NetApp Tridentは"NetApp AFX ストレージ システム"をサポートします。NetApp AFXストレージシステムは、ストレージ レイヤの実装において、他のONTAPシステム（ASA、AFF、FAS）とは異なります。

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

Trident バックエンド構成では、システムが AFX であることを指定する必要はありません。 `ontap-nas`を`storageDriverName`として選択すると、Trident は AFX システムを自動的に検出します。

要件

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

ONTAP バックエンドを認証します

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

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

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

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

クレデンシャルベースの認証を有効にする

Trident が ONTAP バックエンドと通信するには、 SVM スコープ / クラスタスコープの管理者のクレデンシャルが必要です。 `admin`や `vsadmin`などの標準の事前定義されたロールを使用することを推奨します。これ

により、将来の ONTAP リリースで公開される可能性のある機能 API を将来の Trident リリースで使用できるように、上位互換性が確保されます。カスタムセキュリティログインロールを作成して Trident で使用することもできますが、推奨されません。

サンプルのバックエンド定義は次のようになります：

YAML

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

JSON

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

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

証明書ベースの認証を有効にする

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

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

無視できます。

一般的なワークフローには次の手順が含まれます。

手順

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

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

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

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

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

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

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

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

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

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

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

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

7. 前の手順で取得した値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションする

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

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

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

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

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

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

Trident 用のカスタム **ONTAP** ロールを作成します

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

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

い。

ONTAPコマンドラインの使用

1. 次のコマンドを使用して新しいロールを作成します：

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

2. Tridentユーザーのユーザー名を作成します：

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

3. ロールをユーザーにマップします：

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

System Managerを使用

ONTAP System Managerで次の手順を実行します。

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

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

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

詳細については、次のページを参照してください：

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

NFS エクスポート ポリシーを管理する

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

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

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

エクスポート ポリシーを動的に管理する

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

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

例

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

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

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

上記の例を使用して、この機能がどのように機能するかを説明します：

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

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

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

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

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

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

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

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

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

SMB ボリュームのプロビジョニングの準備

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

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

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

開始する前に

SMB ボリュームをプロビジョニングする前に、次のものが必要です。

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

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

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

手順

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

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

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

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

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

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

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

   c. 共有が作成されたことを確認します：

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

詳細については、"SMB共有を作成する"を参照してください。

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

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

セキュア**SMB**を有効にする

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

覚えておくべきポイント

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

手順

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

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

2. ストレージ クラスに注釈を追加します。

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

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

1. PVCを作成します。

次の例では、PVC を作成します。

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

ONTAP NAS 構成オプションと例

Tridentインストールで使用するONTAP NASドライバーの作成方法と使用方法について説明します。このセクションでは、バックエンドの設定例と、バックエンドをStorageClassesにマッピングするための詳細について説明します。

25.10リリース以降、NetApp Tridentは"NetApp AFX ストレージ システム"をサポートします。NetApp AFXストレージシステムは、ストレージ レイヤの実装において、他のONTAPベースのシステム（ASA、AFF、FAS）とは異なります。

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

Trident バックエンド設定では、システムがNetApp AFX ストレージ システムであることを指定する必要はありません。 `ontap-nas`を `storageDriverName`として選択すると、Trident は AFX ストレージ システムを自動的に検出します。以下の表に示すように、一部のバックエンド構成パラメータは AFX ストレージ システムには適用されません。

バックエンド構成オプション

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	デフォルト
`version`		常に1

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

パラメータ	概要	デフォルト
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコードされた値。オプション。証明書ベースの認証に使用	""
`username`	クラスタ/SVM に接続するためのユーザー名。資格情報ベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	
`password`	クラスター/SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます。Active Directory 認証については、"Active Directory の認証情報を使用してバックエンド SVM に Trident を認証"を参照してください。	
`storagePrefix`	SVM で新しいボリュームをプロビジョニングするときに使用されるプレフィックス。設定後は更新できません ontap-nas-economy と 24 文字以上の storagePrefix を使用する場合、ボリューム名にはストレージ プレフィックスが含まれますが、qtree にはストレージ プレフィックスが埋め込まれません。	「trident」
`aggregate`	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVM に割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバーの場合、このオプションは無視されます。割り当てられていない場合は、利用可能なアグリゲートのいずれかを使用してFlexGroupボリュームをプロビジョニングできます。 SVM でアグリゲートが更新されると、Trident Controller を再起動することなく、SVM をポーリングすることで Trident で自動的に更新されます。ボリュームをプロビジョニングするために Trident で特定のアグリゲートを設定している場合、そのアグリゲートの名前が変更されたり SVM から移動されたりすると、SVM アグリゲートのポーリング中にバックエンドが Trident で障害状態に移行します。バックエンドをオンラインに戻すには、アグリゲートを SVM 上に存在するものに変更するか、完全に削除する必要があります。 **AFX** ストレージ システムには指定しないでください。	""

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

パラメータ	概要	デフォルト
`qtreesPerFlexvol`	FlexVol あたりの最大 Qtree 数は、範囲 [50, 300] 内である必要があります	"200"
`smbShare`	次のいずれかを指定できます：Microsoft Management ConsoleまたはONTAP CLIを使用して作成されたSMB共有の名前、TridentがSMB共有を作成できるようにする名前、またはパラメータを空白のままにしてボリュームへの共通共有アクセスを防止できます。このパラメータは、オンプレミスONTAPではオプションです。このパラメータは、Amazon FSx for ONTAPバックエンドでは必須であり、空白にすることはできません。	`smb-share`
`useREST`	ONTAP REST APIを使用するブーリアン パラメータ。 `useREST`に設定すると `true`、TridentはONTAP REST APIを使用してバックエンドと通信します。 `false`に設定すると、TridentはONTAPI（ZAPI）呼び出しを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、`ontapi`アプリケーションへのアクセス権が必要です。これは、事前定義された `vsadmin`および`cluster-admin`ロールで満たされます。Trident 24.06リリースおよびONTAP 9.15.1以降では、 `useREST`はデフォルトで `true`に設定されます。ONTAPI（ZAPI）呼び出しを使用するには、 `useREST`を`false`に変更してください。指定する場合は、**AFX**ストレージシステムでは常に **`true`**に設定してください。	ONTAP 9.15.1以降の場合は`true`、それ以外の場合は `false`。
`limitVolumePoolSize`	ontap-nas-economy バックエンドで Qtree を使用する場合の最大リクエスト可能 FlexVol サイズ。	""（デフォルトでは強制されません）
`denyNewVolumePools`	バックエンドが Qtree を格納する新しい FlexVol ボリュームを作成できないように制限 `ontap-nas-economy`します。新しい PV のプロビジョニングには、既存の FlexVol のみが使用されます。	
`adAdminUser`	SMB 共有へのフルアクセス権を持つ Active Directory 管理者ユーザーまたはユーザー グループ。このパラメータを使用して、SMB 共有へのフルコントロールの管理者権限を付与します。	

ボリュームのプロビジョニング用のバックエンド設定オプション

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

パラメータ	概要	デフォルト
`spaceAllocation`	Qtreeのスペース割り当て	"true"

パラメータ	概要	デフォルト
spaceReserve	スペース予約モード：「none」（シン）または「volume」（シック）	「なし」
snapshotPolicy	使用するSnapshotポリシー	「なし」
qosPolicy	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージプール / バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください	""
adaptiveQosPolicy	作成されたボリュームに割り当てるアダプティブ QoS ポリシー グループ。ストレージ プール/バックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。ontap-nas-economy ではサポートされていません。	""
snapshotReserve	Snapshot 用に予約されているボリュームの割合	`snapshotPolicy`が「none」の場合は「0」、それ以外の場合は「」
splitOnClone	作成時にクローンを親から分離する	"false"
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたボリュームはすべてNAEが有効になります。詳細については、次を参照してください："Tridentと NVE および NAE の連携"。	"false"
tieringPolicy	階層化ポリシーで「none」を使用	
unixPermissions	新しいボリュームのモード	NFS ボリュームの場合は「777」、SMB ボリュームの場合は空（該当なし）
snapshotDir	`.snapshot`ディレクトリへのアクセスを制御します	NFSv4の場合は"true"、NFSv3の場合は"false"
exportPolicy	使用するエクスポートポリシー	"default"
securityStyle	新しいボリュームのセキュリティ スタイル。NFSは `mixed`および `unix`セキュリティ スタイルをサポートします。SMBは `mixed`および `ntfs`セキュリティ スタイルをサポートします。	NFSのデフォルトは `unix`です。SMBのデフォルトは `ntfs`です。
nameTemplate	カスタムボリューム名を作成するためのテンプレート。	""

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

ボリュームプロビジョニングの例

デフォルトを定義した例を次に示します：

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

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

計算は次のとおりです：

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

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

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

最小限の構成例

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

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

ONTAP NAS エコノミーの例

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

ONTAP NAS FlexGroupの例

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

MetroCluster の例

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

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

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

SMB ボリュームの例

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

証明書ベースの認証の例

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

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

自動エクスポートポリシーの例

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

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

IPv6アドレスの例

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

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

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

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

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

nameTemplateを使用したバックエンド構成の例

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

仮想プールを使用したバックエンドの例

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

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

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

ONTAP NAS の例

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

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

ONTAP NAS FlexGroupの例

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

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

ONTAP NAS エコノミーの例

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

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

バックエンドを**StorageClasses**にマッピングする

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

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

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

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

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

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

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

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

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

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

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

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

初期設定後にアップデート `dataLIF`

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

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

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

セキュアな **SMB** の例

ontap-nas ドライバを使用したバックエンド構成

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

ontap-nas-economy ドライバを使用したバックエンド構成

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

ストレージ プールを使用したバックエンド構成

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

ontap-nas ドライバを使用したストレージクラスの例

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

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

ontap-nas-economy ドライバを使用したストレージクラスの例

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

単一の **AD** ユーザを使用した **PVC** の例

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

複数の **AD** ユーザを使用した **PVC** の例

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

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP で **Trident** を使用

"Amazon FSx for NetApp ONTAP"は、NetApp ONTAPストレージオペレーティングシステムを搭載したファイルシステムを起動して実行できる、完全に管理されたAWSサービスです。FSx for ONTAPを使用すると、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用しながら、AWSにデータを保存するシンプルさ、俊敏性、セキュリティ、スケーラビリティのメリットを享受できます。FSx for ONTAPは、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。

Amazon FSx for NetApp ONTAP ファイルシステムを Trident と統合することで、Amazon Elastic Kubernetes Service（EKS）で実行されている Kubernetes クラスターが、ONTAP でサポートされ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、Amazon FSx の主要なリソースであり、オンプレミスの ONTAP クラスタに相当します。各 SVM 内に、ファイル システム内のファイルとフォルダーを保存するデータ コンテナーであるボリュームを 1 つまたは複数作成できます。Amazon FSx for NetApp ONTAP では、クラウド内のマネージドファイ

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タイプは **NetApp ONTAP** と呼ばれます。

TridentとAmazon FSx for NetApp ONTAPを使用することで、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"に加えて、FSx for ONTAPをTridentと統合するには、次のものが必要です：

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

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

考慮事項

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

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

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

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

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

認証

Tridentには2つの認証モードがあります。

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

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

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

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

- "ONTAP NAS 認証"
- "ONTAP SAN 認証"

テスト済みの **Amazon** マシンイメージ（**AMI**）

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

AMI	NAS	NASエコノミー	iSCSI	iSCSIエコノミー
AL2023_x86_64_STANDARD	はい	はい	はい	はい
AL2_x86_64	はい	はい	○*	○*
BOTTLEROCKET_x86_64	はい**	はい	N/A	N/A
AL2023_ARM_64_STANDARD	はい	はい	はい	はい
AL2_ARM_64	はい	はい	○*	○*

BOTTLEROCKET_ARM_64	はい**	はい	N/A	N/A

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

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

テストに使用したデバイス：

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

実行されたテスト：

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

詳細情報の参照

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

IAMロールと**AWS**シークレットを作成する

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

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

AWS Secrets Manager シークレットを作成する

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

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

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

IAM ポリシーを作成する

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

次の例では、AWS CLI を使用して IAM ポリシーを作成します：

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

ポリシー **JSON** の例：

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

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

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

Pod アイデンティティ

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

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

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

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

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

trust-relationship.json を作成：

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

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

trust-relationship.json ファイル：

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

IAM ロールにロールポリシーをアタッチします：

前の手順のロールポリシーを、作成した IAM ロールにアタッチします：

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

ポッド **ID** の関連付けを作成します：

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

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

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

AWS CLI の使用：

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

trust-relationship.json ファイル：

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

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

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

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

IAM ロールを **IAM** ポリシーにアタッチします：

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

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

OICD プロバイダーが関連付けられていることを確認します：

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

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

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

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

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

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

Tridentをインストール

Trident は、Kubernetes における Amazon FSx for NetApp ONTAP のストレージ管理を合理化し、開発者と管理者がアプリケーションの展開に集中できるようにします。

Tridentは次のいずれかの方法を使用してインストールできます：

- Helm
- EKS アドオン

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

helm 経由で **Trident** をインストール

Pod アイデンティティ

1. Trident Helm リポジトリを追加します：

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

2. 次の例を使用して Trident をインストールします：

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

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                           APP VERSION

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

サービス アカウント アソシエーション（**IRSA**）

1. Trident Helm リポジトリを追加します：

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

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

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

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

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

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

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

EKS アドオン経由で **Trident** をインストール

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

前提条件

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

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

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

AWS の **Trident** アドオンを有効にする

管理コンソール

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

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

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

```
{

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

}
```

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

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

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

AWS CLI

1. `add-on.json`ファイルを作成する：

Pod Identity の場合は、次の形式を使用します：

ビジネス

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

IRSA 認証の場合は、次の形式を使用します：

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

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

2.Trident EKS アドオンをインストールします。

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

eksctl

次のコマンド例では、Trident EKS アドオンをインストールします：

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

Trident EKS アドオンを更新する

管理コンソール

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

AWS CLI

次の例では、EKS add-on を更新します。

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

eksctl

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

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

  出力例：

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

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

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

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

Trident EKS アドオンをアンインストール/削除します

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

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

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

管理コンソール

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

AWS CLI

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

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

eksctl

次のコマンドは、Trident EKS アドオンをアンインストールします：

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

ストレージバックエンドを設定する

ONTAP SAN および **NAS** ドライバの統合

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

YAML

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

JSON

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

次のコマンドを実行して、Trident バックエンド構成（TBC）を作成および検証します：

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

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- Trident バックエンド構成（TBC）が正常に作成されたことを確認します：

```
Kubectl get tbc -n trident
```

```
NAME                           BACKEND NAME          BACKEND UUID
PHASE    STATUS

backend-tbc-ontap-nas          tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9    Bound    Success
```

FSx for ONTAPドライバーの詳細

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

- `ontap-san`：プロビジョニングされた各PVは、Amazon FSx for NetApp ONTAP ボリューム内のLUNです。ブロックストレージに推奨されます。
- `ontap-nas`：プロビジョニングされた各PVは、完全なAmazon FSx for NetApp ONTAP ボリュームです。NFS および SMB に推奨されます。
- `ontap-san-economy`：プロビジョニングされた各PVは、Amazon FSx for NetApp ONTAP ボリュームあたりのLUN数を設定可能なLUNです。
- `ontap-nas-economy`：プロビジョニングされた各PVはqtreeで、Amazon FSx for NetApp ONTAP volumeあたりのqtree数は設定可能です。
- `ontap-nas-flexgroup`：プロビジョニングされた各PVは、完全なAmazon FSx for NetApp ONTAP FlexGroupボリュームです。

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

設定ファイルが作成されたら、次のコマンドを実行して EKS 内に作成します：

```
kubectl create -f configuration_file
```

ステータスを確認するには、次のコマンドを実行します：

```
kubectl get tbc -n trident
```

```
NAME                      BACKEND NAME            BACKEND UUID
PHASE   STATUS
backend-fsx-ontap-nas   backend-fsx-ontap-nas   7a551921-997c-4c37-a1d1-
f2f4c87fa629   Bound   Success
```

バックエンドの高度な構成と例

バックエンド構成オプションについては、次の表を参照してください：

パラメータ	概要	例
`version`		常に1
`storageDriverName`	ストレージドライバーの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + dataLIF
`managementLIF`	クラスタまたはSVM管理LIFのIPアドレス（完全修飾ドメイン名（FQDN）も指定可能）Tridentをを使用してインストールした場合、IPv6アドレスを使用するように設定できます。IPv6アドレスは、［28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555］のように角括弧で定義する必要があります。もし `fsxFilesystemID`を `aws`フィールドの下で指定した場合、`managementLIF`を指定する必要はありません。なぜならTridentがAWSからSVM `managementLIF`情報を取得するためです。したがって、SVM配下のユーザー（例：vsadmin）の認証情報を必ず指定し、そのユーザーが `vsadmin`ロールを持っている必要があります。	"10.0.0.1"、"[2001:1234:abcd::fefe]"

パラメータ	概要	例
`dataLIF`	プロトコル LIF の IP アドレス。**ONTAP NAS** ドライバー：NetApp では dataLIF を指定することを推奨しています。指定しない場合、 Trident は SVM から dataLIF を取得します。NFS マウント操作に使用する完全修飾ドメイン名（ FQDN ）を指定できます。これにより、複数の dataLIF 間で負荷を分散するラウンドロビン DNS を作成できます。初期設定後に変更できます。を参照してください。**ONTAP SAN** ドライバー： iSCSI には指定しないでください。Trident は ONTAP Selective LUN Map を使用して、マルチパスセッションの確立に必要な iSCI LIF を検出します。dataLIF が明示的に定義されている場合は警告が生成されます。Trident が IPv6 フラグを使用してインストールされている場合は、 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] のように角括弧で定義する必要があります。	
`autoExportPolicy`	自動エクスポート ポリシーの作成と更新を有効にします [ブール値]。`autoExportPolicy`および`autoExportCIDRs`オプションを使用すると、Tridentはエクスポートポリシーを自動的に管理できます。	`false`
`autoExportCIDRs`	`autoExportPolicy`が有効になっている場合にKubernetesのノードIPをフィルタリングするためのCIDRのリスト。 `autoExportPolicy`および `autoExportCIDRs`オプションを使用すると、Tridentはエクスポート ポリシーを自動的に管理できます。	"["0.0.0.0/0", "::/0"]"
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	""
`clientCertificate`	クライアント証明書の Base64 エンコードされた値。証明書ベースの認証に使用	""
`clientPrivateKey`	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用	""

パラメータ	概要	例
`trustedCACertificate`	信頼された CA 証明書の Base64 エンコードされた値。オプション。証明書ベースの認証に使用されます。	""
`username`	クラスタまたは SVM に接続するためのユーザー名。資格情報ベースの認証に使用されます。たとえば、vsadmin。	
`password`	クラスターまたは SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
`svm`	使用するStorage Virtual Machine	SVM 管理 LIF が指定されている場合に派生されます。
`storagePrefix`	SVM で新しいボリュームをプロビジョニングするときに使用されるプレフィックス。作成後は変更できません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`
`limitAggregateUsage`	*Amazon FSx for NetApp ONTAP には指定しないでください。*提供された `fsxadmin`と `vsadmin`には、Trident を使用してアグリゲートの使用状況を取得して制限するために必要な権限が含まれていません。	使用しないでください。
`limitVolumeSize`	要求されたボリューム サイズがこの値を超える場合、プロビジョニングは失敗します。また、qtreeとLUNを管理するボリュームの最大サイズを制限し、`qtreesPerFlexvol`オプションにより、FlexVol volumeあたりのqtreeの最大数をカスタマイズできます	""（デフォルトでは強制されません）
`lunsPerFlexvol`	FlexVol volumeあたりの最大LUN数は、[50、200]の範囲にする必要があります。SANのみ。	“100”
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例：{"api":false, "method":true} `debugTraceFlags`を使用しないでください。ただし、トラブルシューティングを行っており、詳細なログ ダンプが必要な場合を除きます。	null

パラメータ	概要	例
nfsMountOptions	NFS マウント オプションのコンマ区切りリスト。Kubernetes 永続ボリュームのマウント オプションは通常ストレージ クラスで指定されますが、ストレージ クラスでマウント オプションが指定されていない場合、Trident はストレージ バックエンドの構成ファイルで指定されたマウント オプションを使用するようになります。ストレージ クラスまたは構成ファイルにマウント オプションが指定されていない場合、Trident は関連付けられている永続ボリュームにマウント オプションを設定しません。	""
nasType	NFS または SMB ボリュームの作成を設定します。オプションは nfs、 smb、または null です。*SMB ボリュームの場合は `smb`に設定する必要があります。*null に設定すると、デフォルトで NFS ボリュームになります。	nfs
qtreesPerFlexvol	FlexVol volume あたりの最大 qtree 数は、[50, 300] の範囲内である必要があります	"200"
smbShare	次のいずれかを指定できます：Microsoft 管理コンソールまたは ONTAP CLI を使用して作成された SMB 共有の名前、または Trident が SMB 共有を作成できるようにするための名前。このパラメータは、Amazon FSx for NetApp ONTAP バックエンドに必要です。	smb-share
useREST	ONTAP REST APIを使用するためのブーリアン パラメータ。 `true`に設定すると、TridentはONTAP REST APIを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、 `ontap`アプリケーションへのアクセス権が必要です。これは、事前定義された `vsadmin`および `cluster-admin`ロールで満たされます。	false

パラメータ	概要	例
aws	AWS FSx for ONTAP の設定ファイルでは以下を指定できます：fsxFilesystemID：AWS FSx ファイルシステムの ID を指定します。 apiRegion：AWS API リージョン名。 apikey：AWS API キー。 secretKey：AWS 秘密キー。	"" "" ""
credentials	AWS Secrets Manager に保存する FSx SVM 認証情報を指定します。- name：SVM の認証情報が含まれるシークレットの Amazon リソースネーム（ARN）。- type：`awsarn`に設定します。詳細については、"AWS Secrets Manager シークレットを作成する"を参照してください。	

ボリュームのプロビジョニング用のバックエンド設定オプション

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

パラメータ	概要	デフォルト
spaceAllocation	LUNのスペース割り当て	true
spaceReserve	スペース予約モード：「none」（シン）または「volume」（シック）	none
snapshotPolicy	使用するSnapshotポリシー	none
qosPolicy	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージ プールまたはバックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。Trident で QoS ポリシー グループを使用するには、ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各構成要素に個別に適用されるようにする必要があります。共有 QoS ポリシー グループは、すべてのワークロードの合計スループットの上限を適用します。	""

パラメータ	概要	デフォルト
adaptiveQosPolicy	作成されたボリュームに割り当てるアダプティブ QoS ポリシー グループ。ストレージ プールまたはバックエンドごとにqosPolicyまたはadaptiveQosPolicyのいずれかを選択してください。ontap-nas-economy ではサポートされていません。	""
snapshotReserve	スナップショット用に予約されているボリュームの割合「0」	もし snapshotPolicy`が `none`の場合、 `else ""
splitOnClone	作成時にクローンを親から分離する	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたボリュームはすべてNAEが有効になります。詳細については、次を参照してください："Tridentと NVE および NAE の連携"。	false
luksEncryption	LUKS 暗号化を有効にします。"Linux Unified Key Setup（LUKS）を使用する"を参照してください。SAN のみ。	""
tieringPolicy	使用する階層化ポリシー none	
unixPermissions	新しいボリュームのモード。**SMB** ボリュームの場合は空白のままにします。	""
securityStyle	新しいボリュームのセキュリティ スタイル。NFSは `mixed`および `unix`セキュリティ スタイルをサポートします。SMBは `mixed`および `ntfs`セキュリティ スタイルをサポートします。	NFSのデフォルトは `unix`です。SMBのデフォルトは `ntfs`です。

SMB ボリュームのプロビジョニング

`ontap-nas`ドライバを使用してSMBボリュームをプロビジョニングできます。<<ONTAP SAN および NAS ドライバの統合>>
を完了する前に、次の手順を実行します（link:https://docs.netapp.com/us-en/trident/trident-use/worker-node-prep.html#prepare-to-provision-smb-volumes["SMB ボリュームのプロビジョニングの準備"]）。

ストレージクラスと**PVC**を設定する

KubernetesのStorageClassオブジェクトを設定し、ストレージクラスを作成してTridentにボリュームのプロビジョニング方法を指示します。設定したKubernetesのStorageClassを使用して、PVへのアクセスを要求するPersistentVolumeClaim（PVC）を作成します。その後、PVをポッドにマウントできます。

ストレージクラスを作成する

Kubernetes StorageClassオブジェクトを設定する

https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass オブジェクト"^]
オブジェクトは、そのクラスで使用されるプロビジョナーとしてTridentを識別し、ボリュームのプロビジョニング方法をTridentに指示します。この例を使用して、NFSを使用するボリュームのStorageclassを設定します（属性の完全なリストについては、以下のTrident属性セクションを参照してください）：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

iSCSI を使用するボリュームの Storageclass を設定するには、次の例を使用します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

AWS BottlerocketでNFSv3ボリュームをプロビジョニングするには、必要な `mountOptions`をストレージクラスに追加します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

ストレージクラスを作成する

手順

1. これはKubernetesオブジェクトなので、 `kubectl`を使用してKubernetesで作成します。

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

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

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

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

PVC を作成する

A "*PersistentVolumeClaim*"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVC は、特定のサイズまたはアクセス モードのストレージを要求するように構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

PVC を作成したら、ボリュームをポッドにマウントできます。

サンプルマニフェスト

PersistentVolumeClaim サンプルマニフェスト

これらの例は、基本的な PVC 構成オプションを示しています。

RWX アクセス付き **PVC**

この例では、RWXアクセスを持つ基本的なPVCが、StorageClassという名前 `basic-csi`に関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-gold
```

iSCSI を使用した **PVC** の例

この例では、RWOアクセスを持つiSCSI用の基本PVCが、StorageClassという名前 `protection-gold`に関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 1Gi
storageClassName: protection-gold
```

PVCを作成する

手順

1. PVC を作成します。

```
kubectl create -f pvc.yaml
```

2. PVC ステータスを確認します。

```
kubectl get pvc
```

```
NAME          STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                           5m
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

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をプロビジョニングするためにどのTrident管理ストレージプールを使用するかを決定します。

属性	タイプ	値	オファー	要求	サポート対象
メディア[1]	string	HDD、ハイブリッド、SSD	プールにはこのタイプのメディアが含まれます。ハイブリッドとは両方を意味します	指定されたメディアタイプ	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san
provisioningType	string	薄い、厚い	プールはこのプロビジョニング方法をサポートしています	プロビジョニング方法が指定されました	thick：すべての ONTAP、thin：すべての ONTAP および solidfire-san
backendType	string	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、azure-netapp-files、ontap-san-economy	プールはこのタイプのバックエンドに属します	バックエンドが指定されました	すべてのドライバー
Snapshot	ブール値	true、false	プールは Snapshot 付きのボリュームをサポートします	スナップショットが有効になっているボリューム	ontap-nas 、ontap-san 、solidfire-san
クローン	ブール値	true、false	プールはボリュームのクローン作成をサポート	クローンが有効なボリューム	ontap-nas 、ontap-san 、solidfire-san

属性	タイプ	値	オファー	要求	サポート対象
暗号化	ブール値	true、false	プールは暗号化されたボリュームをサポートします	暗号化が有効になっているボリューム	ontap-nas、ontap-nas-economy、ontap-nas-flexgroups、ontap-san
IOPS	int	正の整数	プールはこの範囲の IOPS を保証できる	ボリュームで保証されるIOPS	solidfire-san

[1]: ONTAP Select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をデプロイする

ストレージ クラスと PVC が作成されると、PV をポッドにマウントできます。このセクションでは、PV をポッドに接続するためのコマンドと構成の例を示します。

手順

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

以下の例は、PVC をポッドに接続するための基本構成を示しています。基本構成：

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

進捗状況は `kubectl get pod --watch`で確認できます。

2. ボリュームが `/my/mount/path`にマウントされていることを確認します。

```
kubectl exec -it pv-pod -- df -h /my/mount/path
```

```
Filesystem                                                      Size
Used  Avail Use%  Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1%   /my/mount/path
```

これでPodを削除できます。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

EKS クラスター上の **Trident EKS** アドオンを設定する

NetApp Tridentは、Amazon FSx for NetApp ONTAPのKubernetesストレージ管理を合理化し、開発者と管理者がアプリケーションの導入に集中できるようにします。NetApp Trident EKSアドオンには、最新のセキュリティパッチとバグ修正が含まれており、Amazon EKSで動作することがAWSによって検証されています。EKSアドオンを使用すると、Amazon EKSクラスターの安全性と安定性を常に確保し、アドオンのインストール、設定、更新に必要な作業量を削減できます。

前提条件

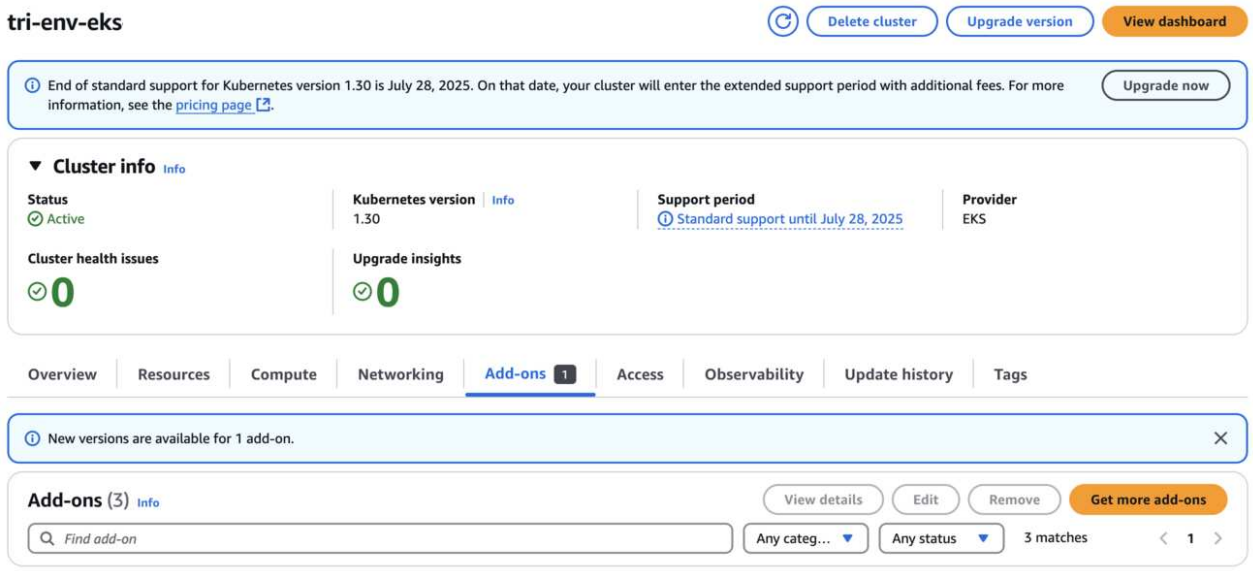
AWS EKS の Trident アドオンを設定する前に、以下のものを用意してください：

- アドオンを操作する権限を持つ Amazon EKS クラスターアカウント。"Amazon EKS アドオン"を参照してください。
- AWS マーケットプレイスへの AWS 権限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI タイプ：Amazon Linux 2（AL2_x86_64）または Amazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存の Amazon FSx for NetApp ONTAP ファイルシステム

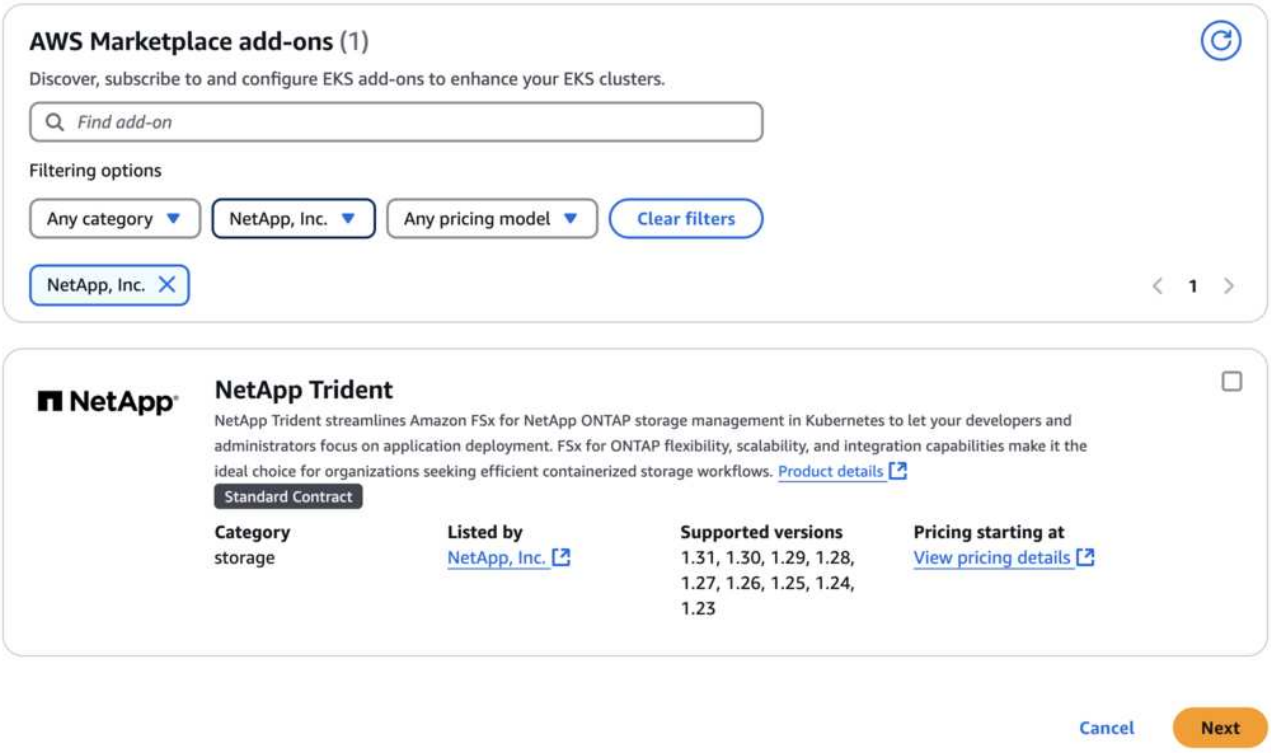
手順

1. EKS ポッドが AWS リソースにアクセスできるようにするには、IAM ロールと AWS シークレットを作成してください。手順については、"IAMロールとAWSシークレットを作成する"を参照してください。

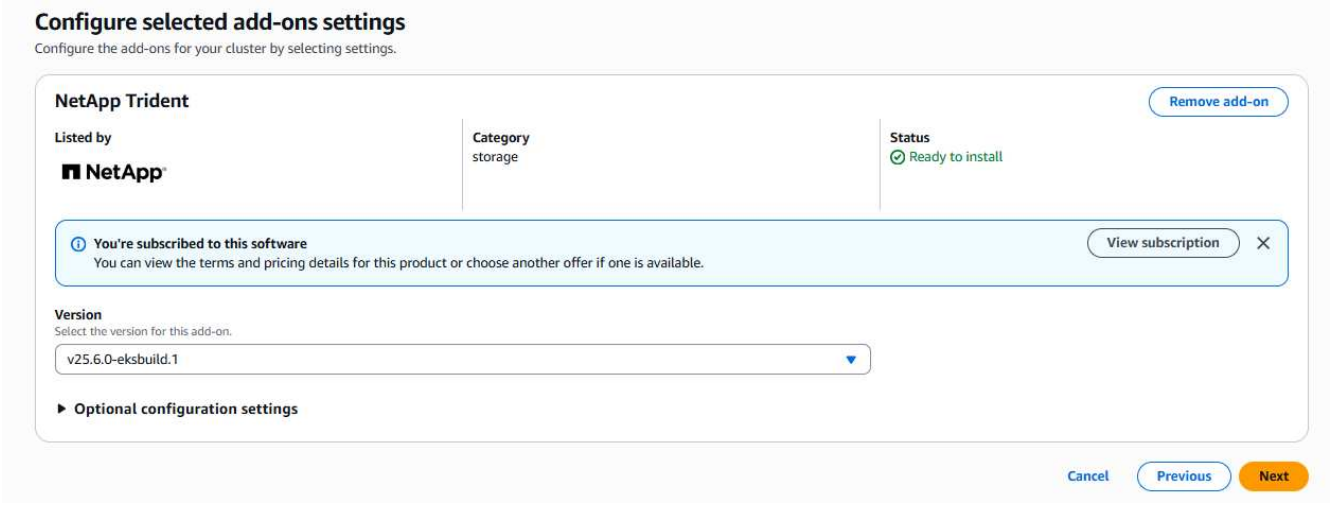
2. EKS Kubernetes クラスターで、 * アドオン * タブに移動します。



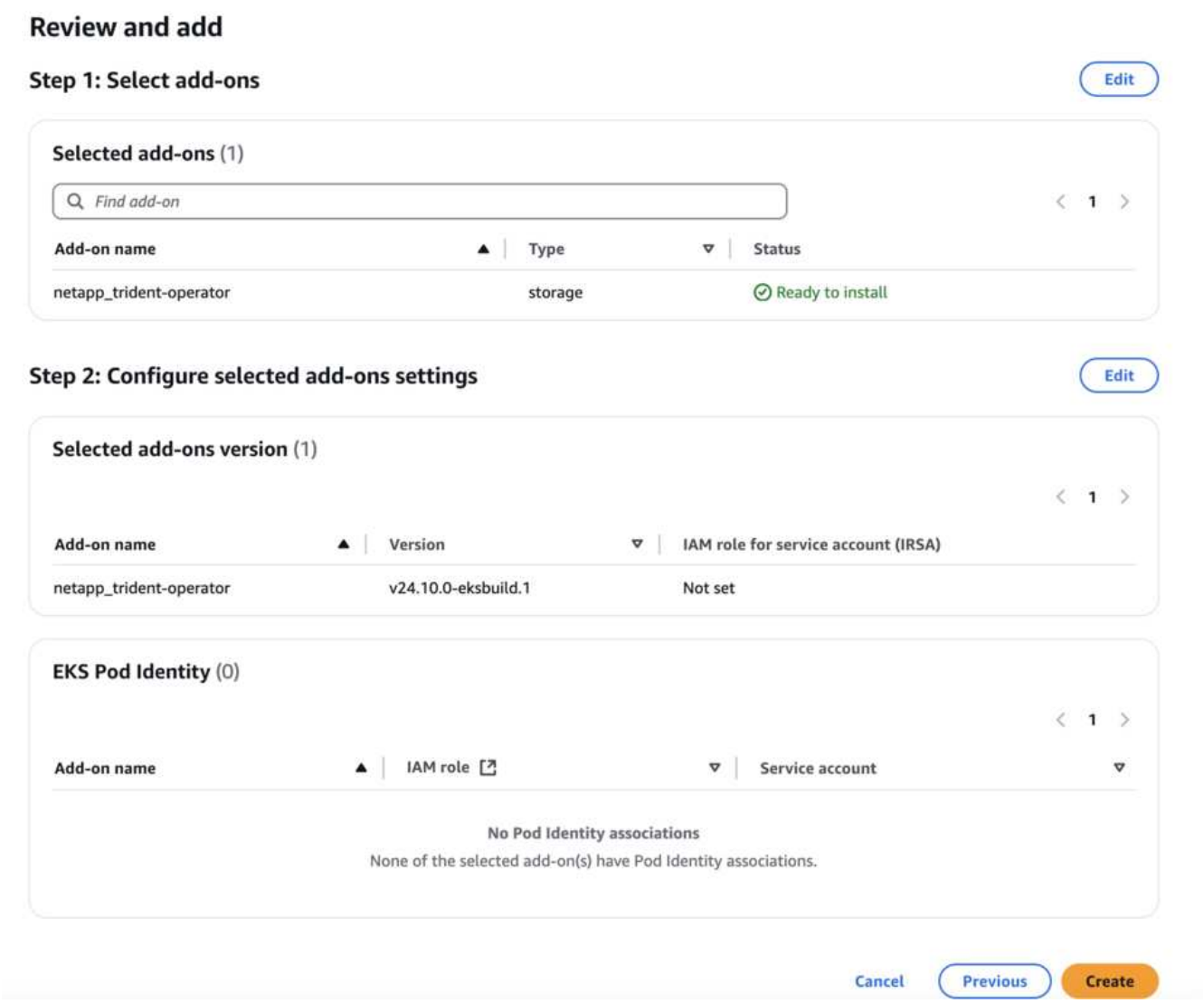
3. **AWS Marketplace** アドオン に移動し、*storage* カテゴリを選択します。



4. **NetApp Trident** を見つけて、Trident アドオンのチェックボックスをオンにし、次へ をクリックします。
5. アドオンの目的のバージョンを選択します。

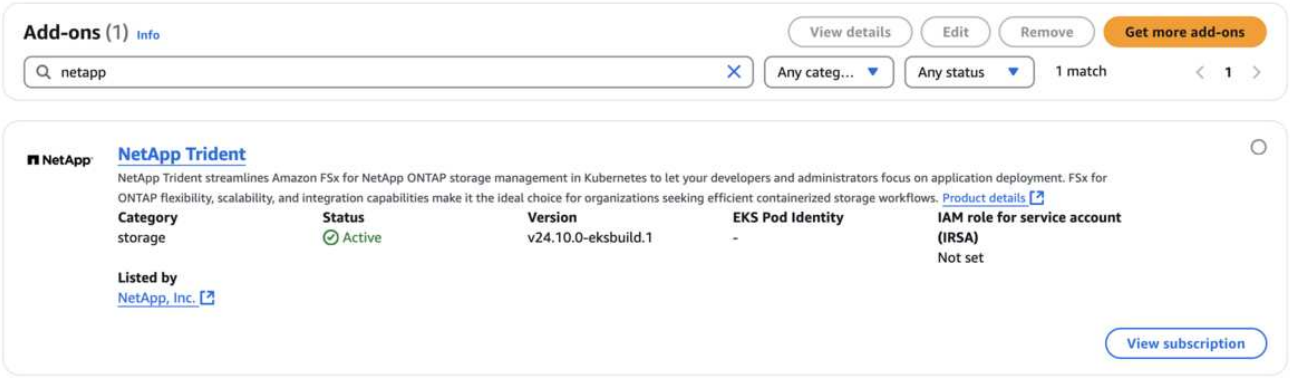


6. 必要なアドオン設定を構成します。



7. IRSA（サービスアカウントのIAMロール）を使用している場合は、追加の構成手順を参照してください"ここをクリックしてください。"。
8. *Create*を選択します。

9. アドオンのステータスが *Active* であることを確認します。



10. 次のコマンドを実行して、Trident がクラスタに正しくインストールされていることを確認します：

```
kubectl get pods -n trident
```

11. セットアップを続行し、ストレージ バックエンドを構成します。詳細については、"ストレージバックエンドを設定する"を参照してください。

CLI を使用した **Trident EKS** アドオンのインストール / アンインストール

CLI を使用して **NetApp Trident EKS** アドオンをインストールします：

次のコマンド例では、Trident EKS アドオンをインストールします：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.0-eksbuild.1`（専用バージョンを使用）

以下のコマンド例は Trident EKS アドオンバージョン 25.6.1 をインストールします：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.1-eksbuild.1`（専用バージョンを使用）

以下のコマンド例は Trident EKS アドオンバージョン 25.6.2 をインストールします：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.2-eksbuild.1`（専用バージョンを使用）

CLI を使用して **NetApp Trident EKS** アドオンをアンインストールします：

次のコマンドは、Trident EKS アドオンをアンインストールします：

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

kubectl でバックエンドを作成する

バックエンドは、Trident とストレージ システム間の関係を定義します。Trident がそのストレージ システムと通信する方法と、Trident がそこからボリュームをプロビジョニングする方法を指定します。Trident のインストール後、次のステップはバックエンドを作成することです。 `TridentBackendConfig` カスタム リソース定義（CRD）を使用す

ると、Kubernetes インターフェースを介して直接 Trident バックエンドを作成および管理できます。これは、 `kubectl` または Kubernetes ディストリビューションに相当する CLI ツールを使用して実行できます。

TridentBackendConfig

`TridentBackendConfig`（`tbc`、 `tbconfig`、 `tbackendconfig`）は、フロントエンドの名前空間CRDであり、 `kubectl`を使用して`Trident`バックエンドを管理できます。`Kubernetes`およびストレージ管理者は、専用のコマンドラインユーティリティ（`tridentctl`）を必要とせずに、Kubernetes CLIを介して直接バックエンドを作成および管理できるようになりました。

`TridentBackendConfig`オブジェクトの作成時に、次のようになります：

- バックエンドは、提供された設定に基づいて Trident によって自動的に作成されます。これは内部的には `TridentBackend`（`tbe`、 `tridentbackend`）CR として表されます。
- `TridentBackendConfig`は、Tridentによって作成された `TridentBackend`に一意にバインドされています。

各 `TridentBackendConfig`は、 `TridentBackend`との1対1のマッピングを維持します。前者は、バックエンドを設計および構成するためにユーザーに提供されるインターフェースであり、後者は Trident が実際のバックエンド オブジェクトを表す方法です。

> `TridentBackend` CRはTridentによって自動的に作成されます。これらを変更*しないでください*。バックエンドを更新する場合は、 `TridentBackendConfig`オブジェクトを変更してください。

`TridentBackendConfig` CRのフォーマットについては、次の例を参照してください：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

また、 "trident-installer"ディレクトリの例で、必要なストレージ プラットフォーム/サービスのサンプル構成を確認することもできます。

`spec`は、バックエンド固有の構成パラメータを受け取ります。この例では、バックエンドは`ontap-san`ストレージ
ドライバを使用し、ここに表されている構成パラメータを使用します。目的のストレージ
ドライバの構成オプションのリストについては、link:backends.html["ストレージ
ドライバのバックエンド設定情報"^]を参照してください。

この `spec`セクションには、 `credentials`や `deletionPolicy`フィールドも含まれており、これらは `TridentBackendConfig`CRで新たに導入されました：

- `credentials`：このパラメータは必須フィールドであり、ストレージ システム/サービスでの認証に使用される資格情報が含まれます。これは、ユーザーが作成した Kubernetes シークレットに設定されます。資格情報はプレーンテキストで渡すことができず、エラーが発生します。
- `deletionPolicy`：このフィールドは、 `TridentBackendConfig`が削除されたときに何が起こるかを定義します。次の2つの値のいずれかを取ることができます：
  - `delete`：これにより、 `TridentBackendConfig` CR と関連するバックエンドの両方が削除されます。これがデフォルト値です。
  - `retain`： `TridentBackendConfig` CR が削除されても、バックエンドの定義はそのまま残り、 `tridentctl`で管理できます。削除ポリシーを `retain`に設定すると、ユーザーは以前のリリース（21.04 より前）にダウングレードし、作成されたバックエンドを保持できます。このフィールドの値は、 `TridentBackendConfig`の作成後に更新できます。

バックエンドの名前は `spec.backendName`を使用して設定されます。指定しない場合、バックエンドの名前は `TridentBackendConfig`オブジェクト（metadata.name）の名前に設定されます。 `spec.backendName`を使用してバックエンド名を明示的に設定することを推奨します。

`tridentctl`で作成されたバックエンドには、関連付けられた `TridentBackendConfig`オブジェクトがありません。そのようなバックエンドは、 `kubectl`で `TridentBackendConfig`CRを作成することで管理することができます。同一の構成パラメータ（例えば、 `spec.backendName`、 `spec.storagePrefix`、 `spec.storageDriverName`など）を指定する必要があるため、注意してください。Tridentは、新しく作成された `TridentBackendConfig`を既存のバックエンドに自動的にバインドします。

手順の概要

`kubectl`を使用して新しいバックエンドを作成するには、次の操作を行う必要があります：

1. "Kubernetes Secret"を作成します。シークレットには、Trident がストレージ クラスタ / サービスと通信するために必要なクレデンシャルが含まれています。
2. `TridentBackendConfig`オブジェクトを作成します。これには、ストレージ クラスタ / サービスに関する詳細が含まれており、前の手順で作成されたシークレットが参照されます。

バックエンドを作成したら、 `kubectl get tbc <tbc-name> -n <trident-namespace>`を使用してそのステータスを確認し、追加の詳細情報を収集できます。

ステップ**1**：**Kubernetes**シークレットを作成する

バックエンドのアクセス資格情報を含むシークレットを作成します。これは各ストレージ サービス/プラットフォームに固有です。次に例を示します：

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: password
```

この表は、各ストレージ プラットフォームの Secret に含める必要があるフィールドをまとめたものです：

ストレージ プラットフォームの **Secret Fields** の説明	シークレット	フィールドの説明
Azure NetApp Files	clientID	アプリ登録からのclient ID
Element（NetApp HCI/SolidFire）	エンドポイント	テナント資格情報を持つSolidFire クラスターのMVIP
ONTAP	ユーザ名	クラスタ/SVM に接続するためのユーザー名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密キーの Base64 エンコードされた値。証明書ベースの認証に使用されます
ONTAP	chapUsername	インバウンドユーザー名。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合

ストレージ プラットフォームの **Secret Fields** の説明	シークレット	フィールドの説明
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合
ONTAP	chapTargetUsername	ターゲットユーザー名。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合
ONTAP	chapTargetInitiatorSecret	CHAP ターゲット イニシエーターシークレット。useCHAP=true の場合は必須です。 `ontap-san`および `ontap-san-economy`の場合

このステップで作成されたSecretは、次のステップで作成される `TridentBackendConfig`オブジェクトの `spec.credentials`フィールドで参照されます。

ステップ**2**： `TridentBackendConfig` **CR**を作成する

これで、 `TridentBackendConfig` CRを作成する準備ができました。この例では、 `ontap-san`ドライバを使用するバックエンドが、以下に示す `TridentBackendConfig`オブジェクトを使用して作成されます：

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

ステップ**3**： `TridentBackendConfig` **CR**のステータスを確認する

これで、 `TridentBackendConfig` CRを作成したので、ステータスを確認できます。次の例を参照してください：

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                       BACKEND NAME           BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

バックエンドが正常に作成され、 `TridentBackendConfig` CRにバインドされました。

フェーズには次のいずれかの値を指定できます：

- `Bound`： `TridentBackendConfig` CR はバックエンドに関連付けられており、そのバックエンドには `configRef`が `TridentBackendConfig` CR の uid に設定されています。
- `Unbound`： `""`を使用して表されます。 `TridentBackendConfig`オブジェクトはバックエンドにバインドされていません。新しく作成されたすべての `TridentBackendConfig` CR は、デフォルトでこのフェーズにあります。フェーズが変更された後は、再び Unbound に戻ることはできません。
- `Deleting`： `TridentBackendConfig` CR の `deletionPolicy`削除するように設定されました。 `TridentBackendConfig` CR が削除されると、削除中状態に移行します。
  - バックエンドに永続ボリュームクレーム（PVC）が存在しない場合は、 `TridentBackendConfig`を削除すると、`Trident`はバックエンドと `TridentBackendConfig` CRも削除します。
  - バックエンドに 1 つ以上の PVC が存在する場合、削除状態になります。 `TridentBackendConfig` CR もその後削除フェーズに入ります。バックエンドと `TridentBackendConfig`は、すべての PVC が削除された後にのみ削除されます。
- `Lost`： `TridentBackendConfig` CR に関連付けられたバックエンドが誤ってまたは故意に削除され、 `TridentBackendConfig` CR には削除されたバックエンドへの参照がまだ残っています。 `TridentBackendConfig` CR は、 `deletionPolicy`の値に関係なく削除できます。
- `Unknown`：Tridentは `TridentBackendConfig` CRに関連付けられているバックエンドの状態または存在を判断できません。たとえば、APIサーバーが応答しない場合や、 `tridentbackends.trident.netapp.io` CRDが存在しない場合などです。この場合、介入が必要になる可能性があります。

この段階で、バックエンドが正常に作成されました。追加で処理できる操作がいくつかあります。"バックエンドの更新とバックエンドの削除"

（オプション）ステップ**4**：詳細を取得する

バックエンドの詳細情報を取得するには、次のコマンドを実行します。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

```
NAME                       BACKEND NAME          BACKEND UUID
PHASE   STATUS     STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success   ontap-san         delete
```

さらに、YAML/JSONダンプを取得することもできます `TridentBackendConfig`。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo` には、 `backendName` と `backendUUID` が含まれており、これは `TridentBackendConfig` CR に応じて作成されたバックエンドのものです。 `lastOperationStatus` フィールドは、 `TridentBackendConfig` CR の直近の操作のステータスを表しており、これはユーザーによってトリガーされる場合（例：ユーザーが `spec` で何かを変更した場合）や、Trident によってトリガーされる場合（例：Trident の再起動時）があります。これは Success または Failed のいずれかになります。 `phase` は、 `TridentBackendConfig` CR とバックエンドとの関係のステータスを表します。上記の例では、 `phase` の値は Bound となっており、これは `TridentBackendConfig` CR がバックエンドと関連付けられていることを意味します。

`kubectl -n trident describe tbc <tbc-cr-name>`コマンドを実行すると、イベントログの詳細を取得できます。

関連付けられた`TridentBackendConfig`オブジェクトを含むバックエンドを`tridentctl`を使用して更新または削除することはできません。`tridentctl`と`TridentBackendConfig`の切り替え手順を理解するには、"こちらをご覧ください"を参照してください。

バックエンドを管理する

kubectl を使用してバックエンド管理を実行する

`kubectl`を使用してバックエンド管理操作を実行する方法について学習します。

バックエンドを削除する

`TridentBackendConfig`を削除すると、`Trident`に対してバックエンドを削除/保持するように指示します（`deletionPolicy`に基づく）。バックエンドを削除するには、`deletionPolicy`が`delete`に設定されていることを確認してください。`TridentBackendConfig`だけを削除するには、`deletionPolicy`が`retain`に設定されていることを確認してください。これにより、バックエンドが引き続き存在し、`tridentctl`を使用して管理できることが保証されます。

次のコマンドを実行します。

```
kubectl delete tbc <tbc-name> -n trident
```

Tridentは`TridentBackendConfig`で使用されていたKubernetes Secretを削除しません。Kubernetesユーザーはシークレットをクリーンアップする責任があります。シークレットを削除するときは注意が必要です。シークレットは、バックエンドで使用されていない場合にのみ削除する必要があります。

既存のバックエンドを表示する

次のコマンドを実行します。

```
kubectl get tbc -n trident
```

`tridentctl get backend -n trident`または`tridentctl get backend -o yaml -n trident`を実行して、存在するすべてのバックエンドのリストを取得することもできます。このリストには、`tridentctl`で作成されたバックエンドも含まれます。

バックエンドを更新する

バックエンドを更新する理由は複数考えられます：

- ストレージシステムへのクレデンシャルが変更されました。クレデンシャルを更新するには、`TridentBackendConfig`オブジェクトで使用されているKubernetes Secretを更新する必要があります。Tridentは、提供された最新のクレデンシャルを使用してバックエンドを自動的に更新します。次のコマンドを実行してKubernetes Secretを更新します：

```
kubectl apply -f <updated-secret-file.yaml> -n trident
```

- パラメータ（使用されている ONTAP SVM の名前など）を更新する必要があります。
  - `TridentBackendConfig`オブジェクトは、次のコマンドを使用してKubernetes経由で直接更新できます：

```
kubectl apply -f <updated-backend-file.yaml>
```

  - あるいは、次のコマンドを使用して既存の `TridentBackendConfig` CR に変更を加えることができます：

```
kubectl edit tbc <tbc-name> -n trident
```

- バックエンドの更新が失敗した場合、バックエンドは最後の既知の構成のままになります。ログを表示して原因を特定するには、 `kubectl get tbc <tbc-name> -o yaml -n trident`または `kubectl describe tbc <tbc-name> -n trident`を実行します。
- 構成ファイルの問題を特定して修正したら、更新コマンドを再実行できます。

tridentctl でバックエンド管理を実行する

`tridentctl`を使用してバックエンド管理操作を実行する方法について説明します。

バックエンドを作成する

"バックエンド設定ファイル"を作成したら、次のコマンドを実行します：

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

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs -n trident
```

設定ファイルの問題を特定して修正したら、 `create`コマンドを再度実行します。

バックエンドを削除する

バックエンドをTridentから削除するには、次の操作を行います：

1. バックエンド名を取得します：

```
tridentctl get backend -n trident
```

2. バックエンドを削除します：

```
tridentctl delete backend <backend-name> -n trident
```

> Tridentがこのバックエンドからプロビジョニングしたボリュームとスナップショットがまだ存在する場合、バックエンドを削除すると、新しいボリュームをプロビジョニングできなくなります。バックエンドは「削除中」の状態で存在し続けます。

既存のバックエンドを表示する

Tridentが認識しているバックエンドを表示するには、次の操作を行います。

- 概要を取得するには、次のコマンドを実行します：

```
tridentctl get backend -n trident
```

- すべての詳細を取得するには、次のコマンドを実行します：

```
tridentctl get backend -o json -n trident
```

バックエンドを更新する

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します：

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

バックエンドの更新が失敗した場合は、バックエンドの構成に問題があるか、無効な更新を試行しました。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs -n trident
```

設定ファイルの問題を特定して修正したら、 `update`コマンドを再度実行します。

バックエンドを使用するストレージクラスを特定する

これは、 `tridentctl`がバックエンドオブジェクトに対して出力するJSONで回答できる質問の種類の例です。これは `jq`ユーティリティを使用しており、インストールする必要があります。

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

これは、 `TridentBackendConfig`を使用して作成されたバックエンドにも適用されます。

バックエンド管理オプション間を移動する

Trident でバックエンドを管理するさまざまな方法について説明します。

バックエンドを管理するオプション

`TridentBackendConfig`の導入により、管理者は、バックエンドを管理する 2 つの独自の方法を利用できるようになりました。これにより、次の疑問が生じます：

- `tridentctl`を使用して作成されたバックエンドは、 `TridentBackendConfig`で管理できますか？
- `TridentBackendConfig`を使用して作成されたバックエンドは、 `tridentctl`を使用して管理できますか？

`tridentctl`を使用してバックエンドを管理 `TridentBackendConfig`

このセクションでは、Kubernetesインターフェイスを通じて `tridentctl`を直接作成し、 `TridentBackendConfig`オブジェクトを作成することで作成されたバックエンドの管理手順について説明します。

これは次のシナリオに適用されます：

- 既存のバックエンドには `TridentBackendConfig`がありません。これは `tridentctl`で作成されたためです。
- `tridentctl`で作成された新しいバックエンド（他の `TridentBackendConfig`オブジェクトが存在する場合）。

どちらのシナリオでも、バックエンドは存在し続け、Tridentがボリュームのスケジュール設定とボリュームに対する操作を行います。管理者には次の2つの選択肢があります（：）

- それを使用して作成されたバックエンドの管理には `tridentctl`を引き続きご利用ください。
- `tridentctl`を使用して作成されたバックエンドを新しい `TridentBackendConfig`オブジェクトにバインドします。これにより、バックエンドは `kubectl`で管理され、 `tridentctl`では管理されなくなります。

`kubectl`を使用して既存のバックエンドを管理するには、既存のバックエンドにバインドする
`TridentBackendConfig`を作成する必要があります。その仕組みの概要は次のとおりです：

1. Kubernetes シークレットを作成します。このシークレットには、Trident がストレージクラスタ/サービスと通信するために必要なクレデンシャルが含まれています。
2. `TridentBackendConfig`オブジェクトを作成します。これには、ストレージクラスタ/サービスに関する詳細が含まれており、前の手順で作成されたシークレットが参照されます。同一の設定パラメータ（ `spec.backendName`、 `spec.storagePrefix`、 `spec.storageDriverName`など）を指定するように注意する必要があります。 `spec.backendName`は既存のバックエンドの名前に設定する必要があります。

ステップ**0**：バックエンドを特定する

既存のバックエンドにバインドする `TridentBackendConfig`を作成するには、バックエンドの構成を取得する必要があります。この例では、次の JSON 定義を使用してバックエンドが作成されたと仮定します：

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

ステップ**1**：**Kubernetes**シークレットを作成する

次の例に示すように、バックエンドの資格情報を含む Secret を作成します：

```
cat tbc-ontap-nas-backend-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password
```

```
kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

ステップ**2**： `TridentBackendConfig` **CR**を作成する

次のステップは、既存の `ontap-nas-backend`に自動的にバインドされる `TridentBackendConfig` CR を作成することです（この例のように）。次の要件が満たされていることを確認してください：

- 同じバックエンド名が `spec.backendName`で定義されています。
- 構成パラメータは元のバックエンドと同一です。
- 仮想プール（存在する場合）は、元のバックエンドと同じ順序を維持する必要があります。
- クレデンシャルはプレーンテキストではなく、Kubernetes Secret を通じて提供されます。

この場合、 `TridentBackendConfig`は次のようになります：

```
cat backend-tbc-ontap-nas.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

ステップ3： `TridentBackendConfig` CRのステータスを確認する

`TridentBackendConfig`が作成された後、そのフェーズは`Bound`である必要があります。また、既存のバックエンドと同じバックエンド名とUUIDを反映する必要があります。

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

バックエンドは、 tbc-ontap-nas-backend `TridentBackendConfig`オブジェクトを使用して完全に管理されるようになります。

`TridentBackendConfig`を使用してバックエンドを管理 `tridentctl`

`tridentctl`は、`TridentBackendConfig`を使用して作成されたバックエンドを一覧表示するために使用できます。さらに、管理者は`tridentctl`を通じてそのようなバックエンドを完全に管理することもでき、`TridentBackendConfig`を削除し、 `spec.deletionPolicy`が`retain`に設定されていることを確認できます。

ステップ**0**：バックエンドを特定する

例えば、次のバックエンドが `TridentBackendConfig`を使用して作成されたとします：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

出力から、`TridentBackendConfig`が正常に作成され、バックエンドにバインドされていることがわかります［バックエンドの UUID を確認してください］。

ステップ**1**： **`deletionPolicy`**が**`retain`**に設定されていることを確認します

`deletionPolicy`の値を見てみましょう。これを`retain`に設定する必要があります。これにより、 `TridentBackendConfig` CRが削除されても、バックエンドの定義はそのまま残り、 `tridentctl`で管理できます。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                    BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

`deletionPolicy`が `retain`に設定されていない限り、次のステップに進まないでください。

ステップ**2**： `TridentBackendConfig` **CR**を削除する

最後のステップは、 `TridentBackendConfig` CRを削除することです。 `deletionPolicy`が `retain`に設定されていることを確認した後、削除を進めることができます：

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

`TridentBackendConfig`オブジェクトを削除すると、Tridentはバックエンド自体を実際に削除せずに、単に削除するだけです。

ストレージクラスの作成と管理

ストレージクラスを作成する

Kubernetes StorageClassオブジェクトを設定し、ストレージクラスを作成してTridentにボリュームのプロビジョニング方法を指示します。

Kubernetes StorageClassオブジェクトを設定する

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass オブジェクト"^]`は、そのクラスで使用されるプロビジョナーとしてTridentを識別し、Tridentにボリュームのプロビジョニング方法を指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確認してください。

ストレージクラスを作成する

StorageClassオブジェクトを作成したら、ストレージクラスを作成できます。[ストレージクラスのサンプル]には、使用または変更できる基本的なサンプルがいくつか用意されています。

手順

1. これはKubernetesオブジェクトなので、 `kubectl`を使用してKubernetesで作成します。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

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

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

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

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

ストレージクラスのサンプル

Tridentは "特定のバックエンド向けのシンプルなストレージクラス定義"を提供します。

あるいは、インストーラーに付属の `sample-input/storage-class-csi.yaml.templ`ファイルを編集し、`*BACKEND_TYPE*`をストレージドライバー名に置き換えることもできます。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

ストレージクラスの管理

既存のストレージクラスを表示したり、デフォルトのストレージクラスを設定したり、ストレージクラスのバックエンドを識別したり、ストレージクラスを削除したりできます。

既存のストレージクラスを表示する

- 既存の Kubernetes ストレージ クラスを表示するには、次のコマンドを実行します：

```
kubectl get storageclass
```

- Kubernetes ストレージ クラスの詳細を表示するには、次のコマンドを実行します：

```
kubectl get storageclass <storage-class> -o json
```

- Tridentの同期されたストレージクラスを表示するには、次のコマンドを実行します：

```
tridentctl get storageclass
```

- Tridentの同期されたストレージクラスの詳細を表示するには、次のコマンドを実行します：

```
tridentctl get storageclass <storage-class> -o json
```

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されました。これは、ユーザーが Persistent Volume Claim（PVC）でストレージクラスを指定しない場合に、Persistent Volume をプロビジョニングするために使用されるストレージクラスです。

- ストレージクラス定義でアノテーション `storageclass.kubernetes.io/is-default-class`を true に設定して、デフォルトのストレージクラスを定義します。仕様によれば、その他の値またはアノテーションの欠如は false として解釈されます。
- 次のコマンドを使用して、既存のストレージ クラスをデフォルトのストレージ クラスとして構成できます：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 同様に、次のコマンドを使用して、デフォルトのストレージ クラス注釈を削除できます：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、この注釈を含む Trident インストーラー バンドルの例もあります。

クラスター内に一度に存在できるデフォルトのストレージ クラスは 1 つだけです。Kubernetes では、技術的には複数のストレージ クラスを持つことを禁止していませんが、デフォルトのストレージ クラスがまったく存在しないかのように動作します。

ストレージクラスのバックエンドを識別する

これは、 `tridentctl`がTridentバックエンド オブジェクトに対して出力するJSONで答えられる質問の種類の例です。これは `jq`ユーティリティを使用しますが、最初にインストールする必要がある場合があります。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ストレージ クラスを削除する

Kubernetes からストレージ クラスを削除するには、次のコマンドを実行します：

```
kubectl delete storageclass <storage-class>
```

`<storage-class>`は、ストレージ クラスに置き換える必要があります。

このストレージクラスで作成された永続ボリュームはそのまま残り、Tridentで引き続き管理されます。

Tridentは、作成するボリュームに対して空白 `fsType`を強制します。iSCSIバックエンドの場合、StorageClassで `parameters.fsType`を強制することが推奨されます。既存のStorageClassesを削除し、 `parameters.fsType`を指定して再作成する必要があります。

ボリュームのプロビジョニングと管理

ボリュームをプロビジョニングする

設定済みのKubernetes StorageClass を使用して、PVへのアクセスを要求するPersistentVolumeClaim（PVC）を作成します。その後、PVをポッドにマウントできます。

概要

A "*PersistentVolumeClaim*"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVC は、特定のサイズまたはアクセス モードのストレージを要求するように構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

PVC を作成したら、ボリュームをポッドにマウントできます。

PVC を作成する

手順

1. PVC を作成します。

```
kubectl create -f pvc.yaml
```

2. PVC ステータスを確認します。

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME      CAPACITY    ACCESS MODES   STORAGECLASS AGE
pvc-storage   Bound    pv-name     1Gi         RWO                       5m
```

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

進捗状況は `kubectl get pod --watch`で確認できます。

2. ボリュームが `/my/mount/path`にマウントされていることを確認します。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. これでPodを削除できます。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

サンプルマニフェスト

PersistentVolumeClaim サンプルマニフェスト

これらの例は、基本的な PVC 構成オプションを示しています。

RWO アクセス付き PVC

この例では、RWOアクセスを持つ基本的なPVCが、 `basic-csi`という名前のStorageClassに関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe/TCP を使用した PVC

この例では、RWOアクセスを持つNVMe/TCPの基本的なPVCが、StorageClassという名前 `protection-gold`に関連付けられています。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Podマニフェストのサンプル

これらの例は、PVCをポッドに接続するための基本的な設定を示しています。

基本設定

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

基本的な**NVMe/TCP**構成

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

"KubernetesとTridentオブジェクト"を参照して、ストレージクラスが `PersistentVolumeClaim`とどのように相互作用するか、および Trident がボリュームをプロビジョニングする方法を制御するパラメータの詳細を確

認してください。

ボリュームを拡張する

Trident は、Kubernetes ユーザーにボリューム作成後にボリュームを拡張する機能を提供します。iSCSI、NFS、SMB、NVMe/TCP、および FC ボリュームを拡張するために必要な構成に関する情報を見つけます。

iSCSI ボリュームを拡張する

CSI プロビジョナーを使用して iSCSI 永続ボリューム（PV）を拡張できます。

> iSCSIボリューム拡張は、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`ドライバでサポートされており、Kubernetes 1.16以降が必要です。

ステップ1：ボリューム拡張をサポートするように**StorageClass**を設定する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドを `true`に設定します。

```
cat storageclass-ontapsan.yaml
```

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

既存のStorageClassの場合は、編集して `allowVolumeExpansion`パラメータを含めます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、 `spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順3：**PVC**を接続するポッドを定義する

サイズを変更するには、PV をポッドに接続します。iSCSI PV のサイズを変更する場合、次の 2 つのシナリオがあります：

- PVがポッドに接続されている場合、Tridentはストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
- アタッチされていないPVのサイズを変更しようとすると、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、Tridentはデバイスを再スキャンし、ファイルシステムのサイズを変更します。拡張操作が正常に完了すると、KubernetesはPVCサイズを更新します。

この例では、 `san-pvc`を使用するポッドが作成されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ**4**：**PV**を拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集して、`spec.resources.requests.storage`を2Giに更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

ステップ**5**：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正しく機能したことを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

FCボリュームを拡張する

CSI プロビジョナーを使用して FC 永続ボリューム（PV）を拡張できます。

FC ボリュームの拡張は、 `ontap-san`ドライバでサポートされており、Kubernetes 1.16 以降が必要です。

ステップ**1**：ボリューム拡張をサポートするように**StorageClass**を設定する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドを `true`に設定します。

```
cat storageclass-ontapsan.yaml
```

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

既存のStorageClassの場合は、編集して `allowVolumeExpansion`パラメータを含めます。

手順2：作成した**StorageClass**を使用して**PVC**を作成する

PVC定義を編集し、 `spec.resources.requests.storage`を更新して新しく希望するサイズを反映します。このサイズは元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続ボリューム（PV）を作成し、それをこの永続ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順3：**PVC**を接続するポッドを定義する

サイズを変更するには、PVをポッドに接続します。FC PVのサイズを変更する場合、次の2つのシナリオがあります：

- PVがポッドに接続されている場合、Tridentはストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
- アタッチされていないPVのサイズを変更しようとすると、Tridentはストレージバックエンドのボリュームを拡張します。PVCがポッドにバインドされた後、Tridentはデバイスを再スキャンし、ファイルシステ

ムのサイズを変更します。拡張操作が正常に完了すると、KubernetesはPVCサイズを更新します。

この例では、 `san-pvc`を使用するポッドが作成されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ**4**：**PV**を拡張する

作成されたPVのサイズを1Giから2Giに変更するには、PVC定義を編集して、 `spec.resources.requests.storage`を2Giに更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

ステップ**5**：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正しく機能したことを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

NFSボリュームを拡張する

Tridentは、 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、および `azure-netapp-files`バックエンドでプロビジョニングされたNFS PVのボリューム拡張をサポートしています。

ステップ**1**：ボリューム拡張をサポートするように**StorageClass**を設定する

NFS PVのサイズを変更するには、管理者はまず、 `allowVolumeExpansion`フィールドを `true`に設定して、ボリューム拡張を可能にするようにストレージクラスを設定する必要があります：

```
cat storageclass-ontapnas.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

このオプションを使用せずにすでにストレージクラスを作成している場合は、 `kubectl edit storageclass`を使

用して既存のストレージクラスを編集し、ボリュームの拡張を可能にすることができます。

手順2：作成したStorageClassを使用してPVCを作成する

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident は、この PVC に対して 20 MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES    STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ3：PVを拡張する

新しく作成した20 MiBのPVを1 GiBにサイズ変更するには、PVCを編集して
`spec.resources.requests.storage`を1 GiBに設定します：

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

ステップ**4**：拡張を検証する

PVC、PV、および Trident ボリュームのサイズを確認することで、サイズ変更が正しく機能したことを検証できます：

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
```

ボリュームをインポート

`tridentctl import`を使用するか、Tridentインポート
アノテーションを使用して永続ボリューム要求（PVC）を作成することで、既存のストレージ
ボリュームをKubernetes PVとしてインポートできます。

概要と考慮事項

次の目的でボリュームをTridentにインポートすることができます：

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタを再構築する
- 災害復旧時にアプリケーションデータを移行する

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Tridentでインポートできるのは RW（読み書き）タイプの ONTAP ボリュームのみです。DP（データ保護）タイプのボリュームは SnapMirror デスティネーションボリュームです。ボリュームを Trident にインポートする前に、ミラー関係を解除する必要があります。
- アクティブな接続なしでボリュームをインポートすることをお勧めします。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

> これは、Kubernetes が以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、ブロックボリュームの場合に特に重要です。その結果、データ破損が発生する可能性があります。

- ただし `StorageClass`PVCで指定する必要がありますが、Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。したがって、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在する場合でも、インポートは失敗しません。
- 既存のボリュームサイズが決定され、PVCに設定されます。ボリュームがストレージドライバによってインポートされると、ClaimRefを使用してPVCへの参照を持つPVが作成されます。
  - 回収ポリシーは初期設定では PV で `retain`に設定されています。Kubernetes が PVC と PV を正常にバインドすると、回収ポリシーはストレージクラスの回収ポリシーと一致するように更新されます。
  - ストレージクラスの回収ポリシーが `delete`の場合、PV が削除されると、ストレージボリュームも削除されます。
- デフォルトでは、TridentがPVCを管理し、バックエンドのFlexVolボリュームとLUNの名前を変更します。 `--no-manage`フラグを渡して管理されていないボリュームをインポートし、 `--no-rename`フラグを渡してボリューム名を保持できます。
  - `--no-manage`* - `--no-manage`フラグを使用する場合、Tridentはオブジェクトのライフサイクル中、PVCまたはPVに対して追加の操作は実行されません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などの他の操作も無視されます。
  - `--no-rename`* - `--no-rename`フラグを使用する場合、`Trident`はボリュームをインポートするときに既存のボリューム名を保持し、ボリュームのライフサイクルを管理します。このオプションは、`ontap-nas`、 `ontap-san`（ASA r2システムを含む）、および `ontap-san-economy`ドライバーでのみサポートされます。

> これらのオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、それ以外の場合は Kubernetes の外部でストレージボリュームのライフサイクルを管理する場合に役立ちます。

- PVC と PV に注釈が追加されます。注釈には、ボリュームがインポートされたことと、PVC と PV が管理されているかどうかを示すという 2 つの目的があります。この注釈は変更または削除しないでください。

ボリュームをインポートする

`tridentctl import`を使用するか、`Trident`インポートアノテーションを使用して`PVC`を作成することで、ボリュームをインポートできます。

PVC アノテーションを使用する場合は、 `tridentctl`をダウンロードしたり使用してボリュームをインポートしたりする必要はありません。

tridentctlの使用

手順

1. PVCを作成するために使用するPVCファイル（例： `pvc.yaml`）を作成します。PVCファイルには `name`、 `namespace`、 `accessModes`、および `storageClassName`を含める必要があります。必要に応じて、PVC定義で `unixPermissions`を指定することもできます。

   以下は最小仕様の例です：

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: my_claim
     namespace: my_namespace
   spec:
     accessModes:
       - ReadWriteOnce
     storageClassName: my_storage_class
   ```

   必須パラメータのみを入力してください。PV名やボリュームサイズなどの追加パラメータは、インポートコマンドの失敗の原因となる可能性があります。

2. `tridentctl import`コマンドを使用して、ボリュームを含むTridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（例：ONTAP FlexVol、Element Volume）を指定します。 `-f`引数は、PVCファイルへのパスを指定するために必要です。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

PVCアノテーションの使用

手順

1. 必要なTridentインポートアノテーションを含むPVC YAMLファイル（例： `pvc.yaml`）を作成します。PVCファイルには以下を含める必要があります：

   - `name`および `namespace`メタデータ内
   - `accessModes`、 `resources.requests.storage`、および `storageClassName`仕様
   - 注釈：
     - `trident.netapp.io/importOriginalName`：バックエンドのボリューム名
     - `trident.netapp.io/importBackendUUID`：ボリュームが存在するバックエンドUUID
     - `trident.netapp.io/notManaged`（オプション）：管理されていないボリュームの場合は `"true"`に設定します。デフォルトは `"false"`です。

       以下は、管理対象ボリュームをインポートするための仕様例です：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. PVC YAML ファイルを Kubernetes クラスターに適用します：

```
kubectl apply -f <pvc-file>.yaml
```

Trident はボリュームを自動的にインポートし、PVC にバインドします。

例

サポートされているドライバーの次のボリュームインポート例を確認してください。

ONTAP NAS と ONTAP NAS FlexGroup

Tridentは、 `ontap-nas`および `ontap-nas-flexgroup`ドライバーを使用したボリュームインポートをサポートしています。

- Tridentは `ontap-nas-economy`ドライバを使用したボリュームインポートをサポートしていません。
- `ontap-nas`および `ontap-nas-flexgroup`ドライバは、重複するボリューム名を許可しません。

`ontap-nas`ドライバーで作成された各ボリュームは、ONTAPクラスタ上のFlexVolボリュームです。 `ontap-nas`ドライバーを使用したFlexVolボリュームのインポートも同様に機能します。ONTAPクラスタにすでに存在するFlexVolボリュームは、 `ontap-nas` PVCとしてインポートできます。同様に、FlexGroupボリュームは `ontap-nas-flexgroup` PVCとしてインポートできます。

tridentctl を使用した **ONTAP NAS** の例

次の例は、 `tridentctl`を使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、 `ontap_nas`というバックエンド上の `managed_volume`というボリュームをインポートします：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
```

管理されていないボリューム

`--no-manage`引数を使用する場合、Tridentはボリュームの名前を変更しません。

次の例は、 `unmanaged_volume`を `ontap_nas`バックエンドでインポートします：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
```

PVC アノテーションを使用した **ONTAP NAS** の例

次の例は、PVC アノテーションを使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理ボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、 `81abcb27-ea63-49bb-b606-0a5315ac5f21`から `ontap_volume1`という名前の `ontap-nas`ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

管理されていないボリューム

次の例では、PVC アノテーションを使用して RWO アクセス モードが設定された、バックエンド `34abcb27-ea63-49bb-b606-0a5315ac5f34`からという名前 `ontap-volume2`の `1Gi` `ontap-nas` ボリュームをインポートします：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Tridentは、 `ontap-san`（iSCSI、NVMe/TCP、FC）および `ontap-san-economy`ドライバーを使用したボリュームインポートをサポートしています。

Tridentは、単一のLUNを含むONTAP SAN FlexVolボリュームをインポートできます。これは `ontap-san`ドライバと一致しており、各PVCに対してFlexVolボリュームを作成し、FlexVolボリューム内にLUNを作成します。TridentはFlexVolボリュームをインポートし、PVC定義に関連付けます。Tridentは、複数のLUNを含む `ontap-san-economy`ボリュームをインポートできます。

次の例は、管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています：

管理ボリューム

管理対象ボリュームの場合、TridentはFlexVolボリュームの名前を `pvc-<uuid>`形式に変更し、FlexVolボリューム内のLUNの名前も `lun0`に変更します。

次の例では、 `ontap-san-managed`バックエンド上に存在するFlexVol volumeを `ontap_san_default`インポートします：

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理されていないボリューム

次の例は、 `unmanaged_example_volume`を `ontap_san`バックエンドでインポートします：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

次の例に示すように、Kubernetes ノード IQN と IQN を共有する igroup にマップされた LUN がある場合は、次のエラーが表示されます： `LUN already mapped to initiator(s) in this group`ボリュームをインポートするには、イニシエーターを削除するか、LUN のマップを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

要素

Tridentは、 `solidfire-san`ドライバを使用して、NetApp Element ソフトウェアおよびNetApp HCI ボリュームのインポートをサポートします。

> Element ドライバは重複したボリューム名をサポートします。ただし、 Trident は重複したボリューム名がある場合はエラーを返します。回避策として、ボリュームのクローンを作成し、一意のボリューム名を指定して、クローンされたボリュームをインポートします。

次の例は、バックエンド `element_default`上の `element-managed`ボリュームをインポートします。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Tridentは `azure-netapp-files`ドライバを使用したボリュームインポートをサポートしています。

> Azure NetApp Files ボリュームをインポートするには、ボリュームパスでボリュームを識別します。ボリュームパスは、 `:/`の後のボリュームのエクスポートパスの部分です。たとえば、マウントパスが `10.0.0.2:/importvol1`の場合、ボリュームパスは `importvol1`です。

次の例は、バックエンド `azurenetappfiles_40517`上の `azure-netapp-files`ボリュームを、ボリュームパス `importvol1`でインポートします。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+------------------------------------------+---------+--------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+--------------
+----------+------------------------------------------+---------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+------------------------------------------+---------+--------------
+----------+------------------------------------------+---------+---------+
```

Google Cloud NetApp Volumes

Tridentは `google-cloud-netapp-volumes`ドライバを使用したボリュームインポートをサポートしています。

次の例では、ボリューム `testvoleasiaeast1`を持つバックエンド `backend-tbc-gcnv1`上のボリュームをインポートします。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

次の例では、同じリージョンに2つのボリュームが存在する場合に `google-cloud-netapp-volumes`ボリュームをインポートします：

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

ボリューム名とラベルをカスタマイズする

Tridentを使用すると、作成したボリュームに意味のある名前とラベルを割り当てることができます。これにより、ボリュームを識別し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできるようになります。バックエンドレベルでテンプレートを定義して、カスタムボリューム名とカスタムラベルを作成することもできます。作成、インポート、またはクローンを作成するボリュームはすべてテンプレートに準拠します。

開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

- ボリュームの作成、インポート、およびクローン処理。
- `ontap-nas-economy`ドライバの場合、Qtreeボリュームの名前のみが名前テンプレートに準拠します。
- `ontap-san-economy`ドライバの場合、LUN名のみが名前テンプレートに準拠します。

制限事項

- カスタムボリューム名は ONTAP オンプレミスドライバーのみと互換性があります。
- カスタムラベルは、 `ontap-san`、 `ontap-nas`、および `ontap-nas-flexgroup`ドライバでのみサポートされます。
- カスタムボリューム名は既存のボリュームには適用されません。

カスタマイズ可能なボリューム名の主な動作

- 名前テンプレートの構文が無効であるために障害が発生した場合、バックエンドの作成は失敗します。ただし、テンプレートの適用に失敗した場合、ボリュームは既存の命名規則に従って命名されます。
- バックエンド構成の名前テンプレートを使用してボリュームに名前を付ける場合、ストレージプレフィックスは適用されません。必要なプレフィックス値をテンプレートに直接追加できます。

名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベル、あるいはその両方で定義できます。

ルートレベルの例

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

プールレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

名前 **template** の例

例**1**：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

例**2**：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

考慮すべき点

1. ボリュームのインポートの場合、既存のボリュームに特定の形式のラベルがある場合にのみラベルが更新されます。例えば：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`
2. 管理対象ボリュームのインポートの場合、ボリューム名はバックエンド定義のルートレベルで定義された名前テンプレートに従います。
3. Trident は、ストレージプレフィックスでのスライス演算子の使用をサポートしていません。
4. テンプレートによって一意のボリューム名が生成されない場合は、Tridentがいくつかのランダムな文字を追加して、一意のボリューム名を作成します。
5. NASエコノミーボリュームのカスタム名が64文字を超える場合、Tridentは既存の命名規則に従ってボリューム名を付けます。その他のすべてのONTAPドライバーでは、ボリューム名が名前制限を超えると、ボリューム作成プロセスが失敗します。

名前空間間で**NFS**ボリュームを共有する

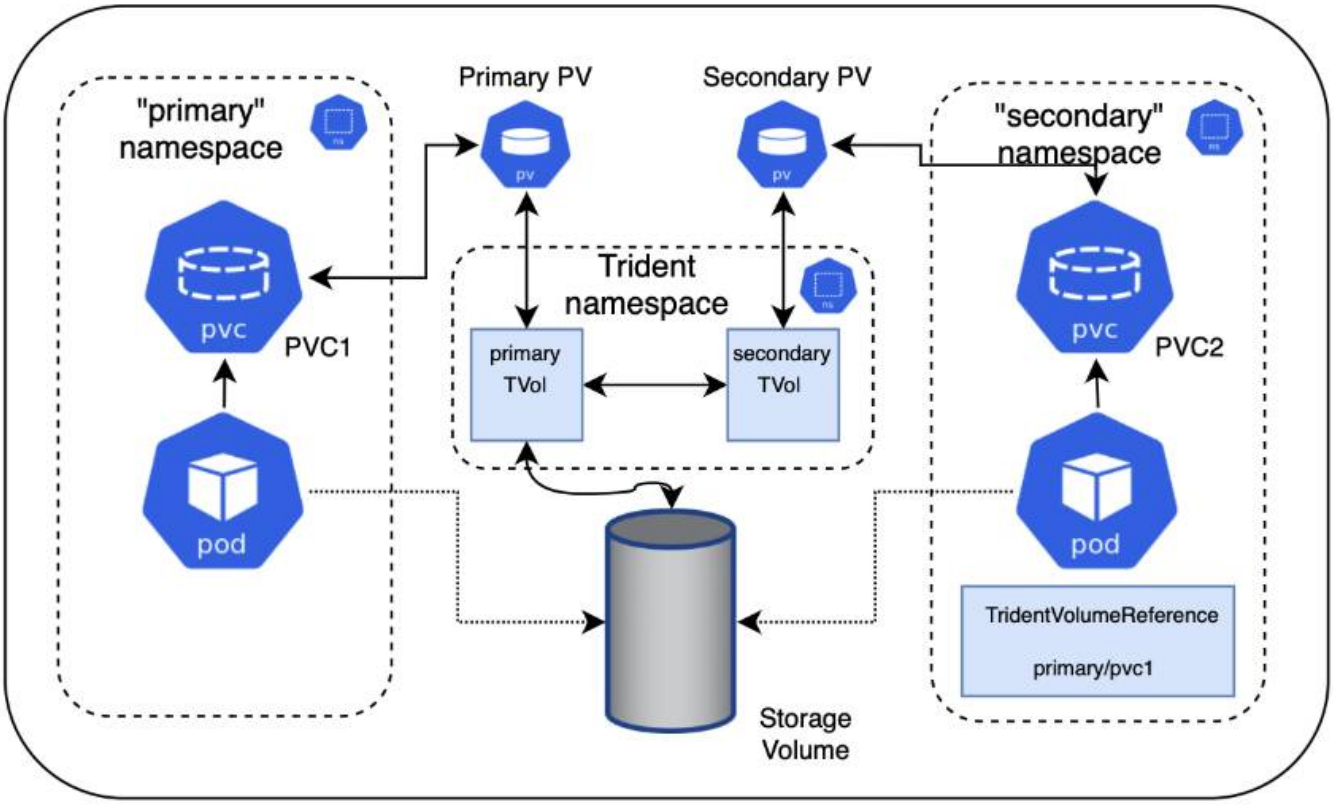
Tridentを使用すると、プライマリ名前空間にボリュームを作成し、それを1つ以上のセカンダリ名前空間で共有できます。

機能

TridentVolumeReference CR を使用すると、ReadWriteMany（RWX）NFS ボリュームを 1 つ以上の Kubernetes 名前空間にわたって安全に共有できます。この Kubernetes ネイティブソリューションには、次の利点があります：

- セキュリティを確保するための複数レベルのアクセス制御
- すべての Trident NFS ボリュームドライバーに対応
- tridentctl やその他の非ネイティブ Kubernetes 機能に依存しない

この図は、2 つの Kubernetes 名前空間間での NFS ボリュームの共有を示しています。



クイック スタート

わずか数ステップで NFS ボリューム共有を設定できます。

1 ボリュームを共有するようにソース **PVC** を設定する

ソース名前空間の所有者は、ソース PVC 内のデータにアクセスする権限を付与します。

2 宛先ネームスペースに **CR** を作成する権限を付与します

クラスタ管理者は、宛先名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。

3 宛先名前空間に**TridentVolumeReference**を作成する

宛先名前空間の所有者は、ソースPVCを参照するTridentVolumeReference CRを作成します。

4 宛先ネームスペースに従属**PVC**を作成する

宛先ネームスペースの所有者は、ソース PVC のデータソースを使用するために従属 PVC を作成します。

ソースネームスペースと宛先ネームスペースを設定する

セキュリティを確保するには、名前空間間の共有には、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による連携とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. ＊ソースネームスペースの所有者＊：ソースネームスペースでPVC(`pvc1`を作成し、(`namespace2`宛てに共有する権限を付与するために、 `shareToNamespace`アノテーションを使用します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident は PV とそのバックエンド NFS ストレージボリュームを作成します。

- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。例えば、 `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、 `*`を使用します。例： `trident.netapp.io/shareToNamespace: *`
- PVCを更新して、 `shareToNamespace`アノテーションをいつでも含めることができます。

2. *クラスタ管理者：*適切なRBACが設定されていることを確認し、宛先ネームスペースの所有者が宛先ネームスペースにTridentVolumeReference CRを作成する権限を付与します。
3. 宛先名前空間の所有者： ソース名前空間を参照するTridentVolumeReference CRを宛先名前空間に作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者： 宛先ネームスペース(`namespace2`）でPVC(`pvc2`）を作成し、ソー

スPVCを指定するために `shareFromPVC`アノテーションを使用します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

デスティネーション PVC のサイズは、ソース PVC のサイズ以下である必要があります。

結果

Tridentは、デスティネーションPVCの `shareFromPVC`アノテーションを読み取り、デスティネーションPVを、独自のストレージリソースを持たず、ソースPVを指してソースPVストレージリソースを共有する従属ボリュームとして作成します。デスティネーションPVCとPVは正常にバインドされているように見えます。

共有ボリュームを削除する

複数の名前空間間で共有されているボリュームを削除できます。Tridentは、ソース名前空間のボリュームへのアクセスを削除し、ボリュームを共有する他の名前空間へのアクセスを維持します。ボリュームを参照するすべての名前空間が削除されると、Tridentはボリュームを削除します。

`tridentctl get`を使用して従属ボリュームを照会する

[`tridentctl`ユーティリティを使用して、 `get`コマンドを実行すると、従属ボリュームを取得できます。詳細については、link：../trident-reference/trident-cl.html[`tridentctl`コマンドとオプション]を参照してください。

```
Usage:
  tridentctl get [option]
```

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを従属ソースボリュームに制限します。
- `--subordinateOf string`：ボリュームの下位にクエリを制限します。

制限事項

- Tridentは、宛先ネームスペースが共有ボリュームに書き込むのを防ぐことはできません。共有ボリュームのデータが上書きされないようにするには、ファイルロックまたはその他のプロセスを使用する必要があります。
- ソースPVCへのアクセスを取り消すには、 `shareToNamespace`または `shareFromNamespace`アノテーションを削除するか、 `TridentVolumeReference`CRを削除しても取り消すことはできません。アクセスを取り消すには、従属PVCを削除する必要があります。
- 従属ボリュームではスナップショット、クローン、ミラーリングは実行できません。

詳細情報

クロスネームスペースボリュームアクセスの詳細については、次を参照してください：

- "名前空間間でボリュームを共有する：クロスネームスペースボリュームアクセスの実現"にアクセスしてください。
- "NetAppTV"でデモをご覧ください。

名前空間を越えてボリュームを複製する

Tridentを使用すると、同じKubernetesクラスター内の別の名前空間の既存のボリュームまたはボリュームスナップショットを使用して、新しいボリュームを作成できます。

前提条件

ボリュームを複製する前に、ソースと宛先のバックエンドが同じタイプであり、同じストレージクラスを持っていることを確認してください。

名前空間をまたがるクローン作成は、 `ontap-san`および `ontap-nas`ストレージドライバでのみサポートされます。読み取り専用クローンはサポートされていません。

クイック スタート

わずか数ステップでボリュームのクローンを設定できます。

1. ボリュームをクローニングするソース **PVC** を設定します

ソース名前空間の所有者は、ソース PVC 内のデータにアクセスする権限を付与します。

2. 宛先ネームスペースに **CR** を作成する権限を付与します

クラスタ管理者は、宛先名前空間の所有者にTridentVolumeReference CRを作成する権限を付与します。

3. 宛先名前空間に**TridentVolumeReference**を作成する

宛先名前空間の所有者は、ソースPVCを参照するTridentVolumeReference CRを作成します。

4 宛先ネームスペースにクローン **PVC** を作成する

宛先ネームスペースの所有者は、ソースネームスペースから PVC をクローニングするための PVC を作成します。

ソースネームスペースと宛先ネームスペースを設定する

セキュリティを確保するために、名前空間間でボリュームを複製するには、ソース名前空間の所有者、クラスター管理者、および宛先名前空間の所有者による共同作業とアクションが必要です。各ステップでユーザーロールが指定されます。

手順

1. ソースネームスペースの所有者： ソースネームスペース(`namespace1`）でPVC(`pvc1`）を作成し、`cloneToNamespace`アノテーションを使用して、宛先ネームスペース(`namespace2`）と共有する権限を付与します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident は PV とそのバックエンドストレージボリュームを作成します。

- コンマ区切りのリストを使用して、PVC を複数の名前空間で共有できます。例えば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- すべての名前空間に共有するには、 `*`を使用します。例えば、`trident.netapp.io/cloneToNamespace: *`
- PVC を更新して、 `cloneToNamespace`アノテーションをいつでも含めることができます。

2. クラスタ管理者： 適切なRBACが設定されていることを確認し、宛先名前空間の所有者が宛先名前空間にTridentVolumeReference CRを作成する権限を付与します(`namespace2`）。
3. 宛先名前空間の所有者： ソース名前空間を参照するTridentVolumeReference CRを宛先名前空間に作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者： 宛先ネームスペース(namespace2）でPVC(pvc2）を作成し、cloneFromPVC`または `cloneFromSnapshot、および `cloneFromNamespace`アノテーションを使用してソースPVCを指定します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

制限事項

- ontap-nas-economy ドライバーを使用してプロビジョニングされた PVC の場合、読み取り専用クローンはサポートされません。

SnapMirrorを使用したボリュームのレプリケート

Tridentは、ディザスタリカバリのためにデータをレプリケートする、1つのクラスタ上のソースボリュームとピアクラスタ上のデスティネーションボリューム間のミラー関係をサポートします。 Trident Mirror Relationship（TMR）と呼ばれる名前空間付きのカスタムリソース定義（CRD）を使用して、次の操作を実行できます：

- ボリューム間のミラー関係を作成する（PVC）
- ボリューム間のミラー関係を削除する

- ミラー関係を解除
- 災害時（フェイルオーバー）にセカンダリ ボリュームを昇格する
- 計画されたフェイルオーバーまたは移行中に、クラスタ間でのアプリケーションのロスレス移行を実行します

レプリケーションの前提条件

始める前に、次の前提条件が満たされていることを確認してください：

ONTAP クラスタ

- **Trident**：ONTAPをバックエンドとして利用するソースとデスティネーションの両方のKubernetesクラスタに、Tridentバージョン22.10以降が存在している必要があります。
- ライセンス：ONTAP SnapMirror データ保護バンドルを使用する非同期ライセンスは、ソースとデスティネーション クラスタの両方で有効にする必要があります。詳細については、 "ONTAP における SnapMirror ライセンスの概要"を参照してください。

  ONTAP 9.10.1以降では、すべてのライセンスがNetAppライセンス ファイル（NLF）として提供されます。これは、複数の機能を有効にする単一のファイルです。詳細については、"ONTAP Oneに含まれるライセンス"を参照してください。

  SnapMirror非同期保護のみがサポートされています。

ピアリング

- * クラスタと SVM * ： ONTAP ストレージ バックエンドはピアリングする必要があります。詳細については、 "クラスタとSVMのピアリングの概要"を参照してください。

  2つのONTAPクラスタ間のレプリケーション関係で使用されるSVM名が一意であることを確認してください。

- **Trident**および**SVM**：ピアリングされたリモートSVMは、デスティネーション クラスタのTridentで使用できる必要があります。

サポートされているドライバ

NetApp Tridentは、次のドライバーでサポートされているストレージクラスを使用して、NetApp SnapMirror テクノロジによるボリュームレプリケーションをサポートします： **`ontap-nas`**：**NFS** `ontap-san`：iSCSI **`ontap-san`**：**FC** `ontap-san`：NVMe/TCP（最小ONTAPバージョン9.15.1が必要）

SnapMirrorを使用したボリュームレプリケーションは、ASA r2システムではサポートされていません。ASA r2システムの詳細については、"ASA r2ストレージシステムの詳細"を参照してください。

ミラー **PVC** を作成する

次の手順に従って、CRD の例を使用して、プライマリ ボリュームとセカンダリ ボリューム間のミラー関係を作成します。

手順

1. プライマリ Kubernetes クラスタで次の手順を実行します：

   a. `trident.netapp.io/replication: true`パラメータを使用してStorageClassオブジェクトを作成します。

      例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 以前に作成したStorageClassを使用してPVCを作成します。

      例

      ```
      kind: PersistentVolumeClaim
      apiVersion: v1
      metadata:
        name: csi-nas
      spec:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 1Gi
        storageClassName: csi-nas
      ```

   c. ローカル情報を含むMirrorRelationship CRを作成します。

      例

      ```
      kind: TridentMirrorRelationship
      apiVersion: trident.netapp.io/v1
      metadata:
        name: csi-nas
      spec:
        state: promoted
        volumeMappings:
        - localPVCName: csi-nas
      ```

      Tridentはボリュームの内部情報とボリュームの現在のデータ保護（DP）状態を取得

し、MirrorRelationshipのstatusフィールドに入力します。

d. TridentMirrorRelationship CR を取得して、PVC の内部名と SVM を取得します。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. セカンダリ Kubernetes クラスタで次の手順を実行します：

a. StorageClassをtrident.netapp.io/replication: trueパラメータで作成します。

例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

b. デスティネーションとソースの情報を含むMirrorRelationship CRを作成します。

例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Tridentは、設定された関係ポリシー名（またはONTAPのデフォルト）でSnapMirror関係を作成し、初期化します。

c. 以前に作成したStorageClassを使用してPVCを作成し、セカンダリ（SnapMirrorデスティネーション）として機能させます。

例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

TridentはTridentMirrorRelationship CRDを確認し、関係が存在しない場合はボリュームの作成に失敗します。関係が存在する場合、Tridentは新しいFlexVolボリュームが、MirrorRelationshipで定義されたリモートSVMとピアリングされたSVM上に配置されるようにします。

ボリュームレプリケーションの状態

Tridentミラー リレーションシップ（TMR）は、PVC間のレプリケーション リレーションシップの一方の端を表すCRDです。デスティネーションTMRには状態があり、Tridentに望ましい状態を伝えます。デスティネーションTMRの状態は次のとおりです：

- 確立済み：ローカル PVC はミラー関係のデスティネーション ボリュームであり、これは新しい関係です。

- 昇格：ローカルPVCはReadWriteでマウント可能ですが、現在ミラー関係は有効ではありません。
- 再確立：ローカル PVC はミラー関係のデスティネーション ボリュームであり、以前もそのミラー関係にありました。
  - デスティネーション ボリュームがソース ボリュームと関係があった場合、デスティネーション ボリュームの内容が上書きされるため、再確立された状態を使用する必要があります。
  - ボリュームが以前にソースと関係していなかった場合、再確立された状態は失敗します。

計画外のフェイルオーバー中にセカンダリ **PVC** を昇格する

セカンダリ Kubernetes クラスタで次の手順を実行します：

- TridentMirrorRelationshipの_spec.state_フィールドを `promoted`に更新します。

計画されたフェイルオーバー中にセカンダリ **PVC** を昇格する

計画されたフェイルオーバー（移行）中に、次の手順を実行してセカンダリ PVC を昇格します：

手順

1. プライマリ Kubernetes クラスタで、PVC のスナップショットを作成し、スナップショットが作成されるまで待機します。
2. プライマリKubernetesクラスタで、SnapshotInfo CRを作成して内部の詳細を取得します。

   例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. セカンダリKubernetesクラスタで、*TridentMirrorRelationship* CRの_spec.state_フィールドを_promoted_に更新し、_spec.promotedSnapshotHandle_をスナップショットのinternalNameに更新します。
4. セカンダリKubernetesクラスタで、TridentMirrorRelationshipのステータス（status.stateフィールド）がpromotedになっていることを確認します。

フェイルオーバー後にミラー関係を復元する

ミラー関係をリストアする前に、新しいプライマリとして作成する側を選択します。

手順

1. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.remoteVolumeHandle_フィールドの値が更新されていることを確認します。
2. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.mirror_フィールドを `reestablished`に更新します。

追加操作

Tridentは、プライマリボリュームとセカンダリボリュームで次の操作をサポートします：

プライマリ**PVC**を新しいセカンダリ**PVC**にレプリケートする

プライマリ PVC とセカンダリ PVC が既に存在することを確認します。

手順

1. 確立されたセカンダリ（デスティネーション）クラスタからPersistentVolumeClaimとTridentMirrorRelationship CRDを削除します。
2. プライマリ（ソース）クラスタから TridentMirrorRelationship CRD を削除します。
3. 新しいセカンダリ（デスティネーション） PVC を確立するために、プライマリ（ソース）クラスタ上に新しい TridentMirrorRelationship CRD を作成します。

ミラーリングされたプライマリまたはセカンダリ **PVC** のサイズを変更する

PVCは通常通りサイズ変更できます。ONTAPは、データの量が現在のサイズを超える場合、デスティネーションのflevxolを自動的に拡張します。

PVCからレプリケーションを削除する

レプリケーションを削除するには、現在のセカンダリ ボリュームで次のいずれかの操作を実行します：

- セカンダリ PVC 上のMirrorRelationshipを削除します。これにより、レプリケーション関係が切断されます。
- または、spec.state フィールドを *promoted* に更新します。

PVC を削除する（以前にミラーリングされたもの）

Tridentはレプリケートされた PVC をチェックし、ボリュームの削除を試みる前にレプリケーション関係を解放します。

TMRを削除する

ミラー関係の片側でTMRを削除すると、残りのTMRは_promoted_状態に移行してから、Tridentが削除を完了します。削除対象として選択されたTMRがすでに_promoted_状態にある場合、既存のミラー関係は存在せず、TMRは削除され、Tridentはローカル PVC を *ReadWrite* に昇格します。この削除により、ONTAPのローカルボリュームのSnapMirrorメタデータが解放されます。このボリュームが将来ミラー関係で使用される場合は、新しいミラー関係を作成するときに、_established_ボリュームレプリケーション状態を持つ新しいTMRを使用する必要があります。

ONTAPがオンラインのときにミラー関係を更新する

ミラー関係は、確立された後はいつでも更新できます。 `state: promoted`または `state: reestablished`フィールドを使用して関係を更新できます。デスティネーションボリュームを通常のReadWriteボリュームに昇格する場合は、_promotedSnapshotHandle_を使用して、現在のボリュームをリストアする特定のSnapshotを指定できます。

ONTAPがオフラインのときにミラー関係を更新する

CRDを使用すると、TridentがONTAPクラスタに直接接続しなくてもSnapMirror更新を実行できます。TridentActionMirrorUpdateの次の例の形式を参照してください：

例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state`は、TridentActionMirrorUpdate CRDの状態を反映しています。*Succeeded*、*In Progress*、または_Failed_のいずれかの値を取ることができます。

CSI トポロジを使用する

Trident は、 "CSI Topology機能"を使用して、Kubernetes クラスタ内のノードにボリュームを選択的に作成して接続することができます。

概要

CSI トポロジ機能を使用すると、リージョンとアベイラビリティーゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウド プロバイダーは Kubernetes 管理者がゾーン ベースのノードを生成できるようにしています。ノードは、リージョン内の異なるアベイラビリティーゾーン、または複数のリージョンに配置できます。マルチゾーンアーキテクチャにおけるワークロードのボリュームのプロビジョニングを容易にするために、Trident は CSI トポロジを使用します。

CSI トポロジー機能の詳細 "ここをクリックしてください。"

Kubernetes は、2 つの独自のボリューム バインディング モードを提供します：

- `VolumeBindingMode`を `Immediate`に設定すると、Tridentはトポロジを意識せずにボリュームを作成します。ボリュームのバインディングと動的プロビジョニングは、PVCの作成時に処理されます。これはデフォルトの `VolumeBindingMode`であり、トポロジ制約を強制しないクラスターに適しています。永続ボリュームは、要求元のポッドのスケジュール要件に依存せずに作成されます。
- `VolumeBindingMode`を `WaitForFirstConsumer`に設定すると、PVC の永続ボリュームの作成とバインドは、PVC を使用するポッドがスケジュールされて作成されるまで遅延されます。このようにして、トポロジ要件によって適用されるスケジュール制約を満たすボリュームが作成されます。

`WaitForFirstConsumer`バインディング モードではトポロジ ラベルは必要ありません。これは、CSI トポロジ機能とは独立して使用できます。

要件

CSI トポロジを利用するには、次のものが必要です：

- "サポートされている Kubernetes バージョン"を実行しているKubernetesクラスタ

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードにはトポロジー認識を導入するラベルが必要です（(`topology.kubernetes.io/region`および `topology.kubernetes.io/zone`）。これらのラベルは、Tridentがトポロジを認識できるようにするために、Tridentをインストールする前に*クラスタ内のノードに存在している必要があります*。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

ステップ**1**：トポロジー対応のバックエンドを作成する

Tridentストレージ バックエンドは、可用性ゾーンに基づいてボリュームを選択的にプロビジョニングするように設計できます。各バックエンドには、サポートされているゾーンとリージョンのリストを表すオプションの `supportedTopologies`ブロックを含めることができます。このようなバックエンドを使用す

るStorageClassesの場合、ボリュームは、サポートされているリージョン/ゾーンでスケジュールされているアプリケーションによって要求された場合にのみ作成されます。

バックエンドの定義の例を次に示します：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

`supportedTopologies`は、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、StorageClassで指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むStorageClassesの場合、Tridentはバックエンドにボリュームを作成します。

`supportedTopologies`をストレージプールごとに定義することもできます。次の例を参照してください：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

この例では、 `region`および `zone`ラベルはストレージ プールの場所を表します。 `topology.kubernetes.io/region`および `topology.kubernetes.io/zone`は、ストレージ プールをどこから使用できるかを指定します。

ステップ**2**：トポロジーを認識する**StorageClasses**を定義する

クラスタ内のノードに提供されるトポロジラベルに基づいて、StorageClassesをトポロジ情報を含めるように定義できます。これにより、PVC要求の候補となるストレージプールと、Tridentによってプロビジョニングされたボリュームを利用できるノードのサブセットが決定されます。

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

上記のStorageClass定義では、 `volumeBindingMode`が `WaitForFirstConsumer`に設定されています。このStorageClassで要求されたPVCは、ポッド内で参照されるまで処理されません。また、 `allowedTopologies`は使用するゾーンとリージョンを提供します。 `netapp-san-us-east1` StorageClassは、上記で定義された `san-backend-us-east1`バックエンドにPVCを作成します。

ステップ**3**：**PVC**を作成して使用する

StorageClassが作成され、バックエンドにマップされたら、PVC を作成できるようになります。

以下の例を参照してください `spec`：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

このマニフェストを使用して PVC を作成すると、次のようになります：

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

Tridentでボリュームを作成してPVCにバインドするには、ポッド内のPVCを使用します。次の例を参照してください：

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

このpodSpecは、Kubernetesに対して、`us-east1`リージョンに存在するノードにポッドをスケジュールし、`us-east1-a`または`us-east1-b`ゾーンに存在する任意のノードから選択するように指示します。

次の出力を参照してください：

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

バックエンドを更新して `**supportedTopologies**`を含める

既存のバックエンドは、 `supportedTopologies`のリストを含めるように `tridentctl backend update`を使用して更新できます。これはすでにプロビジョニングされたボリュームには影響せず、以降のPVCにのみ使用されます。

詳細情報の参照

- "コンテナのリソースを管理する"
- "nodeSelector"
- "親和性と反親和性"
- "TaintsとTolerations"

スナップショットの操作

Kubernetes の Persistent Volume （ PV ）のボリュームスナップショットにより、ポイントインタイムのボリュームコピーが可能になります。Trident を使用して作成されたボリュームのスナップショットを作成したり、 Trident 外部で作成されたスナップショットをインポートしたり、既存のスナップショットから新しいボリュームを作成したり、スナップショットからボリュームデータをリカバリしたりできます。

概要

ボリュームスナップショットは `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `azure-netapp-files`、および `google-cloud-netapp-volumes`ドライバでサポートされています。

開始する前に

スナップショットを操作するには、外部スナップショットコントローラーとカスタムリソース定義（CRD）が必要です。これは Kubernetes オーケストレーター（例： Kubeadm 、 GKE 、OpenShift）の責任です。

Kubernetesディストリビューションにスナップショットコントローラとcrdが含まれていない場合は、ボリュームSnapshotコントローラを導入するを参照してください。

GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEは組み込みの隠しスナップショットコントローラを使用します。

ボリューム **Snapshot** を作成する

手順

1. `VolumeSnapshotClass`を作成します。詳細については、"VolumeSnapshotClass"を参照してください。
   - `driver`はTrident CSIドライバを指しています。
   - `deletionPolicy`は `Delete`または `Retain`に設定できます。 `Retain`に設定すると、`VolumeSnapshot`オブジェクトが削除された場合でも、ストレージクラスター上の基礎となる物理Snapshotは保持されます。

   例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   deletionPolicy: Delete
   ```

2. 既存の PVC のスナップショットを作成します。

   例

   - この例では、既存の PVC のスナップショットを作成します。

   ```
   cat snap.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: pvc1-snap
   spec:
     volumeSnapshotClassName: csi-snapclass
     source:
       persistentVolumeClaimName: pvc1
   ```

   - この例では、 `pvc1`という名前のPVCのボリュームスナップショットオブジェクトを作成し、スナップショットの名前は `pvc1-snap`に設定されます。VolumeSnapshotはPVCに類似しており、実際のスナップショットを表す `VolumeSnapshotContent`オブジェクトに関連付けられています。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 説明することで、 `VolumeSnapshotContent` オブジェクトを `pvc1-snap` VolumeSnapshot用に特定できます。 `Snapshot Content Name`は、このスナップショットに対応するVolumeSnapshotContentオブジェクトを識別します。 `Ready To Use`パラメータは、このスナップショットを使用して新しいPVCを作成できることを示します。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

ボリューム**Snapshot**から**PVC**を作成する

`dataSource`を使用して、VolumeSnapshotという名前の `<pvc-name>`をデータのソースとして使用するPVCを作成できます。PVCが作成されると、ポッドに接続して他のPVCと同じように使用できるようになります。

PVC はソース ボリュームと同じバックエンドに作成されます。"KB：Trident PVC スナップショットからの PVC の作成は代替バックエンドでは実行できません"を参照してください。

次の例では、 `pvc1-snap`をデータソースとして使用してPVCを作成します。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

ボリューム **Snapshot** をインポートする

Tridentは"Kubernetesの事前プロビジョニングされたスナップショットプロセス"をサポートしており、クラスタ管理者が `VolumeSnapshotContent`オブジェクトを作成し、Trident外部で作成されたスナップショットをインポートできるようにします。

開始する前に

Tridentでスナップショットの親ボリュームを作成またはインポートしておく必要があります。

手順

1. *クラスタ管理者：*バックエンドスナップショットを参照する `VolumeSnapshotContent`オブジェクトを作成します。これにより、Tridentでスナップショットワークフローが開始されます。
   - `annotations`のバックエンドスナップショットの名前を `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`として指定します。
   - `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`を `snapshotHandle`で指定します。これは、外部スナップショッターが `ListSnapshots`コールでTridentに提供する唯一の情報です。

     `<volumeSnapshotContentName>`は、CR の命名制約により、バックエンドのスナップショット名と常に一致するとは限りません。

     例

     次の例では、バックエンドスナップショット `snap-01`を参照する `VolumeSnapshotContent`オブジェクトを作成します。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. クラスタ管理者： `VolumeSnapshot`オブジェクトを参照するCRを作成します。 `VolumeSnapshotContent`これにより、指定されたネームスペースで `VolumeSnapshot`を使用するためのアクセスが要求されます。

   例

   次の例では、 `VolumeSnapshot`という名前のCRを作成し、 `VolumeSnapshotContent`という名前の `import-snap-content`を参照します。 `import-snap`

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. *内部処理（操作は不要）：*外部スナップショットツールは新しく作成された `VolumeSnapshotContent`を認識し、 `ListSnapshots`呼び出しを実行します。Tridentは `TridentSnapshot`を作成します。
   - 外部スナップショットは、 `VolumeSnapshotContent`を `readyToUse`に設定し、 `VolumeSnapshot`を `true`に設定します。
   - Tridentが返します `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim`を作成して新しい `VolumeSnapshot`を参照します。ここで、 `spec.dataSource`（または `spec.dataSourceRef`）名は `VolumeSnapshot`名です。

例

次の例では、 `VolumeSnapshot`という名前の `import-snap`を参照するPVCを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

スナップショットを使用してボリュームデータを回復する

`ontap-nas`および `ontap-nas-economy`ドライバーを使用してプロビジョニングされたボリュームの互換性を最大限に高めるために、スナップショットディレクトリはデフォルトでは非表示になっています。 `.snapshot`ディレクトリを有効にして、スナップショットから直接データをリカバリします。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Snapshotコピーをリストアすると、既存のボリューム構成が上書きされます。Snapshotコピーの作成後にボリュームデータに加えられた変更は失われます。

Snapshotからのインプレースボリュームリストア

Tridentは、 `TridentActionSnapshotRestore`（TASR）CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

Tridentは、 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `google-cloud-netapp-volumes`、および `solidfire-san`ドライバでのスナップショットのリストアをサポートしています。

開始する前に

バインドされた PVC と使用可能なボリューム スナップショットが必要です。

- PVC ステータスがバインドされていることを確認します。

```
kubectl get pvc
```

- ボリューム スナップショットが使用できる状態であることを確認します。

```
kubectl get vs
```

手順

1. TASR CRを作成します。この例では、PVC `pvc1`とボリュームSnapshot `pvc1-snapshot`のCRを作成します。

   TASR CR は、PVC と VS が存在する名前空間に存在する必要があります。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. スナップショットから復元するには CR を適用します。この例では snapshot `pvc1`から復元します。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

結果

Tridentはスナップショットからデータを復元します。スナップショットの復元ステータスを確認できます：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- ほとんどの場合、Tridentは失敗した場合に操作を自動的に再試行しません。再度操作を実行する必要があります。
- 管理者アクセス権を持たない Kubernetes ユーザーには、アプリケーション名前空間で TASR CR を作成するために、管理者からの権限付与が必要になる場合があります。

関連するスナップショットを含む **PV** を削除する

関連するスナップショットを持つ永続ボリュームを削除すると、対応するTridentボリュームは「削除中」状態に更新されます。Tridentボリュームを削除するには、ボリュームスナップショットを削除します。

ボリューム**Snapshot**コントローラを導入する

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. Snapshot Controller を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

必要に応じて `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`を開き、 `namespace`をネームスペースに更新します。

関連リンク

- "ボリュームスナップショット"
- "VolumeSnapshotClass"

ボリュームグループのスナップショットを操作する

永続ボリューム（PV）の Kubernetes ボリューム グループ スナップショット NetApp Trident は、複数のボリュームのスナップショット（ボリューム スナップショットのグループ）を作成する機能を提供します。このボリューム グループ スナップショットは、同じポイントインタイムで作成された複数のボリュームからのコピーを表します。

VolumeGroupSnapshotは、ベータ API を備えた Kubernetes のベータ機能です。Kubernetes 1.32は、VolumeGroupSnapshotに必要な最小バージョンです。

ボリュームグループの**Snapshot**を作成する

ボリュームグループスナップショットは、次のストレージドライバでサポートされます：

- `ontap-san`ドライバ - iSCSI および FC プロトコルのみで、NVMe/TCP プロトコルには使用できません。
- `ontap-san-economy` - iSCSI プロトコルのみ。
- `ontap-nas`

> ボリュームグループのスナップショットは、NetApp ASA r2 または AFX ストレージシステムではサポートされていません。

開始する前に

- Kubernetes のバージョンが K8s 1.32 以上であることを確認してください。
- スナップショットを操作するには、外部スナップショットコントローラーとカスタムリソース定義（CRD）が必要です。これは Kubernetes オーケストレーター（例： Kubeadm 、 GKE 、OpenShift）の責任です。

  Kubernetesディストリビューションに外部スナップショットコントローラとCRDが含まれていない場合は、ボリュームSnapshotコントローラを導入するを参照してください。

  > GKE 環境でオンデマンドボリュームグループスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKE は組み込みの隠しスナップショットコントローラを使用します。

- スナップショットコントローラYAMLで、`CSIVolumeGroupSnapshot`機能ゲートを'true'に設定して、ボリュームグループのスナップショットが有効になっていることを確認します。
- ボリュームグループスナップショットを作成する前に、必要なボリュームグループスナップショットクラスを作成します。
- すべてのPVC/ボリュームが同じSVM上にあることを確認して、VolumeGroupSnapshotを作成できるようにします。

手順

- VolumeGroupSnapshot を作成する前に、VolumeGroupSnapshotClass を作成してください。詳細については、"VolumeGroupSnapshotClass" を参照してください。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- 既存のストレージクラスを使用して必要なラベルを持つ PVC を作成するか、これらのラベルを既存の PVC に追加します。

次の例では、 `pvc1-group-snap`をデータソースとして、 `consistentGroupSnapshot: groupA`をラベルとして使用してPVCを作成します。要件に応じてラベルのキーと値を定義します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- PVC で指定されたラベル（(`consistentGroupSnapshot: groupA`）と同じラベルを持つ VolumeGroupSnapshot を作成します。

  この例では、ボリュームグループのスナップショットを作成します：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

グループ**Snapshot**を使用してボリュームデータをリカバリする

ボリューム グループ スナップショットの一部として作成された個々のスナップショットを使用して、個々の永続ボリュームをリストアできます。ボリューム グループ スナップショットをユニットとしてリカバリすることはできません。

volume snapshot restore ONTAP CLIを使用して、ボリュームを以前のスナップショットに記録された状態に復元します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Snapshotコピーをリストアすると、既存のボリューム構成が上書きされます。Snapshotコピーの作成後にボリュームデータに加えられた変更は失われます。

Snapshotからのインプレースボリュームリストア

Tridentは、 `TridentActionSnapshotRestore`（TASR）CRを使用して、スナップショットから迅速なインプレースボリュームリストアを提供します。このCRは命令型のKubernetesアクションとして機能し、操作の完了後は保持されません。

詳細については、"Snapshotからのインプレースボリュームリストア"を参照してください。

関連するグループスナップショットを含む **PV** を削除する

グループボリュームのスナップショットを削除する場合：

- VolumeGroupSnapshotsは、グループ内の個々のスナップショットではなく、グループ全体として削除できます。
- PersistentVolumesにスナップショットが存在する状態で削除された場合、Tridentはそのボリュームを「削除中」状態に移動します。これは、ボリュームを安全に削除する前にスナップショットを削除する必要があるためです。
- グループ化されたスナップショットを使用してクローンが作成され、その後グループを削除する場合、split-on-clone処理が開始され、分割が完了するまでグループを削除することはできません。

ボリューム**Snapshot**コントローラを導入する

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のようにデプロイできます。

手順

1. ボリュームスナップショット CRD を作成します。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.
yaml
```

2. Snapshot Controller を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

必要に応じて `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`を開き、 `namespace`をネームスペースに更新します。

関連リンク

- "VolumeGroupSnapshotClass"
- "ボリュームスナップショット"

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 4 か月ごとにリリースされ、毎年 3 つのメジャーリリースが提供されます。新しいリリースはそれぞれ以前のリリースに基づいて構築され、新しい機能、パフォーマンスの強化、バグ修正、改善が提供されます。Trident の新機能をご利用いただくために、少なくとも年に 1 回は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 クラスター内のすべての名前空間にわたって、Trident インスタンスが 1 つだけインストールされている必要があります。
- Trident 23.07以降では、v1ボリュームスナップショットが必須となり、アルファスナップショットやベータスナップショットはサポートされなくなりました。
- アップグレード時には、Tridentで使用されている `parameter.fsType`に `StorageClasses`を必ず指定することが重要です。既存のボリュームに影響を与えることなく、 `StorageClasses`を削除して再作成できます。
  - これは、SAN ボリュームに "セキュリティコンテキスト"を適用するための要件です。
  - 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[`storage-class-basic.yaml.templ`]や`storage-class-bronze-default.yaml`などの例が含まれています。
  - 詳細については、"既知の問題"を参照してください。

ステップ**1**：バージョンを選択する

Trident のバージョンは日付ベース `YY.MM`の命名規則に従います。「YY」は年の最後の 2 桁、「MM」は月です。ドットリリースは `YY.MM.X`の規則に従います。「X」はパッチレベルです。アップグレード元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から4リリース以内の任意のターゲットリリースへの直接アップグレードを実行できます。たとえば、24.06（または任意の24.06ドットリリース）から25.06に直接アップグレードできます。
- 4リリース期間外のリリースからアップグレードする場合は、複数段階のアップグレードを実行します。"以前のバージョン"からアップグレードする場合は、そのバージョンのアップグレード手順を使用して、4リリース期間に収まる最新のリリースにアップグレードします。たとえば、23.07を実行していて25.06にアップグレードする場合（
  a. 最初に 23.07 から 24.06 にアップグレードします。
  b. 次に、24.06から25.06にアップグレードします。

OpenShift Container Platform で Trident オペレーター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 オペレーターには既知の問題が含まれていますが、21.01.1 で修正されています。詳細については、 "GitHubの問題の詳細"を参照してください。

ステップ**2**：元のインストール方法を決定する

最初に Trident をインストールする際に使用したバージョンを確認するには：

1. `kubectl get pods -n trident`を使用してポッドを検査します。
   - オペレーターポッドがない場合、Trident は `tridentctl`を使用してインストールされました。
   - オペレーターポッドがある場合、Trident は Trident オペレーターを使用して手動または Helm を使用してインストールされました。
2. オペレータポッドがある場合は、 `kubectl describe torc`を使用して、TridentがHelmを使用してインストールされた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ました。
   - Helmラベルがない場合、TridentはTridentオペレーターを使用して手動でインストールされました。

ステップ**3**：アップグレード方法を選択する

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、"インストール方法を切り替える"。Trident をアップグレードするには 2 つのオプションがあります。

- "Tridentオペレータを使用してアップグレードする"

オペレータでアップグレードする前に"オペレーターのアップグレードワークフローを理解する"を確認することをお勧めします。

*

オペレータを使用したアップグレード

オペレーターのアップグレードワークフローを理解する

Tridentオペレーターを使用して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 プロセスを理解する必要があります。これには、Tridentコントローラー、コントローラーポッドとノードポッド、およびローリングアップデートを可能にするノードDaemonSetへの変更が含まれます。

Tridentオペレータのアップグレード処理

Trident のインストールとアップグレードの多くの"Tridentオペレーターを使用する利点"の 1 つは、既存のマウントされたボリュームを中断することなく、Trident と Kubernetes オブジェクトを自動的に処理することです。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。または"ローリングアップデート"。特に、Trident オペレーターは Kubernetes クラスターと通信して次の操作を行います（：）

- Trident コントローラの導入とノード DaemonSet を削除して再作成します。

- Trident Controller PodとTrident Node Podを新しいバージョンに置き換えます。
  - 1 つのノードが更新されない場合でも、残りのノードの更新は妨げられません。
  - 実行中の Trident Node Pod を持つノードのみがボリュームをマウントできます。

Kubernetesクラスタ上のTridentアーキテクチャの詳細については、"Tridentアーキテクチャ"を参照してください。

オペレータのアップグレードワークフロー

Trident オペレータを使用してアップグレードを開始すると：

1. **Trident** オペレータ：
   a. 現在インストールされている Trident のバージョン（バージョン *n*）を検出します。
   b. CRD、RBAC、Trident SVC を含むすべての Kubernetes オブジェクトを更新します。
   c. バージョン *n* の Trident Controller デプロイメントを削除します。
   d. バージョン *n+1* の Trident Controller デプロイメントを作成します。
2. **Kubernetes** は *n+1* 用の Trident Controller Pod を作成します。
3. **Trident** オペレータ：
   a. *n* の Trident ノード DaemonSet を削除します。オペレーターはノード Pod の終了を待機しません。
   b. *n+1* の Trident Node Daemonset を作成します。
4. **Kubernetes** は、Trident Node Pod *n* を実行していないノードに Trident Node Pod を作成します。これにより、ノード上に任意のバージョンの Trident Node Pod が複数存在しないことが保証されます。

Trident Operator または **Helm** を使用して **Trident** インストールをアップグレードする

Trident オペレーターを使用して、手動または Helm を使用して Trident をアップグレードできます。Trident オペレーターのインストールから別の Trident オペレーターのインストールにアップグレードすることも、 `tridentctl`のインストールから Trident オペレーターバージョンにアップグレードすることもできます。Trident オペレーターのインストールをアップグレードする前に"アップグレード方法を選択"を確認してください。

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 オペレータのインストールから、別のクラスタを対象とした Trident オペレータのインストールにアップグレードできます。すべての Trident バージョンでは、クラスタを対象としたオペレータを使用します。

名前空間スコープのオペレータを使用してインストールされたTrident（バージョン20.07から20.10）からアップグレードするには、"インストールされているバージョン"のTridentのアップグレード手順を使用してください。

タスク概要

Trident は、オペレーターをインストールし、Kubernetes バージョンに関連付けられたオブジェクトを作成するために使用できるバンドルファイルを提供します。

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

開始する前に

"サポートされている Kubernetes バージョン"を実行しているKubernetesクラスタを使用していることを確認してください。

手順

1. Trident のバージョンを確認してください：

```
./tridentctl -n trident version
```

2. `operator.yaml`、`tridentorchestrator_cr.yaml`、`post_1_25_bundle.yaml`を、アップグレード先のバージョン（例：25.06）のレジストリとイメージパス、および正しいシークレットで更新します。
3. 現在の Trident インスタンスのインストールに使用された Trident オペレーターを削除します。たとえば、25.02 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 25.02.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

4. 初期インストールを `TridentOrchestrator`属性を使用してカスタマイズした場合は、`TridentOrchestrator`オブジェクトを編集してインストールパラメータを変更できます。これには、オフラインモード用のミラー化されたTridentおよびCSIイメージレジストリの指定、デバッグログの有効化、またはイメージプルシークレットの指定などの変更が含まれる場合があります。
5. 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。_<bundle.yaml>_は`bundle_pre_1_25.yaml`または `bundle_post_1_25.yaml`Kubernetesのバージョンに基づきます。たとえば、Trident 25.06.0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 25.06.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

6. Trident torc を編集して、イメージ 25.06.0 を含めます。

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、クラスタをアップグレードする前に、values.yamlを更新して`excludePodSecurityPolicy`を `true`に設定するか、 `--set excludePodSecurityPolicy=true`を`helm upgrade`コマンドに追加する必要があります。

Trident helm をアップグレードせずに Kubernetes クラスターを 1.24 から 1.25 にアップグレード済みの場合、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、前提条件として次の手順を実行してください：

1. https://github.com/helm/helm-mapkubeapisからhelm-mapkubeapisプラグインをインストールします。
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。クリーンアップされるリソースが一覧表示されます。

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. helm を使用して完全実行を実行し、クリーンアップを実行します。

```
helm mapkubeapis trident  --namespace trident
```

手順

1. "Helmを使用してTridentをインストールしました"の場合は、 `helm upgrade trident netapp-trident/trident-operator --version 100.2506.0`を使用してワンステップでアップグレードできます。Helmリポジトリを追加していない場合、またはアップグレードに使用できない場合：
   a. 最新のTridentリリースを"GitHub の *Assets* セクション"からダウンロードします。
   b. `helm upgrade`コマンドを使用します。 `trident-operator-25.10.0.tgz`にはアップグレード先のバージョンを指定します。

```
helm upgrade <name> trident-operator-25.10.0.tgz
```

> ⓘ 初期インストール時にカスタムオプション（TridentおよびCSIイメージ用のプライベートミラーリポジトリの指定など）を設定した場合は、それらのオプションがアップグレードコマンドに含まれるように、 `helm upgrade`コマンドに `--set`を追加してください。そうしないと、値がデフォルトにリセットされます。

2. `helm list`を実行して、チャートとアプリのバージョンが両方ともアップグレードされたことを確認します。 `tridentctl logs`を実行して、デバッグメッセージを確認します。

`tridentctl`インストールから**Trident**オペレーターへのアップグレード

Tridentオペレーターの最新リリースには、 `tridentctl`インストールからアップグレードできます。既存のバックエンドとPVCは自動的に利用できるようになります。

> ⓘ インストール方法を切り替える前に、"インストール方法間の移行"を確認してください。

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 [25.10.0]
mkdir 25.10.0
cd 25.10.0
wget
https://github.com/NetApp/trident/releases/download/v25.10.0/trident-
installer-25.10.0.tar.gz
tar -xf trident-installer-25.10.0.tar.gz
cd trident-installer
```

2. マニフェストから `tridentorchestrator` CRD を作成します。

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

3. クラスタースコープのオペレーターを同じ名前空間にデプロイします。

```
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 `TridentOrchestrator` CR を作成します。

```
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 が意図したバージョンにアップグレードされたことを確認します。

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v25.10.0
```

tridentctl によるアップグレード

既存のTridentインストールは `tridentctl`を使用して簡単にアップグレードできます。

タスク概要

Trident をアンインストールして再インストールすると、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 デプロイメントで使用される永続ボリューム要求（PVC）と永続ボリューム（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 PV は、Trident がオフラインの間も引き続き使用可能であり、Trident はオンラインに戻った後、その間に作成されたすべての PVC のボリュームをプロビジョニングします。

開始する前に

"アップグレード方法を選択"を使用してアップグレードする前に `tridentctl`を確認してください。

手順

1. `tridentctl`でアンインストールコマンドを実行して、CRDと関連オブジェクトを除く、Tridentに関連付けられた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。"tridentctl を使用して Trident をインストールする"を参照してください。

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まで実行されることを確認してください。

tridentctlを使用してTridentを管理する

`https://github.com/NetApp/trident/releases["Trident`インストーラバンドル"^]には、`Trident`への簡単なアクセスを提供する `tridentctl`コマンドラインユーティリティが含まれています。十分な権限を持つ`Kubernetes`ユーザーは、これを使用して`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

`tridentctl help`を実行すると、 `tridentctl`で使用可能なコマンドのリストを取得できます。または、任意のコマンドに `--help`フラグを追加すると、その特定のコマンドのオプションとフラグのリストを取得できます。

```
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
1つ以上のリソースをTridentから削除します。

get
Tridentから1つ以上のリソースを取得します。

help
あらゆるコマンドに関するヘルプ。

images
Tridentに必要な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をインストールします。

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
バックエンド操作を一時的に停止します。

upgrade
Tridentでリソースをアップグレードします。

version
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

`tridentctl`のヘルプ。

-k, --kubeconfig string

`KUBECONFIG`パスを指定して、コマンドをローカルで実行したり、ある Kubernetes クラスターから別のクラスターに実行したりします。

あるいは、 `KUBECONFIG`変数をエクスポートして特定のKubernetesクラスタを指定し、そのクラスタに `tridentctl`コマンドを発行できます。

-n, --namespace string

Trident 導入の名前空間。

-o, --output string

出力形式。json|yaml|name|wide|ps のいずれか（デフォルト）。

-s, --server string

Trident REST インターフェースのアドレス / 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）でのみリッスンおよびサービスを提供するように構成できます。

コマンドオプションとフラグ

作成する

`create`コマンドを使用して、Tridentにリソースを追加します。

```
tridentctl create [option]
```

オプション

backend：Trident にバックエンドを追加します。

削除

`delete`コマンドを使用して、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

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

`get`コマンドを使用して、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

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：クエリを従属ソース ボリュームに制限します。
--subordinateOf string：ボリュームの下位にクエリを制限します。

画像

`images`フラグを使用して、Tridentに必要なコンテナイメージのテーブルを出力します。

```
tridentctl images [flags]
```

フラグ

-h, --help：画像のヘルプ。
-v, --k8s-version string：Kubernetesクラスターのセマンティック バージョン。

ボリュームのインポート

`import volume`コマンドを使用して、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。

インストール

`install`フラグを使用してTridentをインストールします。

```
tridentctl install [flags]
```

フラグ

`--autosupport-image string`：Autosupport Telemetry のコンテナイメージ（デフォルトは "netapp/trident autosupport:<current-version>"）。
`--autosupport-proxy string`：Autosupport Telemetry を送信するためのプロキシのアドレス/ポート。
`--enable-node-prep`：ノードに必要なパッケージを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 コントローラーの REST API の HTTP リクエストのタイムアウトをオーバーライドします（デフォルトは 1m30s）。
`--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 操作のタイムアウト（デフォルトは 3m0s）。
`--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 のログ形式（text、json）（デフォルトは "text"）。
`--node-prep`：指定されたデータストレージ プロトコルを使用してボリュームを管理するために、Kubernetes クラスタのノードを準備できるように Trident を有効にします。現在、 **`iscsi`**がサポートされている唯一の値です。**`OpenShift 4.19` 以降、この機能でサポートされている `Trident` の最小バージョンは `25.06.1` 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 に自動的に送信しません（デフォルトは true）。
`--silent`：インストール中のほとんどの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。
`--k8s-api-qps`：Kubernetes API リクエストの 1 秒あたりのクエリ数（QPS）の制限（デフォルトは 100、オプション）。
`--use-custom-yaml`：setup 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

`logs`フラグを使用して、Tridentからログを出力します。

```
tridentctl logs [flags]
```

フラグ

`-a、 --archive`：特に指定がない限り、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 ログを収集する Kubernetes ノード名。
`-p、 --previous`：以前のコンテナ インスタンスのログが存在する場合は取得します。
`--sidecars`：サイドカー コンテナのログを取得します。

送信

`send`コマンドを使用して、Tridentからリソースを送信します。

```
tridentctl send [option]
```

オプション

`autosupport`：AutosupportアーカイブをNetAppに送信する。

uninstall

`uninstall`フラグを使用して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にほとんどの出力を無効にします。

更新

`update`コマンドを使用して、Tridentのリソースを変更します。

```
tridentctl update [option]
```

オプション

`backend`：Trident でバックエンドを更新します。

バックエンドの状態を更新する

`update backend state`コマンドを使用して、バックエンド操作を一時停止または再開します。

```
tridentctl update backend state <backend-name> [flag]
```

考慮すべき点

- バックエンドがTridentBackendConfig（tbc）を使用して作成されている場合、そのバックエンドは `backend.json`ファイルを使用して更新できません。

- `userState`がtbcに設定されている場合、 `tridentctl update backend state <backend-name> --user-state suspended/normal`コマンドを使用して変更することはできません。
- tbc 経由で設定した後に tridentctl 経由で `userState`を設定する機能を取り戻すには、 `userState`フィールドを tbc から削除する必要があります。これは `kubectl edit tbc`コマンドを使用して実行できます。 `userState`フィールドが削除されたら、 `tridentctl update backend state`コマンドを使用してバックエンドの `userState`を変更できます。
- `tridentctl update backend state`を使用して `userState`を変更します。また、 `userState`は `TridentBackendConfig`または `backend.json`ファイルを使って更新することもできます。この操作はバックエンドの完全な再初期化を引き起こし、時間がかかる場合があります。

  フラグ

  `-h`、 `--help`：バックエンドの状態に関するヘルプ。
  `--user-state`： `suspended`に設定すると、バックエンド操作が一時停止されます。 `normal`に設定すると、バックエンド操作が再開されます。 `suspended`に設定した場合：

- `AddVolume`と `Import Volume`は一時停止されます。
- `CloneVolume`、 `ResizeVolume`、 `PublishVolume`、 `UnPublishVolume`、 `CreateSnapshot`、 `GetSnapshot`、 `RestoreSnapshot`、 `DeleteSnapshot`、 `RemoveVolume`、 `GetVolumeExternal`、 `ReconcileNodeAccess`は引き続き使用できます。

> `userState`フィールドを使用して、バックエンド構成ファイル `TridentBackendConfig`または `backend.json`でバックエンドの状態を更新することもできます。詳細については、link:../trident-use/backend_options.html["バックエンドを管理するオプション"]およびlink:../trident-use/backend_ops_kubectl.html["kubectlを使用してバックエンド管理を実行する"]を参照してください。

例：

JSON

次の手順に従って、 `userState`を `backend.json`ファイルを使用して更新します：

1. `backend.json`ファイルを編集して、 `userState`フィールドを値「suspended」に設定します。
2. `tridentctl update backend`コマンドと更新された `backend.json`ファイルへのパスを使用してバックエンドを更新します。

   例： `tridentctl update backend -f /<path to backend JSON file>/backend.json -n trident`

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "<redacted>",
  "svm": "nas-svm",
  "backendName": "customBackend",
  "username": "<redacted>",
  "password": "<redacted>",
  "userState": "suspended"
}
```

YAML

`kubectl edit <tbc-name> -n <namespace>`コマンドを使用して、適用後にtbcを編集できます。次の例では、 `userState: suspended`オプションを使用して、バックエンドの状態をsuspendに更新します：

```
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
```

version

`version`フラグを使用して、 `tridentctl`のバージョンと実行中のTridentサービスを出力します。

```
tridentctl version [flags]
```

フラグ

--client：クライアント バージョンのみ (サーバーは不要)。
-h, --help：バージョンのヘルプ。

プラグインのサポート

Tridentctl は kubectl と同様のプラグインをサポートしています。Tridentctl は、プラグインのバイナリファイル名が「tridentctl-<plugin>」というスキームに従い、バイナリが PATH 環境変数にリストされているフォルダーに配置されている場合、プラグインを検出します。検出されたすべてのプラグインは、tridentctl ヘルプのプラグインセクションにリストされます。オプションとして、環境変数 TRIDENTCTL_PLUGIN_PATH でプラグインフォルダー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/）。変数を使用すると、tridenctl は指定されたフォルダー内でのみ検索します。

Tridentを監視

Tridentは、Tridentのパフォーマンスを監視するために使用できるPrometheusメトリクスエンドポイントのセットを提供します。

概要

Trident が提供する指標により、次のことが可能になります。

- Tridentの健全性と構成を監視します。処理がどの程度成功したか、期待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情報を調べて、バックエンドに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の量などを把握します。
- 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します。Trident がバックエンドと通信して処理を実行するのにどれくらい時間がかかるかを確認できます。

> デフォルトでは、Tridentのメトリックはターゲットポートでを介して公開されます`8001`の `/metrics`エンドポイントで公開されます。これらのメトリックは、Tridentのインストール時に*デフォルトで有効*になります。ポート `8444`でHTTPS経由でTridentメトリックを使用するように設定することもできます。

要件

- 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 "コンテナ化されたPrometheusの導入"にすることも、Prometheus を "

ネイティブアプリケーション"として実行することもできます。

ステップ1：Prometheusターゲットを定義する

メトリックを収集し、Tridentが管理するバックエンド、作成するボリュームなどに関する情報を取得するには、Prometheusターゲットを定義する必要があります。"Prometheus Operator ドキュメント"を参照してください。

ステップ2：Prometheus ServiceMonitorを作成する

Tridentメトリクスを使用するには、 `trident-csi`サービスを監視し、 `metrics`ポートでリッスンするPrometheus ServiceMonitorを作成する必要があります。サンプルServiceMonitor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、 `trident-csi`サービスによって返されるメトリックを取得し、特にサービスの`metrics`エンドポイントを検索します。その結果、PrometheusはTridentのメトリックを理解できるように設定されました。

Trident から直接利用可能なメトリックに加えて、kubelet は独自のメトリック エンドポイントを介して多くの `kubelet_volume_*`メトリックを公開します。Kubelet は、接続されているボリューム、およびそれが処理するポッドやその他の内部操作に関する情報を提供できます。を参照してください "ここをクリックしてください。"。

HTTPS経由でTridentメトリクスを使用する

HTTPS（ポート8444）経由でTridentメトリクスを利用するには、ServiceMonitor定義を修正してTLS構成を含める必要があります。また、 `trident-csi`シークレットを `trident`ネームスペースからPrometheusが実行されているネームスペースにコピーする必要があります。これには次のコマンドを使用できます：

```
kubectl get secret trident-csi -n trident -o yaml | sed 's/namespace:
```

```
trident/namespace: monitoring/' | kubectl apply -f -
```

HTTPS メトリックのサンプルServiceMonitor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:
    - interval: 15s
      path: /metrics
      port: https-metrics
      scheme: https
      tlsConfig:
        ca:
          secret:
            key: caCert
            name: trident-csi
        cert:
          secret:
            key: clientCert
            name: trident-csi
        keySecret:
          key: clientKey
          name: trident-csi
        serverName: trident-csi
```

Tridentは、すべてのインストール方法（tridentctl、Helm chart、Operator）でHTTPSメトリクスをサポートします。

- `tridentctl install`コマンドを使用している場合は、 `--https-metrics`フラグを渡してHTTPSメトリックを有効にすることができます。
- Helmチャートを使用している場合は、 `httpsMetrics`パラメータを設定してHTTPSメトリックを有効にできます。

- YAMLファイルを使用している場合、 `--https_metrics`フラグを `trident-main`コンテナに `trident-deployment.yaml`ファイルで追加できます。

ステップ3：PromQLを使用したTridentメトリクスのクエリ

PromQL は、時系列データまたは表形式のデータを返す式を作成するのに適しています。

使用できる PromQL クエリをいくつか示します：

Tridentの健康情報を取得する

- **Trident**からの **HTTP 2XX** レスポンスの割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- ステータスコード別の**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所要時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Trident使用情報を取得する

- 平均ボリュームサイズ

```
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ボリュームスペースの合計

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリューム使用量を取得する

これは、kubelet メトリックも収集される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テレメトリーの詳細

デフォルトでは、Tridentは、Prometheusメトリクスと基本的なバックエンド情報をNetAppに毎日送信します。

- TridentがNetAppにPrometheusメトリクスと基本的なバックエンド情報を送信しないようにするには、Tridentのインストール時に `--silence-autosupport`フラグを渡します。
- Tridentは、 `tridentctl send autosupport`を介して、コンテナログをオンデマンドでNetAppサポートに送信することもできます。Tridentがログをアップロードするようにトリガーする必要があります。ログを送信する前に、NetAppのhttps://www.netapp.com/company/legal/privacy-policy/["プライバシー ポリシー"^]に同意する必要があります。
- 特に指定がない限り、Trident は過去 24 時間のログを取得します。
- ログの保存期間を指定するには、 `--since`フラグを使用します。例： `tridentctl send autosupport --since=1h`。この情報は、Tridentと一緒にインストールされる `trident-autosupport`コンテナを介して収集および送信されます。コンテナイメージは "Trident AutoSupport"から入手できます。
- Trident AutoSupportは、個人を特定できる情報（PII）または個人情報を収集または送信しません。付属の "EULA"は、Tridentコンテナイメージ自体には適用されません。NetAppのデータセキュリティと信頼への取り組みの詳細については、 "ここをクリックしてください。"を参照してください。

Trident から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: null
      disableDelete: false
      serialNumbers:
        - nwkvzfanek_SN
      limitVolumeSize: ""
    state: online
    online: true
```

- AutoSupportメッセージはNetAppのAutoSupportエンドポイントに送信されます。コンテナイメージを保存するためにプライベートレジストリを使用している場合は、 `--image-registry`フラグを使用できます。

- インストール用の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、`--proxy-url`引数を`trident-autosupport`コンテナの`trident-deployment.yaml`に追加することで実行できます。

Tridentメトリクスを無効にする

メトリクスの報告を無効化するには、カスタムYAMLを生成し（`--generate-custom-yaml`フラグを使用）、それらを編集して`--metrics`フラグが`trident-main`コンテナで呼び出されないように削除してください。

Tridentのアンインストール

Tridentのアンインストールには、Tridentのインストールに使用したのと同じ方法を使用してください。

タスク概要

- アップグレード後に発生したバグ、依存関係の問題、またはアップグレードの失敗や不完全さを修正する必要がある場合は、Tridentをアンインストールし、その"version"専用の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。これは、以前のバージョンに_ダウングレード_する場合に推奨される唯一の方法です。
- アップグレードや再インストールを簡単にするため、Tridentをアンインストールしても、Tridentによって作成されたCRDまたは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、"Tridentおよび CRD を完全に削除"を参照してください。

開始する前に

Kubernetesクラスターを廃止する場合は、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のインストールに使用したのと同じ方法でアンインストールする必要があります。アンインストールする前に、最初にTrident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. `kubectl get pods -n trident`を使用してポッドを検査します。
   - オペレーターポッドがない場合、Trident は`tridentctl`を使用してインストールされました。
   - オペレーターポッドがある場合、Trident は Trident オペレーターを使用して手動または Helm を使用してインストールされました。
2. オペレーターポッドがある場合は、`kubectl describe tproc trident`を使用して、Trident が Helm を使用してインストールされたかどうかを判断します。
   - 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ました。
   - Helmラベルがない場合、TridentはTridentオペレーターを使用して手動でインストールされました。

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 Operator 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。

手動インストールをアンインストールする

Tridentをオペレータを使用してインストールした場合は、次のいずれかを実行してアンインストールできます：

1. 編集 **`TridentOrchestrator` CR** を実行してアンインストール フラグを設定します：

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

`uninstall`フラグが `true`に設定されている場合、`Trident`オペレーターは `Trident` をアンインストールしますが、`TridentOrchestrator`自体は削除しません。`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ものを作成する必要があります。

2. 削除 **`TridentOrchestrator`**（：）Trident の導入に使用された `TridentOrchestrator` CR を削除することで、オペレータに Trident のアンインストールを指示します。オペレータは `TridentOrchestrator`の削除を処理し、Trident の導入とデーモンセットを削除して、インストールの一部として作成された Trident ポッドを削除します。

```
kubectl delete -f deploy/<bundle.yaml> -n <namespace>
```

Helmのインストールをアンインストールする

Helmを使用してTridentをインストールした場合は、 `helm uninstall`を使用してアンインストールできます。

```
#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
```

`tridentctl`インストールをアンインストールします

Tridentに関連するCRDおよび関連オブジェクトを除くすべてのリソースを削除するには、 `uninstall`コマンドを `tridentctl`で使用してください：

```
./tridentctl uninstall -n <namespace>
```

Docker向けTrident

導入の前提条件

Tridentを導入する前に、ホストに必要なプロトコルの前提条件をインストールして設定する必要があります。

要件を確認する

- 導入がすべての"要件"を満たしていることを確認します。
- 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のバージョンが古い場合は、 "インストールまたはアップデートする"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールされ、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
```

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

iSCSIツール

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

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します：

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

4. マルチパスを有効にする：

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

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

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

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

6. 有効化して起動 `iscsi`：

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

Ubuntu

1. 次のシステムパッケージをインストールします：

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

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

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

3. スキャンを手動に設定します：

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

4. マルチパスを有効にする：

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

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

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

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

NVMeツール

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

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

RHEL 9

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

Ubuntu

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

FCツール

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

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

RHEL 8以降

1. 次のシステムパッケージをインストールします：

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

2. マルチパスを有効にする：

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

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

3. `multipathd`が実行されていることを確認します：

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

Ubuntu

1. 次のシステムパッケージをインストールします：

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

2. マルチパスを有効にする：

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

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

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

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

Tridentを導入

Trident for Dockerは、NetAppストレージプラットフォーム向けのDockerエコシステムとの直接統合を提供します。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、将来的に追加のプラットフォームを追加するためのフレームワークを備えています。

複数のTridentインスタンスを同じホスト上で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およびストレージタイプへの同時接続が可能になり、Dockerボリュームに使用されるストレージをカスタマイズできるようになります。

要件

"導入の前提条件"を参照してください。前提条件が満たされ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 マネージド プラグイン方式（バージョン 1.13/17.03 以降）

開始する前に

Docker 1.13/17.03より前のバージョンで従来のデーモン方式でTridentを使用したことがある場合は、マネージドプラグイン方式を使用する前に、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てください。

```
docker --version
```

バージョンが古い場合は、 "インストールまたはインストールを更新する"。

手順

1. 構成ファイルを作成し、次のように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、 `config`オプションでファイル名を指定することで任意の名前を使用できます。設定ファイルは、ホストシステムの `/etc/netappdvp`ディレクトリに配置する必要があります。
   - `log-level`：ログレベルを指定する(`debug`、 `info`、 `warn`、 `error`、 `fatal`）。デフォルトは

info。

- debug：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. 設定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. 管理されたプラグイン システムを使用して Trident を起動します。 `<version>`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。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. 構成されたシステムからストレージを消費するために Trident の使用を開始します。

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. コンテナの起動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）

開始する前に

1. Docker バージョン 1.10 以降がインストールされていることを確認してください。

```
docker --version
```

バージョンが古い場合は、インストールを更新してください。

```
curl -fsSL https://get.docker.com/ | sh
```

または、 "ディストリビューションの手順に従ってください"。

2. NFS および/または iSCSI がシステムに設定されていることを確認します。

手順

1. NetApp Docker ボリューム プラグイン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解凍します：

```
wget
https://github.com/NetApp/trident/releases/download/10.0/trident-
installer-25.10.0.tar.gz
tar zxf trident-installer-25.10.0.tar.gz
```

b. bin 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. 設定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、設定ファイルを作成したら、必要な設定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
```

> 指定されない限り、ボリューム ドライバーのデフォルト名は「netapp」です。

デーモンを起動したら、Docker CLI インターフェー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. コンテナを起動するときに Docker ボリュームをプロビジョニングします：

```
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

systemdベースのシステムのサンプルユニットファイルは、Gitリポジトリの `contrib/trident.service.example` にあります。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する必要があります。

   ```
   cp contrib/trident.service.example
   /usr/lib/systemd/system/trident.service
   ```

2. ファイルを編集し、ドライバー名と一致するように説明（2 行目）を変更し、環境を反映するように構成ファイルパス（9 行目）を変更します。
3. 変更を取り込むために systemd 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`ディレクトリ内でファイルに付けた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. ステータスを表示します。

   ```
   systemctl status trident
   ```

ユニットファイルを変更するたびに、変更を認識させるために `systemctl daemon-reload`コマンドを実行します。

Tridentのアップグレードまたはアンインストール

使用中のボリュームに影響を与えることなく、Docker 用 Trident 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中、プラグインに対する `docker volume`コマンドが成功しない短い期間があり、プラグインが再び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、これは数秒の問題です。

Upgrade

Docker用のTridentをアップグレードするには、以下の手順を実行してください。

手順

1. 既存のボリュームを一覧表示します：

```
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 を置き換えます。 `netapp/ndvp-plugin`イメージから `netapp/trident-plugin`イメージに直接アップグレードする必要があります。

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てい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以前）からTrident 20.10以降にアップグレードする場合、エラーが発生する可能性があります。詳細については、"既知の問題"を参照してください。エラーが発生した場合は、まずプラグインを無効にし、次にプラグインを削除してから、追加の設定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してください： `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all -permissions config=config.json`

アンインストール

Docker用のTridentをアンインストールするには、以下の手順を実行してください。

手順

1. プラグインによって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
```

ボリュームの操作

標準の `docker volume`コマンドを使用して、必要に応じてTridentドライバー名を指定することで、ボリュームを簡単に作成、複製、削除できます。

ボリュームの作成

- デフォルト名を使用してドライバー付きのボリュームを作成します：

```
docker volume create -d netapp --name firstVolume
```

- 特定のTridentインスタンスでボリュームを作成：

```
docker volume create -d ntap_bronze --name bronzeVolume
```

"options"を指定しない場合は、ドライバー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。ドライバーを使用して20 GiBのボリュームを作成するには、次の例を参照してください：

```
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（例：10G、20GB、3TiB）を持つ整数値を含む文字列として表されます。単位が指定されていない場合、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）で表すことができます。省略単位には2の累乗を使用します（G = GiB、T = TiB、…）。

ボリュームを削除する

- 他の Docker ボリュームと同じようにボリュームを削除します：

```
docker volume rm firstVolume
```

`solidfire-san`ドライバを使用する場合、上記の例ではボリュームが削除およびパージされます。

Docker用のTridentをアップグレードするには、以下の手順を実行してください。

ボリュームをクローニングする

`ontap-nas`、 `ontap-san`、および `solidfire-san`ストレージドライバを使用する場合、Tridentはボリュームをクローニングできます。 `ontap-nas-flexgroup`または `ontap-nas-economy`ドライバを使用する場合、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、新しいスナップショットが作成されます。

- ボリュームを検査してスナップショットを列挙します：

```
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。これにより、新しいスナップショットが作成されます：

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume>
```

- ボリューム上の既存のスナップショットから新しいボリュームを作成します。これにより、新しいスナップショットは作成されません：

```
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する

コンテナは、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に*のみ*、Tridentを使用して外部で作成されたブロックデバイス（またはそのクローン）にアクセスできます（例： `ext4`でフォーマットされた `/dev/sdc1`は、Trident経由ではアクセスできません）。

ドライバー固有のボリュームオプション

各ストレージドライバには異なるオプションセットがあり、ボリュームの作成時に指定して結果をカスタマイズできます。構成された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。

ボリューム作成操作中にこれらのオプションを使用するのは簡単です。CLI 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を上書きします。

ONTAPボリュームオプション

NFS、iSCSI、FC 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	概要
`size`	ボリュームのサイズ。デフォルトは 1 GiB です。
`spaceReserve`	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。デフォルトはシンプロビジョニングです。有効な値は `none`（シンプロビジョニング）および `volume`（シックプロビジョニング）です。
`snapshotPolicy`	これにより、スナップショットポリシーが目的の値に設定されます。デフォルトは `none`で、ボリュームのスナップショットは自動的に作成されません。ストレージ管理者によって変更されない限り、「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つの時間別スナップショット、2つの日次スナップショット、2つの週次スナップショットを作成して保持します。スナップショットに保存されたデータは、ボリューム内の任意のディレクトリにある `.snapshot`ディレクトリを参照することで復元できます。
`snapshotReserve`	これにより、スナップショットの予約が希望のパーセンテージに設定されます。デフォルトでは値なし、つまり ONTAP が snapshotReserve（通常 5%）を選択します（snapshotPolicy を選択した場合）。snapshotPolicy が none の場合は 0% です。すべての ONTAP バックエンドの設定ファイルでデフォルトの snapshotReserve 値を設定でき、ontap-nas-economy を除くすべての ONTAP バックエンドのボリューム作成オプションとして使用できます。

オプション	概要
`splitOnClone`	ボリュームをクローンする場合、これによりONTAPはクローンを親からすぐに分離します。デフォルトは `false`です。ボリュームのクローン作成の一部のユースケースでは、ストレージ効率を高める機会がほとんどないため、作成後すぐにクローンを親から分割するのが最適です。たとえば、空のデータベースのクローンを作成すると、時間は大幅に節約できますが、ストレージの節約はほとんどないため、クローンをすぐに分割するのが最適です。
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVE のライセンスを取得し、クラスターで有効にする必要があります。 NAEがバックエンドで有効になっている場合、Tridentでプロビジョニングされたボリュームは、NAEが有効になります。 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 の連携"
`tieringPolicy`	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、データが非アクティブ（コールド）になったときにクラウド層に移動するか否かが決定されます。

次の追加オプションは NFS *のみ*に有効です。

オプション	概要
`unixPermissions`	これは、ボリューム自体の権限セットを制御します。デフォルトでは、権限は ``---rwxr-xr-x`に設定されます。または数値表記では0755となり、 `root`が所有者になります。テキスト形式または数値形式のいずれかが機能します。
`snapshotDir`	これを `true`に設定すると、ボリュームにアクセスするクライアントに `.snapshot`ディレクトリが表示されるようになります。デフォルト値は `false`で、`.snapshot`ディレクトリの表示はデフォルトで無効になっています。公式のMySQLイメージなど一部のイメージは、 `.snapshot`ディレクトリが表示されていると期待どおりに機能しません。
`exportPolicy`	ボリュームに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 `default`。

オプション	概要
`securityStyle`	ボリュームへのアクセスに使用するセキュリティ スタイルを設定します。デフォルトは `unix`です。有効な値は `unix`と `mixed`です。

次の追加オプションは iSCSI *のみ*に適用されます。

オプション	概要
`fileSystemType`	iSCSI ボリュームをフォーマットするため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`、および `xfs`です。
`spaceAllocation`	これを `false`に設定すると、LUNのスペース割り当て機能がオフになります。デフォルト値は `true`で、ボリュームの容量が不足し、ボリューム内のLUNが書き込みを受け付けることができなくなった場合にONTAPがホストに通知することを意味します。このオプションでは、ホストがデータを削除したときにONTAPがスペースを自動的に再利用することもできます。

例

以下の例を参照してください：

- 10 GiB のボリュームを作成します：

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- スナップショット付きの 100 GiB 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。

スナップショットリザーブが指定されておらず、スナップショットポリシーが `none`の場合、Tridentはスナップショットリザーブとして0%を使用します。

- スナップショットポリシーとスナップショットリザーブのないボリュームを作成します：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- スナップショットポリシーがなく、カスタムスナップショット予約が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
```

- スナップショットポリシーと10%のカスタムスナップショット予約を持つ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
```

- スナップショットポリシーでボリュームを作成し、ONTAPのデフォルトのスナップショット予約量（通常は5%）を使用します：

```
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 ソフトウェアオプションは、ボリュームに関連付けられたサイズとサービス品質（QoS）ポリシーを公開します。ボリュームが作成されると、それに関連付けられた QoS ポリシーは `-o type=service_level` の命名法を使用して指定されます。

Element ドライバーを使用して 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、およびバースト IOPS を指定することです。

その他の Element ソフトウェア 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	概要
`size`	ボリュームのサイズ。デフォルトは 1 GiB または config エントリです…"defaults"：{"size": "5G"}。
`blocksize`	512または4096を使用します。デフォルトは512または設定エントリ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
      }
    }
  ]
}
```

上記の構成には、Bronze、Silver、Gold の 3 つのポリシー定義があります。これらの名前は任意です。

- 10 GiB 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
```

- 100 GiB の 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
```

ログを収集する

トラブルシューティングに役立つログを収集できます。ログを収集するために使用する方法は、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のためにログを収集する

手順

1. 推奨される管理プラグイン方式を使用してTridentを実行している場合（つまり、 `docker plugin`コマンドを使用）、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
```

標準のログレベルでは、ほとんどの問題を診断できます。それだけでは不十分な場合は、デバッグログを有効にすることができます。

2. デバッグ ログを有効にするには、デバッグ ログを有効にした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は、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. バイナリ自体をホスト上で実行している場合、ログはホストの `/var/log/netappdvp`ディレクトリで利用できます。デバッグログを有効にするには、プラグインを実行するときに `-debug`を指定します。

一般的なトラブルシューティングのヒント

- 新しいユーザーが遭遇する最も一般的な問題は、プラグインの初期化を妨げる誤った構成です。このような場合、プラグインをインストールまたは有効化しようとすると、次のようなメッセージが表示される可能性があります：

  ```
  Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
  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。幸いなことに、このプラグインには包括的なログ機能が組み込まれているため、発生する可能性のあるほとんどの問題の診断に役立ちます。

- PVをコンテナにマウントする際に問題がある場合は、 `rpcbind`がインストールされ、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ーを使用して、 `rpcbind`が実行されているかどうかを確認してください。rpcbindサービスのステータスを確認するには、 `systemctl status rpcbind`またはそれに相当するコマンドを実行します。

複数のTridentインスタンスを管理

複数のストレージ構成を同時に利用したい場合は、複数の Trident インスタンスが必要です。複数のインスタンスを作成するための鍵は、コンテナ化されたプラグインでは `--alias`オプションを使用して、またはホスト上で Trident をインスタンス化する際には `--volume-driver`オプションを使用して、それぞれに異なる名前を付けることです。

Docker マネージドプラグインの手順（バージョン 1.13/17.03 以降）

1. エイリアスと設定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. エイリアスをドライバー名として指定してボリュームを作成します。

   たとえば、金の数量の場合：

   ```
   docker volume create -d gold --name ntapGold
   ```

   たとえば、シルバーのボリュームの場合：

```
docker volume create -d silver --name ntapSilver
```

従来の手順（バージョン**1.12**以前）

1. カスタム ドライバー ID を使用して NFS 構成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. カスタムドライバー 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. 各ドライバインスタンスの 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。

グローバル設定オプション

これらの設定オプションは、使用されてい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	概要	例
`version`	設定ファイルのバージョン番号	`1`

オプション	概要	例
`storageDriverName`	ストレージドライバの名前	`ontap-nas`、 `ontap-san`、`ontap-nas-economy`、`ontap-nas-flexgroup`、`solidfire-san`
`storagePrefix`	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_`
`limitVolumeSize`	ボリュームサイズに対するオプションの制限。デフォルト：""（強制されません）	`10g`

`storagePrefix`（デフォルトを含む）は Element バックエンドには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、プレフィックスを使用しません。NetApp では、Docker ボリュームマッピングに特定の tenantID を使用するか、名前の変更が使用された可能性がある場合は Docker バージョン、ドライバ情報、および Docker からの元の名前が入力された属性データを使用することを推奨します。

作成するボリュームごとにオプションを指定する必要がないように、デフォルトのオプションが用意されています。 `size`オプションはすべてのコントローラータイプで使用できます。デフォルトのボリュームサイズを設定する方法の例については、ONTAP構成セクションを参照してください。

オプション	概要	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

上記のグローバル設定値に加えて、ONTAPを使用する場合は、次の最上位オプションが利用可能です。

オプション	概要	例
`managementLIF`	ONTAP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。	`10.0.0.1`

オプション	概要	例
dataLIF	プロトコル LIF の IP アドレス。 **ONTAP NAS** ドライバー：NetApp では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 dataLIF を取得します。NFS マウント操作に使用する完全修飾ドメイン名（FQDN）を指定できます。これにより、複数の dataLIF 間で負荷を分散するラウンドロビン DNS を作成できます。 **ONTAP SAN** ドライバー：iSCSI または FC の場合は指定しないでください。Trident は"ONTAP セレクティブLUNマップ"を使用して、マルチパス セッションを確立するために必要な iSCSI または FC LIF を検出します。 `dataLIF`が明示的に定義されている場合、警告が生成されます。	10.0.0.2
svm	使用するStorage Virtual Machine（管理LIFがクラスタLIFの場合は必須）	svm_nfs
username	ストレージデバイスに接続するためのユーザ名	vsadmin
password	ストレージデバイスに接続するためのパスワード	secret
aggregate	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVMに割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバの場合、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は、FlexGroupボリュームのプロビジョニングに使用されます。	aggr1
limitAggregateUsage	オプション：使用率がこのパーセンテージを超える場合はプロビジョニングを失敗します	75%

オプション	概要	例
nfsMountOptions	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-economy`ドライバーでのみ使用可能* "NFSホストの構成情報についてはここを参照してください"。	-o nfsvers=4
igroupName	Tridentはノードごとに `igroups`を `netappdvp`として作成および管理します。 この値は変更または省略できません。 • `ontap-san`ドライバでのみ使用可能*。	netappdvp
limitVolumeSize	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]の範囲内である必要があり、デフォルトは200です。 • `ontap-nas-economy`ドライバの場合、このオプションを使用すると、FlexVolあたりのqtreeの最大数をカスタマイズできます*。	300
sanType	* ontap-san`ドライバーのみでサポートされます。*iSCSIの場合は `iscsi、NVMe/TCPの場合は nvme、またはSCSI over Fibre Channel（FC）の場合は `fcp`を選択するために使用します。	iscsi 空白の場合
limitVolumePoolSize	* `ontap-san-economy`および `ontap-san-economy`ドライバーでのみサポートされます。*ONTAPのontap-nas-economyおよびontap-san-economyドライバーでFlexVolのサイズを制限します。	300g

作成するボリュームごとに指定する必要がないように、デフォルトのオプションが用意されています。

オプション	概要	例
spaceReserve	スペース予約モード： none（シンプロビジョニング）または volume（シック）	none
snapshotPolicy	使用するSnapshotポリシー。デフォルトは none	none
snapshotReserve	Snapshotリザーブの割合。デフォルトは「」でONTAPのデフォルトを受け入れます	10
splitOnClone	クローン作成時に親からクローンを分割します。デフォルトは false	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、NVEのライセンスを取得し、クラスタで有効にする必要があります。 NAEがバックエンドで有効になっている場合、Tridentでプロビジョニングされたボリュームは、NAEが有効になります。 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 の連携"	true
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 777	777
snapshotDir	`.snapshot`ディレクトリへのアクセスのためのNASオプション。	NFSv4の場合は"true"、NFSv3の場合は"false"
exportPolicy	使用するNFSエクスポートポリシーのNASオプション。デフォルトは `default`です	default
securityStyle	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は `mixed`および `unix`セキュリティスタイルをサポートします。デフォルトは `unix`です。	unix
fileSystemType	SANオプションでファイルシステムのタイプを選択します。デフォルトは `ext4`です	xfs
tieringPolicy	使用する階層化ポリシー。デフォルトは `none`です。	none

オプション	概要	例
skipRecovery Queue	ボリュームの削除中は、ストレージ内のリカバリキューをバイパスし、ボリュームを直ちに削除します。	``

スケーリングオプション

`ontap-nas`および `ontap-san`ドライバーは、Docker ボリュームごとに ONTAP FlexVol を作成します。ONTAP は、クラスタノードあたり最大 1000 個の FlexVolsをサポートし、クラスタ全体では最大 12,000 個の FlexVolボリュームをサポートします。Docker ボリュームの要件がこの制限内に収まる場合、`ontap-nas`ドライバーは、Docker ボリューム単位の Snapshot やクローン作成など、FlexVolsが提供する追加機能により、NAS ソリューションとして推奨されます。

FlexVolの制限で収容できる数よりも多くのDockerボリュームが必要な場合は、`ontap-nas-economy`または`ontap-san-economy`ドライバを選択してください。

`ontap-nas-economy`ドライバは、自動管理されるFlexVolボリュームのプール内で、DockerボリュームをONTAP Qtreeとして作成します。Qtreeは、一部の機能を犠牲にして、クラスタノードあたり最大100,000、クラスタあたり最大2,400,000という、はるかに優れたスケーリングを提供します。 `ontap-nas-economy`ドライバは、Dockerボリューム単位のSnapshotやクローニングをサポートしていません。

`ontap-nas-economy`ドライバは、Docker Swarmが複数のノードにわたるボリューム作成を調整しないため、現在Docker Swarmではサポートされていません。

`ontap-san-economy`ドライバは、自動管理されるFlexVol ボリュームの共有プール内で、DockerボリュームをONTAP LUNとして作成します。これにより、各FlexVolは1つのLUNに制限されず、SANワークロードに対してより優れたスケーラビリティを提供します。ストレージアレイによっては、ONTAPはクラスタあたり最大16384個のLUNをサポートします。ボリュームは内部的にLUNであるため、このドライバはDockerボリューム単位のスナップショットとクローニングをサポートします。

`ontap-nas-flexgroup`ドライバーを選択して、1つのボリュームへの並列性を高め、数十億のファイルを持つペタバイト規模まで拡張できます。FlexGroupsの理想的なユースケースには、AI／ML／DL、ビッグデータと分析、ソフトウェアビルド、ストリーミング、ファイルリポジトリなどがあります。Tridentは、FlexGroupボリュームをプロビジョニングする際にSVMに割り当てられたすべてのアグリゲートを使用します。TridentでのFlexGroupサポートには、以下の考慮事項もあります：

- ONTAP バージョン 9.2 以降が必要です。
- 本稿執筆時点では、FlexGroupsはNFS v3のみをサポートします。
- SVM に対して 64 ビット NFSv3 識別子を有効にすることを推奨します。
- 推奨される最小のFlexGroupメンバー/ボリュームサイズは100GiBです。
- クローニングはFlexGroupボリュームではサポートされていません。

FlexGroupsおよびFlexGroupsに適したワークロードの詳細については、 "NetApp FlexGroupボリュームのベストプラクティスと実装ガイド"を参照してください。

同じ環境で高度な機能と大規模なスケールを実現するには、Dockerボリュームプラグインの複数のインスタンスを実行することができます。1つは `ontap-nas`を使用し、もう1つは `ontap-nas-economy`を使用します。

Tridentのカスタム**ONTAP** ロール

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

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

ONTAPコマンドラインの使用

1. 次のコマンドを使用して新しいロールを作成します：

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

2. Tridentユーザーのユーザー名を作成します：

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

3. ロールをユーザーにマップします：

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

System Managerを使用

ONTAP System Managerで次の手順を実行します。

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

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

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

詳細については、次のページを参照してください：

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

ONTAP設定ファイルの例

<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
}
```

<code>ontap-san</code> ドライバの **SCSI over FC** の例

```
{
  "version": 1,
  "backendName": "ontap-san-backend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "sanType": "fcp",
  "svm": "trident_svm",
  "username": "vsadmin",
  "password": "password",
  "useREST": true
}
```

Element ソフトウェア構成

グローバル設定値に加えて、Element ソフトウェア（NetApp HCI/SolidFire）を使用する場合、これらのオプションが利用可能です。

オプション	概要	例
`Endpoint`	https://<login>：<password>@<mvip>/json-rpc/<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IPアドレスとポート	10.0.0.7:3260

オプション	概要	例
`TenantName`	使用する SolidFire テナント（見つからない場合は作成）	`docker`
`InitiatorIFace`	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にインターフェイスを指定する	`default`
`Types`	QoS仕様	下記の例をご覧ください
`LegacyNamePrefix`	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する場合は、ボリューム名方式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-`

`solidfire-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
      }
    }
  ]
}
```

既知の問題と制限事項

Docker で Trident を使用する際の既知の問題と制限に関する情報を確認してください。

Trident Docker Volume Plugin を古いバージョンから **20.10** 以降にアップグレードすると、「そのようなファイルまたはディレクトリはありません」というエラーが発生し、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 文字以上である必要があります。

これは Docker クライアントの制限です。クライアントは、1 文字の名前を Windows パスとして解釈します "バグ25773を参照"。

Docker Swarmには、Tridentがあらゆるストレージとドライバの組み合わせでサポートすることを妨げる特定の動作があります。

- Docker Swarm は現在、一意のボリューム識別子としてボリューム ID ではなくボリューム名を使用しています。
- ボリュームリクエストは、Swarmクラスター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独立して実行する必要があります。ONTAP の仕組みと `ontap-nas`および `ontap-san`ドライバの機能により、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。

残りのドライバーは、競合状態などの問題の影響を受けやすく、その結果、明確な「勝者」なしに単一のリクエストに対して大量のボリュームが作成されることがあります。たとえば、Elementには、ボリュームに同じ名前を付けてIDを異ならせることができる機能があります。

NetAppはDockerチームにフィードバックを提供しましたが、今後の対応については何も示されていません。

FlexGroupがプロビジョニングされている場合、2番目のFlexGroupがプロビジョニングされているFlexGroupと1つ以上のアグリゲートを共有している場合、ONTAPは2番目の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を導入する際は、ここに記載されている推奨事項を使用してください。

専用の名前空間にデプロイする

"ネームスペース"異なるアプリケーション間の管理上の分離を提供し、リソース共有の障壁となります。たとえば、ある名前空間のPVCを別の名前空間で使用することはできません。TridentはKubernetesクラスター内のすべての名前空間にPVリソースを提供し、その結果、昇格された権限を持つサービスアカウントを活用します。

さらに、Tridentポッドへのアクセスにより、ユーザーがストレージシステムの資格情報やその他の機密情報にアクセスできるようになる可能性があります。アプリケーションユーザー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

Kubernetesには2つの機能があり、これら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"により、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消費制限をネームスペースごとに実装できます。さらに、 "範囲制限"を使用することで、リクエストがプロビジョナーに転送される前に、PVCリクエストが最小値と最大値の範囲内であることを確認します。

これらの値は名前空間ごとに定義されます。つまり、各名前空間には、そのリソース要件に適合する値が定義されている必要があります。詳細については、こちらをご覧ください "割り当てを活用する方法"。

ストレージ構成

NetApp ポートフォリオの各ストレージプラットフォームには、コンテナ化されているかどうかに関係なく、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およびElementと連携します。すべてのアプリケーションやシナリオに他のプラットフォームよりも適したプラットフォームは存在しませんが、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。

利用しているプロトコルに応じて、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。オプションとして、利用可能な場合は、バックエンド、ストレージ クラス、および PVC 設定にアプリケーションのベスト プラクティスを組み込んで、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を検討することもできます。

ONTAPおよびCloud Volumes ONTAPのベストプラクティス

Trident 向けに ONTAP および Cloud Volumes ONTAP を設定する際のベストプラクティスを学習します。

以下の推奨事項は、Tridentによって動的にプロビジョニングされるボリュームを消費するコンテナ化されたワークロード向けにONTAPを設定するためのガイドラインです。それぞれの推奨事項について、お客様の環境における適切性を考慮して評価する必要があります。

Trident専用の**SVM**を使用する

Storage Virtual Machine（SVM）は、ONTAPシステム上のテナント間の分離と管理の分離を実現します。SVMをアプリケーション専用にすることで、権限の委任が可能になり、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るようになります。

SVM の管理にはいくつかのオプションがあります：

- バックエンド構成でクラスタ管理インターフェイスを指定し、適切なクレデンシャルとともに 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。
- 管理ロールを NFS データインターフェイスと共有します。

いずれの場合も、インターフェイスはDNSに登録され、Tridentの設定時に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の保持を使用しないSVM-DRなど、一部のDRシナリオが容易になります。

SVM に専用の管理 LIF を使用するか、共有の管理 LIF を使用するかという優先順位はありませんが、ネットワーク セキュリティ ポリシーが選択したアプローチと一致していることを確認する必要があります。いずれにせよ、管理 LIF は DNS 経由でアクセスできるようにして、 "SVM-DR"Trident と組み合わせて使用する場合に最大限の柔軟性を実現する必要があります。

最大ボリューム数を制限する

ONTAP ストレージシステムには最大ボリューム数があり、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。特定のプラットフォームと ONTAP バージョンの正確な制限を確認するには、 "NetApp Hardware Universe"を参照してください。ボリューム数が上限に達すると、Trident 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操作が失敗します。

Tridentの `ontap-nas`および `ontap-san`ドライバーは、作成される各Kubernetes永続ボリューム（PV）に対してFlexVolumeをプロビジョニングします。 `ontap-nas-economy`ドライバーは、200PVごとに約1つのFlexVolumeを作成します（50～300の間で設定可能）。 `ontap-san-economy`ドライバーは、100PVごとに約1つのFlexVolumeを作成します（50～200の間で設定可能）。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次のように実行できます：

```
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基準によって異なります：

- ONTAP クラスタ内の既存のボリューム数
- Trident 以外のアプリケーション用にプロビジョニングする予定のボリュームの数
- Kubernetes アプリケーションで使用される永続的ボリュームの数

この `max-volumes`値は、ONTAP クラスタ内のすべてのノードにプロビジョニングされたボリュームの合計

であり、個々の ONTAP ノードのものではありません。その結果、ONTAP クラスタノードによっては、Trident でプロビジョニングされたボリュームが他のノードよりもはるかに多い場合や少ない場合があります。

たとえば、2ノードのONTAPクラスタには、最大2000個のFlexVolボリュームをホストする機能があります。最大ボリューム数を1250に設定することは非常に妥当に思えます。ただし、1つのノードからの "アグリゲート"のみがSVMに割り当てられている場合、または1つの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（容量不足など）、もう一方のノードがすべてのTridentプロビジョニングボリュームのターゲットになります。つまり、 `max-volumes`の値に達する前に、そのノードのボリューム制限に達する可能性があり、その結果、Tridentとそのノードを使用する他のボリューム操作の両方に影響を与えます。この状況を回避するには、クラスタ内の各ノードからのアグリゲートが、**Trident**で使用される**SVM**に同数割り当てられるようにします。

ボリュームをクローニングする

NetApp Tridentは、 `ontap-nas`、 `ontap-san`、および `solidfire-san`ストレージドライバーを使用する場合、ボリュームのクローン作成をサポートします。 `ontap-nas-flexgroup`または `ontap-nas-economy`ドライバーを使用する場合、クローン作成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、新しいスナップショットが作成されます。

異なるStorageClassに関連付けられたPVCの複製は避けてください。互換性を確保し、予期しない動作を防ぐために、同じStorageClass内でクローン操作を実行してください。

Tridentで作成されるボリュームの最大サイズを制限する

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、 `limitVolumeSize`パラメータを `backend.json`定義で使用します。

ストレージ アレイでのボリューム サイズの制御に加えて、Kubernetes の機能も活用する必要があります。

Tridentによって作成される**FlexVols**の最大サイズを制限する

ontap-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FlexVolsの最大サイズを設定するには、 `limitVolumePoolSize`パラメータを `backend.json`定義で使用してください。

双方向**CHAP**を使用するように**Trident**を設定

バックエンド定義で CHAP イニシエーターとターゲットのユーザー名とパスワードを指定し、Trident で SVM の CHAP を有効にすることができます。バックエンド構成で `useCHAP`パラメータを使用すると、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する

SVMに適用されるONTAP QoSポリシーを活用することで、Tridentでプロビジョニングされたボリュームが消費できるIOPSの数を制限します。これにより、 "いじめを防ぐ"または制御不能なコンテナがTrident SVM外部のワークロードに影響を与えるのを防ぎます。

SVM の QoS ポリシーは数ステップで作成できます。最も正確な情報については、お使いのバージョンの ONTAP のドキュメントを参照してください。以下の例では、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>
```

さらに、ONTAP のバージョンでサポートされている場合は、コンテナ化されたワークロードへのスループット量を保証するために、QoS 最小値の使用を検討できます。アダプティブ QoS は SVM レベルのポリシーと互換性がありません。

コンテナ化されたワークロード専用の IOPS の数は、さまざまな側面によって異なります。とりわけ、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 デプロイメントに関連しない他のワークロードがストレージ リソースを利用している場合は、それらのワークロードが誤って悪影響を受けないように注意する必要があります。
- コンテナ内で実行されると予想されるワークロード。高い IOPS 要件を持つワークロードがコンテナ内で実行される場合、低い QoS ポリシーではエクスペリエンスが悪くなります。

SVM レベルで割り当てられた QoS ポリシーにより、SVM にプロビジョニングされたすべてのボリュームが同じ IOPS プールを共有することになるということを覚えておくことが重要です。コンテナ化されたアプリケーションの 1 つまたは少数に高い IOPS 要件がある場合、他のコンテナ化されたワークロードに対して脅威となる可能性があります。このような場合は、外部自動化を使用してボリュームごとの QoS ポリシーを割り当てることを検討してください。

QoSポリシーグループをSVMに割り当てる必要があるのは、ONTAPバージョンが9.8より前の場合*のみ*です。

Trident用の**QoS**ポリシーグループを作成する

サービス品質（QoS）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ことを保証します。ONTAP QoS ポリシーグループはボリュームの QoS オプションを提供し、ユーザーが 1 つ以上のワークロードのスループット上限を定義できるようにします。QoS の詳細については、 "QoSによるスループットの保証"を参照してください。バックエンドまたはストレージプールで QoS ポリシーグループを指定すると、そのプールまたはバックエンドで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、IOPS で一定した最大（または後のバージョンでは最小）スループットを提供します。アダプティブ QoS は、ワークロードのサイズの変化に応じて IOPS と TB|GB の比率を維持しながら、スループットをワークロードのサイズに合わせて自動的にスケーリングします。これは、大規模な展開で数百または数千のワークロードを管理する場合に大きな利点となります。

QoS ポリシー グループを作成するときは、次の点を考慮してください：

- `qosPolicy`キーは、バックエンド構成の `defaults`ブロックに設定する必要があります。次のバックエンド構成の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
```

- 各ボリュームがポリシー グループで指定されたスループット全体を取得できるように、ボリュームごとにポリシー グループを適用する必要があります。共有ポリシー グループはサポートされていません。

QoSポリシーグループの詳細については、 "ONTAPコマンド リファレンス"を参照してください。

Kubernetes クラスタメンバーへのストレージリソースアクセスを制限する

Tridentによって作成されたNFSボリューム、iSCSI LUN、FC LUNへのアクセスを制限することは、Kubernetesデプロイメントのセキュリティ体制にとって重要なコンポーネントです。こうすることで、Kubernetesクラスターの一部ではないホストがボリュームにアクセスして予期せずデータを変更する可能性を防ぐことができます。

名前空間は Kubernetes 内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。同じ名前空間内のリソースは共有できると想定されていますが、重要なのは、名前空間間の機能がないことです。つまり、PV 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、同じ名前空間にあるポッドからのみアクセスできます。適切な場合には、名前空間を使用して分離を行うことが重要です。

Kubernetes コンテキストでのデータセキュリティに関してほとんどの組織が主に懸念するのは、コンテナ内のプロセスが、ホストにマウントされているがコンテナ用ではないストレージにアクセスできることです。 "ネームスペース"は、この種の侵害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。

特権コンテナとは、通常よりも大幅に高いホストレベルの権限で実行されるコンテナです。これらはデフォルトでは拒否されないため、 "Pod セキュリティポリシー"を使用してこの機能を無効にしてください。

Kubernetes と外部ホストの両方からアクセスする必要があるボリュームの場合、ストレージは従来の方法で管理する必要があります。PV は管理者によって導入され、Trident では管理されません。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なくなった場合にのみ、ストレージ ボリュームが破棄されるようになります。さらに、カスタム エクスポート ポリシーを適用して、Kubernetes ク

ラスター ノードおよび Kubernetes クラスター外部の対象サーバーからのアクセスを有効にすることもできます。

専用のインフラストラクチャノードを持つデプロイメント（例：OpenShift）またはユーザーアプリケーションをスケジュールできないその他のノードの場合は、別のエクスポートポリシーを使用して、ストレージリソースへのアクセスをさらに制限する必要があります。これには、これらのインフラストラクチャノードに導入されるサービス（例：OpenShift MetricsおよびLoggingサービス）、およびインフラストラクチャ以外のノードに導入される標準アプリケーションのエクスポートポリシー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する

各バックエンドに対して、Kubernetes クラスター内に存在する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および管理できます。これにより、Trident は Kubernetes クラスター内のノードにプロビジョニングされたボリュームへのアクセスを制限し、ノードの追加/削除を簡素化します。

あるいは、エクスポート ポリシーを手動で作成し、各ノード 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。
- エクスポートポリシーにルールを追加するには、 `vserver export-policy rule create` ONTAP CLI コマンドを使用します。

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。

アプリケーション **SVM** の **`showmount`**を無効にします

`showmount`機能により、NFSクライアントはSVMに対して利用可能なNFSエクスポートのリストを照会できます。Kubernetesクラスタにデプロイされたポッドは、 `showmount -e`コマンドを実行し、アクセスできないマウントも含め、使用可能なマウントのリストを受け取ることができます。これ自体はセキュリティ侵害ではありませんが、権限のないユーザーがNFSエクスポートに接続する際に役立つ可能性のある不要な情報を提供することになります。

`showmount`を無効にするには、SVMレベルのONTAP CLIコマンドを使用します：

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire ベストプラクティス

Trident 用の SolidFire ストレージを設定する際のベストプラクティスを学びます。

SolidFireアカウントを作成する

各SolidFireアカウントは一意のボリューム所有者を表し、独自のチャレンジハンドシェイク認証プロトコル（CHAP）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、アカウント名と関連するCHAPクレデンシャルを使用するか、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。1つのアカウントには最大2,000個のボリュームを割り当てることができますが、1つのボリュームは1つのアカウントにのみ属することができます。

QoS ポリシーを作成する

SolidFire のサービス品質（QoS）ポリシーは、多くのボリュームに適用できる標準化されたサービス品質設定を作成して保存する場合に使用します。

ボリュームごとに QoS パラメータを設定できます。QoS を定義する 3 つの構成可能なパラメータ（最小 IOPS、最大 IOPS、バースト IOPS）を設定することで、各ボリュームのパフォーマンスを保証できます。

4KB ブロック サイズで可能な最小、最大、およびバースト IOPS 値は次のとおりです。

IOPSパラメータ	定義	最小値	デフォルト値	最大値（4Kb）
最小 IOPS	ボリュームの保証されたパフォーマンス レベル。	50	50	15000
最大 IOPS	パフォーマンスはこの制限を超えることはありません。	50	15000	200,000
バースト IOPS	短いバーストシナリオで許可される最大 IOPS。	50	15000	200,000

> 最大 IOPS とバースト IOPS は最大 200,000 まで設定できますが、ボリュームの実際の最大パフォーマンスは、クラスターの使用状況とノードごとのパフォーマンス レベルによって制限されます。

ブロック サイズと帯域幅は IOPS の数に直接影響します。ブロック サイズが大きくなるにつれて、システムはより大きなブロック サイズを処理するために必要なレベルまで帯域幅を増加させます。帯域幅が増加すると、システムが達成できる IOPS の数は減少します。QoS とパフォーマンスの詳細については、 "SolidFire のサービス品質"を参照してください。

SolidFire 認証

Element は、CHAP とボリューム アクセス グループ（VAG）の 2 つの認証方法をサポートしています。CHAP は、CHAP プロトコルを使用して、バックエンドに対してホストを認証します。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。NetApp は、認証には、よりシンプルでスケーリングの制限がない CHAP を使用することをお勧めします。

> 拡張 CSI プロビジョナーを使用する Trident では、 CHAP 認証の使用がサポートされます。VAG は、従来の非 CSI 動作モードでのみ使用してください。

CHAP 認証（イニシエーターが意図したボリューム ユーザーであることの検証）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 を使用する場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションが利用できます。単方向 CHAP は、SolidFire アカウント名とイニシエーター 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 CHAP オプションは、ボリュームがアカウント名とイニ

シエーター シークレットを使用してホストを認証し、次に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、ボリュームを認証する最も安全な方法を提供します。

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 イニシエーター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 認証の有無にかかわらず、グループ内の各ボリュームにアクセスできます。iSCSI イニシエーターが CHAP 認証を使用するように構成されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエーターが CHAP 認証を使用するように構成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。

さらに詳しい情報はどこで見つかりますか？

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。 "NetApp ライブラリ"で最新バージョンを検索してください。

ONTAP

- "NFS のベストプラクティスと実装ガイド"
- "SAN の管理"（iSCSI の場合）
- "RHEL の iSCSI Express 構成"

Element ソフトウェア

- "Linux用SolidFireの設定"

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上のMySQLのベストプラクティス"
- "SolidFire での MySQL のベストプラクティス"
- "NetApp SolidFire と 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"

すべてのアプリケーションに特定のガイドラインがあるわけではないため、NetApp チームと協力し、"NetApp ライブラリ"を使用して最新のドキュメントを見つけることが重要です。

Tridentを統合

Tridentを統合するには、次のような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：ドライバーの選択と展開、ストレージクラスの設計、仮想プールの設計、永続ボリュームクレーム（PVC）がストレージ プロビジョニングに与える影響、ボリューム操作、およびTridentを使用したOpenShiftサービスの展開。

ドライバの選択と導入

ストレージ システムのバックエンド 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、使用す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区別されます。そのため、導入するドライバを決定する際は慎重に検討してください。

より高いレベルでは、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 ベースのドライバーがデフォルトの選択肢になりますが、ブロックベースの iSCSI ドライバー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、ストレージおよびインフラストラクチャ チームの習熟度に基づいてプロトコルを選択します。一般的に言えば、ほとんどのアプリケーションではそれらの間にほとんど違いがないため、共有ストレージ（複数のポッドが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決定することがよくあります。

利用可能な ONTAP バックエンドドライバーは次のとおりです：

- `ontap-nas`：プロビジョニングされた各PVは完全なONTAP FlexVolumeです。
- `ontap-nas-economy`：プロビジョニングされた各PVはqtreeで、FlexVolumeあたりのqtree数は設定可能です（デフォルトは200）。
- `ontap-nas-flexgroup`：各PVは完全なONTAP FlexGroupとしてプロビジョニングされ、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、それ自身のFlexVolume内のLUNです。
- `ontap-san-economy`：プロビジョニングされた各PVはLUNであり、FlexVolumeあたりのLUN数は設定可能です（デフォルトは100）。

3 つの NAS ドライバーのいずれかを選択すると、アプリケーションで利用できる機能にいくつかの影響が生じます。

以下の表では、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、その機能が必要な場合、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。上付き脚注は、機能およびドライバーごとに機能を区別します。

ONTAP NAS ドライバー	**Snapshot** 数	クローン	動的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	はい [5]	はい	はい [1]	はい	はい [1]
`ontap-nas-economy`	NOfootnote：3 []	NOfootnote：3 []	はい [5]	はい	NOfootnote：3 []	はい	NOfootnote：3 []
`ontap-nas-flexgroup`	はい [1]	いいえ	はい [5]	はい	はい [1]	はい	はい [1]

Tridentは、ONTAP用に2つのSANドライバを提供しており、その機能は以下のとおりです。

ONTAP SANドライバー	**Snapshot** 数	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	はい [4]	はい	はい [1]	はい	はい [1]

ONTAP SANドライバー	Snapshot数	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-economy	はい	はい	はい [4]	はい	NOfootnote：3 []	はい	NOfootnote：3 []

> 上記の表の脚注：Yes [1]：Tridentで管理されていません Yes [2]：Tridentで管理されていますが、PV単位ではありません NO [3]：Tridentで管理されておらず、PV単位でもありません Yes [4]：rawブロックボリュームでサポートされています Yes [5]：Tridentでサポートされています

PV粒度ではない機能は、FlexVolume全体に適用され、すべてのPV（つまり、共有FlexVols内のqtreeまたはLUN）は共通のスケジュールを共有します。

上の表からわかるように、 ontap-nas`と `ontap-nas-economy`の機能の多くは同じです。ただし、`ontap-nas-economy`ドライバーはPV単位の粒度でスケジュールを制御する機能を制限するため、特にディザスタリカバリとバックアップ計画に影響を与える可能性があります。ONTAPストレージでPVCクローン機能を活用したい開発チームにとって、これは `ontap-nas、 `ontap-san`または `ontap-san-economy`ドライバーを使用する場合にのみ可能です。

> `solidfire-san`ドライバは PVC のクローニングも可能です。

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のストレージなど、さまざまなユースケースに対応するデータ管理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と互換性のあるドライバーは ontap-nas、ontap-nas-economy、 ontap-san、 `ontap-san-economy`です。これらはCloud Volume ONTAP for AzureおよびCloud Volume ONTAP for GCPに適用されます。

Amazon FSx for ONTAP バックエンドドライバ

Amazon FSx for NetApp ONTAP では、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用しながら、AWS にデータを保存することのシンプルさ、俊敏性、セキュリティ、スケーラビリティのメリットを享受できます。FSx for ONTAP は、多数の 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 と互換性のあるドライバーは、 ontap-nas、 ontap-nas-economy、 ontap-nas-flexgroup、 ontap-san、 `ontap-san-economy`です。

NetApp HCI/SolidFire バックエンドドライバ

`solidfire-san`ドライバは、NetApp HCI/SolidFireプラットフォームで使用され、管理者がQoS制限に基づいてTrident用のElementバックエンドを設定するのに役立ちます。Tridentによってプロビジョニングされたボリュームに特定のQoS制限を設定するよう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複製などのElementストレージ機能は、 `solidfire-san`ドライバではサポートされていません。これらの操作は、Element Software Web UIを通じて手動で実行する必要があります。

SolidFireドライバ	Snapshot数	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
solidfire-san	はい	はい	はい [2]	はい	はい	はい	はい [1]

脚注：はい脚注：1[]：Tridentで管理されていません はい脚注：2[]：rawブロック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`azure-netapp-files`ドライバーを使用して"Azure NetApp Files"サービスを管理します。

このドライバの詳細と設定方法については、"Azure NetApp Files の Trident バックエンド構成"を参照してください。

Azure NetApp Filesドライバ	Snapshot数	クローン	マルチアタッチ	QoS	拡張	レプリケーション
azure-netapp-files	はい	はい	はい	はい	はい	はい [1]

脚注：はい脚注：1[]：Tridentで管理されていません

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、アプリケーションのストレージクラスを設計する方法について説明します。

特定のバックエンドの利用

特定のストレージ クラス オブジェクト内でフィルタリングを使用すると、その特定のストレージ クラスで使用されるストレージ プールまたはプール セットを決定できます。ストレージ クラスでは、次の 3 セットのフィルターを設定できます： `storagePools`、 `additionalStoragePools`、および/または `excludeStoragePools`。

`storagePools`パラメータは、指定された属性に一致するプールのセットにストレージを制限するのに役立ちます。 `additionalStoragePools`パラメータは、属性および`storagePools`パラメータによって選択されたプールのセットとともに、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するために使用されます。いずれかのパラメータを単独で使用することも、両方を組み合わせて使用することもでき、適切なストレージプールのセットが選択されるようにすることができます。

`excludeStoragePools`パラメータは、属性に一致するリストされたプールのセットを具体的に除外するために使用されます。

QoS ポリシーをエミュレートする

ストレージクラスを設計してサービス品質ポリシーをエミュレートする場合は、 `media`属性を `hdd`または`ssd`として設定したストレージクラスを作成します。ストレージクラスに記載されている `media`属性に基づいて、Tridentはメディア属性に一致する `hdd`または `ssd`アグリゲートを提供する適切なバックエンドを選択し、特定のアグリゲートへのボリュームのプロビジョニングを指示します。したがって、 `media`属性を`ssd`に設定したストレージクラスPREMIUMを作成でき、これはPREMIUM QoSポリシーとして分類できます。メディア属性を`hdd'に設定した別のストレージクラスSTANDARDを作成でき、これはSTANDARD QoSポリシーとして分類できます。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、シック プロビジョニング、スナップショット、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 プロビジョニングを指示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、必要な機能が有効になっている適切なバックエンドを指定するストレージ 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 バックエンドで利用できます。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 クラスを通じて参照できるバックエンドの抽象化レベルを作成できるため、バックエンドでのボリュームの配置の柔軟性と効率性が向上します。同じサービス クラスで異なるバックエンドを定義できます。さらに、同じバックエンド上に、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 クラスが特定のラベルを持つセレクタで構成されている場合、Trident はボリュームを配置するために、すべてのセレクター 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 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 つを選択してボリュームをプロビジョニングします。

仮想プールの設計

バックエンドを作成するときに、通常は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資格情報と異なるパラメータセットを持つ別のバックエンドを作成することは不可能でした。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の抽象化であり、管理者は、バックエンドに依存しない方法で、セレクターとして Kubernetes ストレージ クラスを通じて参照できるラベルとともにパラメータを定義できます。仮想プールは、Trident でサポートされているすべての NetApp バックエンドに対して定義できます。そのリストには、SolidFire/NetApp

HCI、ONTAP、および Azure NetApp Files が含まれます。

> 仮想プールを定義するときは、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ことをお勧めします。既存の仮想プールの属性を編集/変更せず、代わりに新しい仮想プールを定義することもお勧めします。

異なるサービスレベル／**QoS**のエミュレーション

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することが可能です。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設定する方法を見てみましょう。Azure NetApp Filesバックエンドを、異なるパフォーマンスレベルを表す複数のラベルで構成します。`servicelevel`アスペクトを適切なパフォーマンスレベルに設定し、各ラベルの下に他の必要なアスペクトを追加します。次に、異なる仮想プールにマッピングされる異なるKubernetesストレージクラスを作成します。`parameters.selector`フィールドを使用して、各StorageClassは、ボリュームをホストするために使用できる仮想プールを指定します。

特定のアスペクトセットの割り当て

単一のストレージ バックエンドから、特定の側面を持つ複数の仮想プールを設計できます。そのためには、バックエンドを複数のラベルで構成し、各ラベルの下に必要な側面を設定します。次に、`parameters.selector`フィールドを使用して、異なる仮想プールにマップされる異なるKubernetesストレージクラスを作成します。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で定義された側面が設定されます。

ストレージ プロビジョニングに影響する**PVC**の特性

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が、PVC を作成する際の Trident プロビジョニング決定プロセスに影響を与え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 つはアクセスモードです。希望するモードは、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する可能性があります。

Tridentは、次のマトリックスに従って指定されたアクセス方法で使用されるストレージ プロトコルとの一致を試みます。これは、基盤となるストレージ プラットフォームとは独立しています。

	ReadWriteOnce	**ReadOnlyMany**	**ReadWriteMany**
iSCSI	はい	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 ReadWriteMany PVC の要求を送信すると、ボリュームはプロビジョニングされません。このため、要求者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

永続ボリュームを変更する

永続ボリュームは、2つの例外を除き、Kubernetes内の不変オブジェクトです。作成したら、再利用ポリシー

とサイズを変更できます。ただし、これによってボリュームの一部の側面がKubernetesの外部で変更されるのを防ぐことはできません。これは、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、容量が誤って消費されないようにしたり、あるいは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望ましいことがあります。

> Kubernetes のツリー内プロビジョナーは、現時点では NFS、iSCSI、または FC PV のボリューム サイズ変更操作をサポートしていません。Trident は NFS、iSCSI、FC ボリュームの拡張をサポートします。

PV の接続詳細は作成後に変更できません。

オンデマンドボリューム**Snapshot**を作成

Tridentは、CSIフレームワークを使用したオンデマンドボリュームスナップショットの作成と、スナップショットからのPVCの作成をサポートしています。スナップショットは、ポイントインタイムデータのコピーを維持する便利な方法であり、KubernetesのソースPVから独立したライフサイクルを持ちます。これらのスナップショットを使用してPVCをクローニングできます。

スナップショットからボリュームを作成する

Tridentは、ボリュームスナップショットからのPersistentVolumesの作成もサポートしています。これを実現するには、PersistentVolumeClaimを作成し、 `datasource`をボリュームの作成元として必要なスナップショットとして指定します。Tridentは、スナップショットに存在するデータを含むボリュームを作成することで、このPVCを処理します。この機能により、リージョン間でのデータの複製、テスト環境の作成、破損または壊れた本番ボリューム全体の交換、特定のファイルやディレクトリの取得と別の接続されたボリュームへの転送が可能になります。

クラスタ内のボリュームを移動する

ストレージ管理者は、ストレージコンシューマに影響を与えることなく、ONTAPクラスタ内のアグリゲートとコントローラ間でボリュームを無停止で移動できます。この処理は、TridentまたはKubernetesクラスタに影響しません。ただし、デスティネーションアグリゲートが、Tridentで使用しているSVMがアクセスできるものである必要があります。重要な点として、アグリゲートがSVMに新しく追加された場合は、Tridentに再度追加してバックエンドを更新する必要があります。これにより、Tridentが新しいアグリゲートを認識できるようにSVMの再インベントリが実行されます。

ただし、バックエンド間でのボリュームの移動は、Tridentでは自動的に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、または異なるストレージプラットフォームへの移動が含まれます（そのストレージシステムがTridentに接続されている場合でも）。

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。

ボリュームを拡張する

Tridentは、NFS、iSCSI、FCのPVのリサイズをサポートしています。これにより、ユーザーはKubernetesレイヤーを通じてボリュームを直接リサイズできます。ボリューム拡張は、ONTAPを含むすべての主要なNetAppストレージプラットフォームおよびSolidFire/NetApp HCIバックエンドで可能です。後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たStorageClassで `allowVolumeExpansion`を `true`に設定してください。永続ボリュームのリサイズが必要な場合は、Persistent Volume Claimの `spec.resources.requests.storage`アノテーションを必要なボリュームサイズに編集します。Tridentはストレージクラスタ上で自動的にボリュームのリサイズを行います。

既存のボリュームを**Kubernetes**にインポートする

ボリュームインポートでは、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`、および `azure-netapp-files`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役立ちます。

ONTAPおよび `solidfire-san`ドライバーを使用する場合は、コマンド `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`を使用して、既存のボリュームをKubernetesにインポートし、Tridentで管理します。インポートボリュームコマンドで使用されるPVC YAMLまたはJSONファイルは、Tridentをプロビジョナーとして識別するストレージクラスを指します。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でクローニングします。

> `azure-netapp-files`ドライバを使用する場合は、コマンド `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`を使用して、ボリュームを`Kubernetes`にインポートし、`Trident`で管理します。これにより、一意のボリューム参照が保証されます。

上記のコマンドを実行すると、Tridentはバックエンドでボリュームを見つけて、そのサイズを読み取ります。設定されたPVCのボリュームサイズが自動的に追加され（必要に応じて上書きされます）。Tridentは新しいPVを作成し、KubernetesはPVCをPVにバインドします。

特定のインポートされた PVC を必要とするコンテナがデプロイされた場合、PVC/PV ペアがボリューム インポート プロセスによって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た後、他の問題がなければコンテナが起動するはずです。

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、"netapp.io"の"ブログ"に記載されています。

ログサービス

他のOpenShiftサービスと同様に、ログサービスは、プレイブックに提供されるインベントリファイル（ホスト）によって指定される構成パラメータを使用して、Ansibleを使用してデプロイされます。ここでは、2つのインストール方法について説明します：OpenShiftの初期インストール時にロギングをデプロイする方法と、OpenShiftのインストール後にロギングをデプロイする方法です。

> Red Hat OpenShiftバージョン3.9以降、公式ドキュメントでは、データ破損に関する懸念から、ロギングサービスにNFSを使用しないことを推奨しています。これはRed Hatによる自社製品のテストに基づいています。ONTAP NFSサーバにはこれらの問題はなく、ロギング環境を簡単にバックアップできます。最終的には、ロギングサービスのプロトコルの選択はお客様次第ですが、NetAppプラットフォームを使用する場合、どちらも問題なく動作することを知っておいてください。NFSがお好みであれば、NFSを避ける理由はありません。

ログサービスでNFSを使用する場合は、Ansible変数 `openshift_enable_unsupported_configurations`を `true`に設定して、インストーラーが失敗するのを防ぐ必要があります。

ログサービスは、オプションで、アプリケーションとOpenShiftクラスタ自体のコア運用の両方に導入できます。変数 `openshift_logging_use_ops`を `true`として指定して運用ログを導入することを選択した場合、サービスのインスタンスが2つ作成されます。運用のログインスタンスを制御する変数には「ops」が含まれますが、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によって正しいストレージが確実に利用されるようにするには、デプロイメント方法に応じて Ansible 変数を設定することが重要です。それぞれのデプロイメント方法のオプションを見てみましょう。

以下の表には、ログ サービスに関連するストレージ構成に関係する変数のみが含まれています。その他のオプションについては"Red Hat OpenShiftログ記録ドキュメント"を参照してください。これらは、導入に応じて確認、設定、および使用する必要があります。

以下の表の変数により、Ansible プレイブックは提供された詳細を使用して、ログ サービス用の PV と PVC を作成します。この方法は、OpenShift の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柔軟性が大幅に低下しますが、既存のボリュームが利用可能な場合は、オプションになります。

変数	詳細
`openshift_logging_storage_kind`	`nfs`に設定すると、インストーラーでログ サービス用のNFS PVを作成します。
`openshift_logging_storage_host`	NFSホストのホスト名またはIPアドレス。これは仮想マシンのdataLIFに設定する必要があります。
`openshift_logging_storage_nfs_directory`	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`のように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_logging_storage_volume_name`	作成するPVの名前（例： `pv_ose_logs`）。
`openshift_logging_storage_volume_size`	NFSエクスポートのサイズ（例： `100Gi`）。

OpenShiftクラスタがすでに実行されており、Tridentが導入および設定されている場合、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	詳細
`openshift_logging_es_pvc_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には、trueに設定します。
`openshift_logging_es_pvc_storage_class_name`	PVC で使用されるストレージ クラスの名前。
`openshift_logging_es_pvc_size`	PVC で要求されたボリュームのサイズ。
`openshift_logging_es_pvc_prefix`	ロギング サービスで使用される PVC のプレフィックス。
`openshift_logging_es_ops_pvc_dynamic`	`true`に設定すると、オペレーションログインスタンスに動的にプロビジョニングされたボリュームを使用します。
`openshift_logging_es_ops_pvc_storage_class_name`	オペレーションログインスタンスのストレージクラスの名前。

変数	詳細
`openshift_logging_es_ops_pvc_size`	ops インスタンスのボリューム要求のサイズ。
`openshift_logging_es_ops_pvc_prefix`	ops インスタンス PVC のプレフィックス。

ログスタックをデプロイする

初期OpenShiftインストール プロセスの一部としてログ記録を導入する場合は、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オブジェクトを設定して導入するため、Ansibleが完了するとすぐにサービスが利用できるようになります。

ただし、初期インストール後にデプロイする場合は、Ansible でコンポーネント プレイブックを使用する必要があります。このプロセスは、OpenShift のバージョンによって若干異なる場合があるため、お使いのバージョンの"Red Hat OpenShift Container Platform 3.11ドキュメント"を必ずお読みになり、従ってください。

メトリクスサービス

メトリクスサービスは、管理者にOpenShiftクラスタのステータス、リソース使用率、可用性に関する貴重な情報を提供します。また、ポッドの自動スケール機能にも必要であり、多くの組織はメトリクスサービスからのデータをチャージバックやショーバックアプリケーションに使用しています。

ログサービスと同様に、OpenShift全体として、Ansibleはメトリクスサービスのデプロイに使用されます。また、ロ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時、またはコンポーネントインストール方法を使用して運用開始後にデプロイできます。次の表には、メトリクスサービスの永続ストレージを構成するときに重要な変数が含まれています。

以下の表には、メトリック サービスに関連するストレージ構成に関係する変数のみが含まれています。ドキュメントには他にも多くのオプションが記載されており、導入に応じて確認、設定、使用する必要があります。

変数	詳細
`openshift_metrics_storage_kind`	`nfs`に設定すると、インストーラーでログ サービス用のNFS PVを作成します。
`openshift_metrics_storage_host`	NFSホストのホスト名またはIPアドレス。これは、SVMのdataLIFに設定する必要があります。
`openshift_metrics_storage_nfs_directory`	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_metrics`のように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`	作成するPVの名前（例： `pv_ose_metrics`）。
`openshift_metrics_storage_volume_size`	NFSエクスポートのサイズ（例： `100Gi`）。

OpenShiftクラスタがすでに実行されており、Tridentが導入および設定されている場合、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	詳細
`openshift_metrics_cassandra_pvc_prefix`	メトリック PVC に使用するプレフィックス。
`openshift_metrics_cassandra_pvc_size`	要求するボリュームのサイズ。

変数	詳細
`openshift_metrics_cassandra_storage_type`	メトリックに使用するストレージのタイプ。Ansibleが適切なストレージクラスを持つPVCを作成するには、これをdynamicに設定する必要があります。
`openshift_metrics_cassanda_pvc_storage_class_name`	使用するストレージ クラスの名前。

メトリクスサービスを導入する

hosts/inventory ファイルに適切な Ansible 変数を定義して、Ansible を使用してサービスをデプロイします。OpenShift のインストール時にデプロイする場合は、PV が自動的に作成されて使用されます。OpenShift のインストール後にコンポーネントプレイブックを使用してデプロイする場合は、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後、サービスがデプロイされます。

上記の変数と導入プロセスは、OpenShiftのバージョンごとに変更される可能性があります。お使いの環境に合わせて構成されるように、バージョンに応じた"Red HatのOpenShift導入ガイド"を必ず確認して従ってください。

データ保護とディザスタ リカバリ

Tridentを使用して作成されたTridentおよび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。永続性を必要とするアプリケーションごとに、データ保護およびリカバリ戦略を用意する必要があります。

Tridentのレプリケーションとリカバリ

災害が発生した場合に備えて、Tridentを復元するための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保存および管理し、Kubernetes クラスター etcd を使用してメタデータを保存します。

手順

1. "Kubernetes：etcdクラスタのバックアップ"を使用してKubernetesクラスタetcdをバックアップします。
2. バックアップアーティファクトをFlexVolボリュームに配置する

> NetAppでは、FlexVolが存在するSVMをSnapMirror関係で別のSVMに保護することを推奨しています。

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すると、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を含むボリュームを、マスターノードとしてセットアップされ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 `/etc/kubernetes/pki`にコピーし、etcdメンバーファイルを `/var/lib/etcd`にコピーします。
3. "Kubernetes：etcdクラスタのリストア"を使用して、etcdバックアップからKubernetesクラスタを復元します。
4. 実行 `kubectl get crd`すべてのTridentカスタムリソースが起動したことを確認し、Tridentオブジェクトを取得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はレプリケーション関係を構成することはできませんが、ストレージ管理者は "ONTAP SnapMirror"を使用してSVMをレプリケートできます。

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。システムが復元されたら、プライマリに切り替えることができます。

タスク概要

SnapMirror SVM レプリケーション機能を使用する際は、以下の点にご留意ください：

- SVM-DR が有効になっている各 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。
- レプリケーションを必要としないボリュームがSVM-DRをサポートするバックエンドにプロビジョニングされることを回避するために、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構成します。
- アプリケーション管理者は、レプリケーションに関連する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計画を慎重に検討する必要があります。

SVMレプリケーション

"ONTAP：SnapMirror SVMレプリケーション"を使用してSVMレプリケーション関係を作成できます。

SnapMirrorでは、複製する内容を制御するオプションを設定できます。Tridentを使用したSVMリカバリを実行する際に選択したオプションを把握しておく必要があります。

- "-identity-preserve true"SVM設定全体をレプリケートします。
- "-discard-configs network"LIFおよび関連するネットワーク設定は除外されます。
- "-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リカバリ

TridentはSVMの障害を自動的に検出しません。災害が発生した場合、管理者は手動でTridentフェイルオーバーを新しいSVMに開始できます。

手順

1. スケジュール済みおよび実行中の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. SVMレプリケーションの設定時に `-identity-preserve false`または `-discard-config network`を指定した場合は、Tridentバックエンド定義ファイル内の `managementLIF`および `dataLIF`を更新します。
3. `storagePrefix`がTridentバックエンド定義ファイルに存在することを確認します。このパラメータは変更

できません。 `storagePrefix`を省略すると、バックエンドの更新が失敗します。

4. 次のコマンドを使用して、新しいデスティネーションSVM名を反映するように必要なすべてのバックエンドを更新します：

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. `-identity-preserve false`または `discard-config network`を指定した場合は、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。

> `-identity-preserve true`を指定した場合、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は、デスティネーションSVMがアクティブ化されたときにデータの提供を開始します。

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"ONTAP SnapMirrorレプリケーションとリカバリ"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。

その後、復元したボリュームを"tridentctl volume import"を使用してTridentにインポートできます。

> `ontap-nas-economy`、 `ontap-san-economy`、または `ontap-flexgroup-economy`ドライバではインポートはサポートされていません。

Snapshotデータ保護

以下を使用してデータを保護および復元できます：

- 永続ボリューム（PV）のKubernetesボリュームスナップショットを作成するための外部スナップショットコントローラーとCRD。

  "ボリュームスナップショット"

- ONTAPスナップショットを使用して、ボリュームの内容全体を復元したり、個々のファイルまたはLUNをリカバリしたりできます。

  "ONTAPスナップショット"

Tridentを使用したステートフルアプリケーションのフェイルオーバーの自動化

Tridentの強制デタッチ機能を使用すると、Kubernetesクラスター内の正常でないノードからボリュームを自動的にデタッチして、データ破損を防ぎ、アプリケーションの可用性を確保できます。この機能は、ノードが応答しなくなったり、メンテナンスのためにオフラインになったりするシナリオで特に役立ちます。

強制デタッチの詳細

強制デタッチは `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-economy`でのみ使用できます。強制デタッチを有効にする前に、Kubernetesクラスタで非正常なノードシャットダウン（NGNS）を有効にする必要があります。NGNSはKubernetes 1.28以降ではデフォルトで有効になっています。詳細については、"Kubernetes：非正常なノードシャットダウン"を参照してください。

> `ontap-nas`または `ontap-nas-economy`ドライバを使用する場合は、バックエンド構成の `autoExportPolicy`パラメータを `true`に設定して、Tridentが管理対象のエクスポートポリシーを使用してテイントが適用されたKubernetesノードからのアクセスを制限できるようにする必要があります。

> Tridentは Kubernetes NGNS に依存しているため、すべての許容できないワークロードが再スケジュールされるまで、異常なノードから `out-of-service`テイントを削除しないでください。テイント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クラスタ管理者がノードに `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute`テイントを適用し、 `enableForceDetach`が `true`に設定されている場合、Tridentはノードのステータスを判断し、次の処理を実行します：

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 `dirty`（新規パブリケーションには安全ではない）としてマークします。

> Tridentコントローラは、Tridentノードポッドによってノードが再認定されるまで（ `dirty`とマークされた後）、新しいボリューム公開要求を拒否します。マウントされたPVCで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完了した後でも）は、Tridentがノード `clean`（新しい公開に対して安全）を検証できるまで受け入れられません。

ノードの健全性が回復し、汚染が除去されると、Tridentは次の処理を実行します：

1. ノード上の古くなった公開パスを識別してクリーンアップします。
2. ノードが `cleanable`状態（サービス停止の汚染が除去され、ノードが `Ready`状態）で、すべての古い公開済みパスがクリーンである場合、Tridentはノードを `clean`として再度承認し、新しい公開ボリュームをノードに許可します。

自動フェイルオーバーの詳細

"node health check（NHC）operator"との統合により、強制デタッチプロセスを自動化できます。ノード障害が発生すると、NHCはTridentの名前空間で障害が発生したノードを定義するTridentNodeRemediation CRを作成することで、Tridentノード修復（TNR）をトリガーし、強制デタッチを自動的に実行します。TNRはノード障害時にのみ作成され、ノードがオンラインに戻るかノードが削除されるとNHCによって削除されます。

障害ノードのポッド削除プロセス

自動フェイルオーバーは、障害が発生したノードから削除するワークロードを選択します。TNRが作成されると、TNRコントローラーはノードをダーティとしてマークし、新しいボリュームの公開を防止し、強制デタッチがサポートされているポッドとそのボリュームアタッチメントの削除を開始します。

強制デタッチでサポートされているすべてのボリューム/PVCは、自動フェイルオーバーでもサポートされます：

- 自動エクスポートポリシーを使用するNASおよびNASエコノミーボリューム（SMBはまだサポートされていません）。
- SAN および SAN-economy ボリューム。

[強制デタッチの詳細]を参照してください。

デフォルトの動作：

- 強制デタッチがサポートされているボリュームを使用しているポッドは、障害が発生したノードから削除されます。Kubernetesはこれらを正常なノードに再スケジュールします。
- 強制デタッチでサポートされていないボリューム（Trident以外のボリュームを含む）を使用するポッドは、障害が発生したノードから削除されません。
- ステートレスポッド（PVCではない）は、ポッドアノテーション `trident.netapp.io/podRemediationPolicy: delete`が設定されていない限り、障害が発生したノードから削除されません。

ポッド削除動作のオーバーライド：

ポッドの削除動作は、ポッド アノテーションを使用してカスタマイズできます `trident.netapp.io/podRemediationPolicy[retain, delete]`。これらのアノテーションは、フェイルオーバーが発生したときに検査され、使用されます。フェイルオーバー後にアノテーションが消えないように、Kubernetesデプロイメント/レプリカセット ポッド仕様にアノテーションを適用します：

- `retain` - 自動フェイルオーバー中に、障害が発生したノードからPodは削除されません。
- `delete` - 自動フェイルオーバー中に、障害が発生したノードからPodが削除されます。

これらの注釈はどのポッドにも適用できます。

- 強制デタッチをサポートするボリュームの場合、障害が発生したノードでのみI/O操作がブロックされます。
- 強制デタッチをサポートしていないボリュームの場合、データ破損およびマルチアタッチの問題が発生するリスクがあります。

TridentNodeRemediation CR

TridentNodeRemediation（TNR）CRは障害が発生したノードを定義します。TNRの名前は、障害が発生したノードの名前です。

TNRの例：

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediation
metadata:
  name: <K8s-node-name>
spec: {}
```

TNR の状態：TNR のステータスを表示するには、次のコマンドを使用します。
`kubectl get tnr <name> -n <trident-namespace>`

TNR は次のいずれかの状態になります：

- 修復中：
  - そのノードにマウントされた、強制デタッチでサポートされているボリュームへのバックエンドI/Oアクセスを停止します。
  - Tridentノードオブジェクトはダーティとしてマークされています（新しいパブリケーションには安全ではありません）。
  - ノードからポッドとボリュームアタッチメントを削除します
- *NodeRecoveryPending*：
  - コントローラはノードがオンラインに戻るのを待機しています。
  - ノードがオンラインになると、パブリッシュ強制により、ノードがクリーンであり、新しいボリュームのパブリケーションの準備ができていることが確認されます。
- ノードが K8s から削除されると、TNR コントローラーは TNR を削除し、調整を停止します。
- 成功：
  - すべての修復およびノード回復手順が正常に完了しました。ノードはクリーンであり、新しいボリュームの公開の準備ができています。
- *Failed*：
  - 回復不能なエラーです。エラー理由は、CRのstatus.messageフィールドに設定されます。

自動フェイルオーバーの有効化

前提条件：

- 自動フェイルオーバーを有効にする前に、強制デタッチが有効になっていることを確認してください。詳細については、[強制デタッチの詳細]を参照してください。
- Kubernetesクラスタにノードヘルスチェック（NHC）をインストールします。
  - "operator-sdkをインストールする"。
  - まだインストールされていない場合は、クラスタに Operator Lifecycle Manager（OLM）をインストールします `operator-sdk olm install`
  - Node Health check Operatorをインストールします `kubectl create -f https://operatorhub.io/install/node-healthcheck-operator.yaml`。

> 以下の[Integrating Custom Node Health Check Solutions]セクションで指定されている別の方法を使用して、ノード障害を検出することもできます。

"ノードヘルスチェックオペレーター"を参照してください。

手順

1. Trident ネームスペース内に NodeHealthCheck（NHC）CR を作成して、クラスター内のワーカーノードを監視します。例：

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: <CR name>
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  remediationTemplate:
    apiVersion: trident.netapp.io/v1
    kind: TridentNodeRemediationTemplate
    namespace: <Trident installation namespace>
    name: trident-node-remediation-template
  minHealthy: 0 # Trigger force-detach upon one or more node failures
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 0s
    - type: Ready
      status: Unknown
      duration: 0s
```

2. `trident`ネームスペースにノードヘルスチェックCRを適用します。

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

上記の CR は、K8s ワーカーノードのノード状態 Ready：false および Unknown を監視するように構成されています。自動フェイルオーバーは、ノードが Ready：false または Ready：Unknown 状態になるとトリガーされます。

CR の `unhealthyConditions`では 0 秒の猶予期間が使用されます。これにより、K8s がノードからのハートビートを失った後に設定されるノード条件 Ready：false を K8s が設定すると、自動フェイルオーバーが直ちにトリガーされます。K8s では、最後のハートビートの後に Ready：false を設定するまで、デフォルトで 40 秒間待機します。この猶予期間は、K8s デプロイメント オプションでカスタマイズできます。

追加の設定オプションについては、"Node-Healthcheck-Operator ドキュメント"を参照してください。

追加のセットアップ情報

Tridentを強制デタッチを有効にしてインストールすると、NHCとの統合を容易にするために、Trident名前空間に2つの追加リソースが自動的に作成されます：TridentNodeRemediationTemplate（TNRT）とClusterRole。

TridentNodeRemediationTemplate（TNRT）：

TNRT は NHC コントローラのテンプレートとして機能し、必要に応じて TNR リソースを生成します。

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

ClusterRole：

強制デタッチが有効になっている場合、インストール中にクラスター ロールも追加されます。これにより、NHCにTrident名前空間のTNRに対する権限が付与されます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.ext-remediation/aggregate-to-ext-remediation: "true"
  name: tridentnoderemediation-access
rules:
- apiGroups:
  - trident.netapp.io
  resources:
  - tridentnoderemediationtemplates
  - tridentnoderemediations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

K8s クラスタのアップグレードとメンテナンス

フェイルオーバーを防ぐには、ノードがダウンしたり再起動することが予想される K8s メンテナンスまたはアップグレード中は、自動フェイルオーバーを一時停止します。NHC CR（上記で説明）を一時停止するには、その CR にパッチを適用します。

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

これにより、自動フェイルオーバーが一時停止されます。自動フェイルオーバーを再度有効にするには、メンテナンス完了後に仕様からpauseRequestsを削除します。

制限事項

- 強制デタッチでサポートされているボリュームの場合、障害が発生したノード上でのみ I/O 操作が防止されます。強制デタッチでサポートされているボリューム/PVC を使用しているポッドのみが自動的に削除されます。
- 自動フェイルオーバーと強制デタッチは、trident-controller ポッド内で実行されます。trident-controller をホストしているノードに障害が発生した場合、K8s がポッドを正常なノードに移動するまで、自動フェイルオーバーは遅延されます。

カスタムノードヘルスチェックソリューションの統合

自動フェイルオーバーをトリガーするために、Node Healthcheck Operator を代替のノード障害検出ツールに置き換えることができます。自動フェイルオーバーメカニズムとの互換性を確保するために、カスタムソリューションは次の要件を満たす必要があります：

- ノード障害が検出されると、障害が発生したノードの名前を TNR CR 名として使用して TNR を作成します。
- ノードが回復し、TNR が Succeeded 状態になったら、TNR を削除します。

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てください。

Trident を独自の名前空間で実行する

アプリケーション、アプリケーション管理者、ユーザー、および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。これにより、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できます。

他のアプリケーションやユーザーを Trident から分離するには、常に Trident を独自の Kubernetes 名前空間にインストールしてください(`trident`）。Trident を独自の名前空間に配置することで、Kubernetes 管理担当者のみが Trident ポッドおよび名前空間 CRD オブジェクトに保存されている成果物（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。管理者のみが Trident 名前空間にアクセスでき、したがって `tridentctl`アプリケーションにアクセスできるようにする必要があります。

ONTAP SANバックエンドで**CHAP**認証を使用する

Tridentは、ONTAP SANワークロード（ `ontap-san`および `ontap-san-economy`ドライバーを使用）のCHAPベースの認証をサポートしています。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ーを使用する ONTAP バックエンドの場合、 Trident は双方向 CHAP を設定し、CHAP ユーザー名とシークレットを `tridentctl`で管理できます。"ONTAP SANドライバを使用してバックエンドを設定する準備をします"を参照して、 Trident が ONTAP バックエンドで CHAP を設定する方法を理解し

てください。

NetApp HCI および **SolidFire** バックエンドで **CHAP** 認証を使用する

NetAppは、ホストとNetApp HCIおよびSolidFireバックエンド間の認証を確実にするために、双方向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がインストールされると、CHAPシークレットを管理し、それぞれのPVに対応する `tridentvolume`CRオブジェクトに保存します。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経由でNetApp HCIおよびSolidFireシステムと通信します。

Trident によって作成されたボリュームは、どのボリューム アクセス グループにも関連付けられていません。

Tridentと**NVE**および**NAE**を使用

NetApp ONTAP は、ディスクが盗難、返却、または再利用された場合に機密データを保護するために、保存データの暗号化を提供します。詳細については、"NetApp Volume Encryptionの設定 - 概要"を参照してください。

- NAEがバックエンドで有効になっている場合、Tridentでプロビジョニングされたボリュームは、NAE対応になります。
  - NVE 暗号化フラグを `""`に設定すると、NAE 対応ボリュームを作成できます。
- NAEがバックエンドで有効になっていない場合、バックエンド構成でNVE暗号化フラグが `false`（デフォルト値）に設定されていない限り、TridentでプロビジョニングされたボリュームはすべてNVEが有効になります。

NAE 対応のバックエンドで Trident に作成されたボリュームは、NVE または NAE で暗号化されている必要があります。

- 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 暗号化を上書きして、ボリュームごとに特定の暗号化キーを使用できます。
- NAE対応のバックエンドでNVE暗号化フラグを `false`に設定すると、NAE対応のボリュームが作成されます。NVE暗号化フラグを `false`に設定してもNAE暗号化を無効にすることはできません。

- 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 `true`に設定します。

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 SAN構成オプション"
- "ONTAP NAS 構成オプション"

Linux Unified Key Setup（LUKS）

Linux Unified Key Setup（LUKS）を有効にして、Trident上のONTAP SANおよびONTAP SAN ECONOMY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ローテーションとボリューム拡張をサポートします。

Tridentでは、LUKSで暗号化されたボリュームは、"NIST"で推奨されているaes-xts-plain64暗号とモードを使

用します。

LUKS 暗号化は ASA r2 システムではサポートされていません。ASA r2 システムの詳細については、"ASA r2ストレージシステムの詳細"を参照してください。

開始する前に

- ワーカー ノードには、cryptsetup 2.1 以上（3.0 未満）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、"Gitlab：cryptsetup"を参照してください。
- パフォーマンス上の理由から、NetAppではワーカーノードで Advanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨します。AES-NI の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、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、以下をご覧ください："Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にする

Linux Unified Key Setup（LUKS）を使用して、ONTAP SANおよびONTAP SAN ECONOMYボリュームのボリュームごとにホスト側の暗号化を有効にすることができます。

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 のバックエンド設定オプションの詳細については、"ONTAP SAN構成オプション"を参照してください。

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. `parameters.selector`を使用して、LUKS暗号化を使用するストレージプールを定義します。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
```

制限事項

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 `luksEncryption`を（`true`に設定する必要があります。 `luksEncryption`オプションは、次の例に示すように、ボリュームがLUKS準拠（（`true`）かLUKS非準拠（(`false`）かをTridentに通知します。

```
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**構成

LUKS ボリュームを動的にインポートするには、アノテーション `trident.netapp.io/luksEncryption`を `true`に設定し、この例に示すように、PVC に LUKS 対応ストレージクラスを含めます。

```
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する

LUKS パスフレーズをローテーションし、ローテーションを確認できます。

> パスフレーズがボリューム、スナップショット、またはシークレットによって参照されなくなったことを確認するまで、パスフレーズを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なくなり、データは暗号化されたままアクセスできなくなる可能性があります。

タスク概要

新しい LUKS パスフレーズが指定された後にボリュームをマウントするポッドが作成されると、LUKS パスフレーズのローテーションが発生します。新しいポッドが作成されると、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。

- ボリューム上のパスフレーズがシークレット内のアクティブなパスフレーズと一致しない場合は、ローテーションが発生します。
- ボリューム上のパスフレーズがシークレット内のアクティブなパスフレーズと一致する場合、 `previous-luks-passphrase` パラメータは無視されます。

手順

1. `node-publish-secret-name`と `node-publish-secret-namespace` StorageClassパラメータを追加します。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. ボリュームまたはスナップショット上の既存のパスフレーズを特定します。

Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

Snapshot

```
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以前のパスフレーズを指定します。`previous-luke-passphrase-name`と `previous-luks-passphrase`が以前のパスフレーズと一致することを確認します。

```
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。

Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

ボリュームとスナップショットで新しいパスフレーズのみが返されたときに、パスフレーズが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 `luksPassphraseNames: ["B", "A"]`、ローテーションは不完全です。新しいポッドをトリガーして、ローテーションを完了させることができます。

ボリューム拡張を有効にする

LUKS で暗号化されたボリュームでストレージ拡張を有効にすることができます。

手順

1. `CSINodeExpandSecret`機能ゲートを有効にします（ベータ版 1.25 以降）。詳細については、"Kubernetes 1.25：CSI ボリュームのノード駆動型拡張に Secret を使用する"を参照してください。
2. `node-expand-secret-name`と `node-expand-secret-namespace` StorageClassパラメータを追加します。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

オンラインストレージ拡張を開始すると、kubelet は適切な資格情報をドライバーに渡します。

Kerberos のインフライト暗号化

Kerberos インフライト暗号化を使用すると、マネージドクラスタとストレージバックエンド間のトラフィックの暗号化を有効にして、データアクセスのセキュリティを向上させることができます。

Trident は、ストレージバックエンドとして ONTAP の Kerberos 暗号化をサポートしています：

- オンプレミス **ONTAP** - Trident は、Red Hat OpenShift および上流の Kubernetes クラスタからオンプレミス ONTAP ボリュームへの NFSv3 および NFSv4 接続での Kerberos 暗号化をサポートします。

NFS 暗号化を使用するボリュームを作成、削除、サイズ変更、スナップショット、クローン、読み取り専用クローン、およびインポートできます。

オンプレミス **ONTAP** ボリュームでインフライト **Kerberos** 暗号化を設定する

管理対象クラスタとオンプレミス ONTAP ストレージバックエンド間のストレージトラフィックで Kerberos 暗号化を有効にすることができます。

> オンプレミス ONTAP ストレージバックエンドを使用した NFS トラフィックの Kerberos 暗号化は、 `ontap-nas`ストレージドライバーを使用する場合にのみサポートされます。

開始する前に

- `tridentctl`ユーティリティにアクセスできることを確認してください。
- ONTAP ストレージバックエンドへの管理者アクセス権があることを確認してください。
- ONTAP ストレージバックエンドから共有するボリュームの名前を必ず確認してください。
- NFS ボリュームの Kerberos 暗号化をサポートするように ONTAP ストレージ VM を準備しておいてください。手順については "データLIFでKerberosを有効にする"を参照してください。

- Kerberos 暗号化で使用する NFSv4 ボリュームが正しく構成されていることを確認します。NetApp NFSv4 ドメイン設定セクション（13 ページ） "NetApp NFSv4 の機能強化とベストプラクティスガイド" を参照してください。

ONTAP エクスポートポリシーの追加または変更

ONTAP ストレージ VM ルートボリュームおよびアップストリーム Kubernetes クラスタと共有される ONTAP ボリュームに対して、Kerberos 暗号化をサポートする既存の ONTAP エクスポートポリシーにルールを追加するか、新しいエクスポートポリシーを作成する必要があります。追加するエクスポートポリシールール、または作成する新しいエクスポートポリシーは、次のアクセスプロトコルとアクセス権限をサポートする必要があります（

アクセス プロトコル

NFS、NFSv3、および NFSv4 アクセスプロトコルを使用してエクスポートポリシーを構成します。

アクセスの詳細

ボリュームのニーズに応じて、3 つの異なるバージョンの Kerberos 暗号化のいずれかを設定できます：

- **Kerberos 5** -（認証と暗号化）
- **Kerberos 5i** -（ID保護を備えた認証と暗号化）
- **Kerberos 5p** -（IDとプライバシー保護を使用した認証と暗号化）

適切なアクセス権限を持つ ONTAP エクスポート ポリシー ルールを設定します。たとえば、クラスタが Kerberos 5i と Kerberos 5p 暗号化を組み合わせて NFS ボリュームをマウントする場合は、次のアクセス設定を使用します：

タイプ	読み取り専用アクセス	読み取り/書き込みアクセス	スーパーユーザーアクセス
UNIX	有効	有効	有効
Kerberos 5i	有効	有効	有効
Kerberos 5p	有効	有効	有効

ONTAP エクスポートポリシーとエクスポートポリシールールの作成方法については、次のドキュメントを参照してください：

- "エクスポート ポリシーの作成"
- "エクスポート ポリシーへのルールの追加"

ストレージバックエンドを作成する

Kerberos 暗号化機能を含む Trident ストレージバックエンド構成を作成できます。

タスク概要

Kerberos暗号化を設定するストレージバックエンド構成ファイルを作成するときに、 `spec.nfsMountOptions` パラメータを使用して3つの異なるバージョンのKerberos暗号化のいずれかを指定できます：

- `spec.nfsMountOptions: sec=krb5`（認証と暗号化）
- `spec.nfsMountOptions: sec=krb5i`（ID 保護を備えた認証と暗号化）

- `spec.nfsMountOptions: sec=krb5p`（IDとプライバシー保護を備えた認証と暗号化）

Kerberos レベルを 1 つだけ指定します。パラメータリストで複数の Kerberos 暗号化レベルを指定した場合、最初のオプションのみが使用されます。

手順

1. マネージド クラスターで、次の例を使用してストレージ バックエンド構成ファイルを作成します。括弧内の値<>を環境からの情報に置き換えます：

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. 前の手順で作成した構成ファイルを使用して、バックエンドを作成します：

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

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs
```

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

ストレージクラスを作成する

Kerberos 暗号化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。

タスク概要

ストレージクラスオブジェクトを作成するときに、 `mountOptions`パラメータを使用してKerberos暗号化の3つの異なるバージョンのいずれかを指定できます：

- `mountOptions: sec=krb5`（認証と暗号化）
- `mountOptions: sec=krb5i`（ID 保護を備えた認証と暗号化）
- `mountOptions: sec=krb5p`（IDとプライバシー保護を備えた認証と暗号化）

Kerberos レベルを 1 つだけ指定します。パラメータリストで複数の Kerberos 暗号化レベルを指定した場合、最初のオプションのみが使用されます。ストレージバックエンド構成で指定した暗号化レベルがストレージクラスオブジェクトで指定したレベルと異なる場合は、ストレージクラスオブジェクトが優先されます。

手順

1. 次の例を使用して、StorageClass Kubernetes オブジェクトを作成します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. ストレージクラスを作成します：

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. ストレージクラスが作成されていることを確認します：

```
kubectl get sc ontap-nas-sc
```

次のような出力が表示されます。

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになります。手順については、 "ボリュームをプロビジョニングする"を参照してください。

Azure NetApp Files ボリュームでインフライト Kerberos 暗号化を設定する

マネージドクラスタと単一のAzure NetApp Filesストレージバックエンド、またはAzure NetApp Filesストレージバックエンドの仮想プール間のストレージトラフィックでKerberos暗号化を有効にすることができます。

開始する前に

- 管理対象の Red Hat OpenShift クラスタで Trident が有効になっていることを確認してください。
- `tridentctl`ユーティリティにアクセスできることを確認してください。
- Kerberos 暗号化用の Azure NetApp Files ストレージバックエンドを準備済みであることを確認してください。要件を確認し、 "Azure NetApp Files のドキュメント"の手順に従ってください。
- Kerberos 暗号化で使用する NFSv4 ボリュームが正しく構成されていることを確認します。NetApp NFSv4 ドメイン設定セクション（13 ページ） "NetApp NFSv4 の機能強化とベストプラクティスガイド" を参照してください。

ストレージバックエンドを作成する

Kerberos 暗号化機能を含む Azure NetApp Files ストレージバックエンド構成を作成できます。

タスク概要

Kerberos 暗号化を構成するストレージバックエンド構成ファイルを作成するときに、次の 2 つのレベルのいずれかで適用されるように定義できます：

- `spec.kerberos`フィールドを使用した*ストレージバックエンドレベル*
- `spec.storage.kerberos`フィールドを使用した*仮想プールレベル*

仮想プールレベルで設定を定義する場合、ストレージクラスのラベルを使用してプールが選択されます。

どちらのレベルでも、 Kerberos 暗号化の 3 つの異なるバージョンのいずれかを指定できます：

- `kerberos: sec=krb5`（認証と暗号化）
- `kerberos: sec=krb5i`（ID 保護を備えた認証と暗号化）
- `kerberos: sec=krb5p`（IDとプライバシー保護を備えた認証と暗号化）

手順

1. 管理対象クラスタで、ストレージバックエンドを定義する必要がある場所（ストレージバックエンドレベルまたは仮想プールレベル）に応じて、次のいずれかの例を使用してストレージバックエンド構成ファイルを作成します。括弧内の値<>を環境からの情報に置き換えます：

ストレージバックエンドレベルの例

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

仮想プールレベルの例

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

2. 前の手順で作成した構成ファイルを使用して、バックエンドを作成します：

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

バックエンドの作成に失敗した場合は、バックエンドの構成に問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます：

```
tridentctl logs
```

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

ストレージクラスを作成する

Kerberos 暗号化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。

手順

1. 次の例を使用して、StorageClass Kubernetes オブジェクトを作成します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. ストレージクラスを作成します：

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. ストレージクラスが作成されていることを確認します：

```
kubectl get sc -sc-nfs
```

次のような出力が表示されます。

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

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになります。手順については、 "ボリュームをプロビジョニングする"を参照してください。

Trident Protectでアプリケーションを保護

Trident Protectについて学ぶ

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に支えられたステートフルKubernetesアプリケーションの機能性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全体でのコンテナ化されたワークロードの管理、保護、移動を簡素化します。また、APIとCLIを通じて自動化機能も提供します。

カスタムリソース（CR）を作成するか、Trident Protect CLIを使用することで、Trident Protectを使用してアプリケーションを保護できます。

次の手順

Trident Protect をインストールする前に、その要件について確認できます：

- "Trident Protect の要件"

Trident Protectのインストール

Trident Protect 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 Protectを導入および運用するには、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 Protect Kubernetes クラスタの互換性

Trident Protect は、以下を含む幅広いフルマネージドおよびセルフマネージド Kubernetes サービス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
- SUSE Rancher
- VMware Tanzu ポートフォリオ
- アップストリーム Kubernetes

- Trident Protect バックアップは Linux コンピューティング ノードでのみサポートされます。Windows コンピューティング ノードはバックアップ操作ではサポートされていません。
- Trident Protect をインストールするクラスタが、実行中のスナップショット コントローラと関連する CRD で構成されていることを確認してください。スナップショット コントローラをインストールするには、 "これらの手順"を参照してください。
- 少なくとも1つのVolumeSnapshotClassが存在することを確認してください。詳細については、"VolumeSnapshotClass"を参照してください。

Trident Protect ストレージバックエンドの互換性

Trident Protect は次のストレージ バックエンドをサポートしています：

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP ストレージアレイ
- Google Cloud NetApp Volumes
- Azure NetApp Files

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たNetAppストレージが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）。
- NetApp ONTAP ストレージバックエンドがあることを確認してください。
- バックアップを保存するためのオブジェクト ストレージ バケットが設定されていることを確認します。
- アプリケーションまたはアプリケーション データ管理操作に使用する予定のアプリケーション名前空間を作成します。Trident Protect ではこれらの名前空間は作成されません。カスタム リソースに存在しない名前空間を指定すると、操作は失敗します。

nas-economyボリュームの要件

Trident Protect は、nas-economy 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。スナップショット、クローン、および SnapMirror レプリケーションの nas-economy ボリュームへの実行は、現在サポートされていません。Trident Protect で使用する予定の各 nas-economy ボリュームについて、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、スナップショット 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スナップショット 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
```

スナップショットディレクトリを有効にするには、各nas-economyボリュームに対して次のコマンドを実行し、 `<volume-UUID>`を変更したいボリュームのUUIDに置き換えます：

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

新しいボリュームのスナップショット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 `snapshotDir`を `true`に設定します。既存のボリュームは影響を受けません。

KubeVirt VM でデータを保護

Trident Protectは、データ保護操作中にKubeVirt仮想マシンのファイルシステムのフリーズとアンフリーズ機能を提供し、データの整合性を確保します。VMフリーズ操作の設定方法とデフォルトの動作はTrident Protectのバージョンによって異なり、新しいリリースではHelmチャートパラメータを通じて簡素化された設定が提供されます。

復元操作中は、仮想マシン（VM）用に作成された `VirtualMachineSnapshots`は復元されません。

Trident Protect 25.10以降

Trident Protectは、データ保護処理中にKubeVirtファイルシステムを自動的にフリーズおよびアンフリーズして、整合性を確保します。Trident Protect 25.10以降では、Helmチャートのインストール時に `vm.freeze`パラメータを使用してこの動作を無効にできます。このパラメータはデフォルトで有効になっています。

```
helm install ... --set vm.freeze=false ...
```

Trident Protect 24.10.1～25.06

Trident Protect 24.10.1以降、Trident Protectはデータ保護操作中にKubeVirtファイルシステムを自動的にフリーズおよびアンフリーズします。オプションで、次のコマンドを使用してこの自動動作を無効にすることができます：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

Trident Protect 24.10

Trident Protect 24.10は、データ保護処理中にKubeVirt VMファイルシステムの整合性のある状態を自動的に保証するものではありません。KubeVirt VMデータをTrident Protect 24.10を使用して保護する場合は、データ保護処理の前に、ファイルシステムのフリーズ/アンフリーズ機能を手動で有効にする必要があります。これにより、ファイルシステムが整合性のある状態になることが保証されます。

Trident Protect 24.10を設定して、"仮想化の設定"を使用し、次のコマンドを実行することで、データ保護操作中にVMファイルシステムの凍結と解凍を管理できます：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

SnapMirror レプリケーションの要件

NetApp SnapMirrorレプリケーションは、以下のONTAPソリューションでTrident Protectと併用できます：

- オンプレミスのNetApp FAS、AFF、ASAシステム。Trident保護を使用したSnapMirrorレプリケーションは、現在ASA r2システムではサポートされていません。
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

SnapMirror レプリケーションのための ONTAP クラスタ要件

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てください：

- **NetApp Trident**：NetApp Tridentは、ONTAPをバックエンドとして利用するソースKubernetesクラスタとデスティネーションKubernetesクラスタの両方に存在する必要があります。Trident Protectは、次のドライバでサポートされるストレージクラスを使用して、NetApp SnapMirrorテクノロジによるレプリケーションをサポートします：
  - `ontap-nas`：NFS
  - `ontap-san`：iSCSI
  - `ontap-san`：FC
  - `ontap-san`：NVMe/TCP（ONTAP バージョン 9.15.1 以降が必要）
- ライセンス：ONTAP SnapMirror データ保護バンドルを使用する非同期ライセンスは、ソースとデスティネーション クラスタの両方で有効にする必要があります。詳細については、 "ONTAP における SnapMirror ライセンスの概要"を参照してください。

  ONTAP 9.10.1以降では、すべてのライセンスがNetAppライセンス ファイル（NLF）として提供されます。これは、複数の機能を有効にする単一のファイルです。詳細については、"ONTAP Oneに含まれるライセンス"を参照してください。

  SnapMirror非同期保護のみがサポートされています。

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予定の場合は、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 バックエンドはピアリングする必要があります。詳細については、 "クラスタとSVMのピアリングの概要"を参照してください。

> 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。

- **NetApp Trident**および**SVM**：ピアリングされたリモートSVMは、デスティネーション クラスタ上のNetApp Tridentで使用できる必要があります。
- 管理されたバックエンド：レプリケーション関係を作成するには、Trident ProtectでONTAPストレージバックエンドを追加して管理する必要があります。

SnapMirror レプリケーション用の Trident / ONTAP 設定

Trident Protect では、ソース クラスタとデスティネーション クラスタの両方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 クラスタが同じ場合、復元力を最大限に高めるには、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異なるストレージ バックエンドを使用する必要があります。

SnapMirror レプリケーションの Kubernetes クラスタの要件

Kubernetes クラスタが次の要件を満たしていることを確認します。

- **AppVault** アクセシビリティ：ソース クラスタとデスティネーション クラスタの両方が、アプリケーション オブジェクトのレプリケーションのために AppVault への読み取りと書き込みを行うためのネットワーク アクセスを持っている必要があります。
- ネットワーク接続：ファイアウォール ルール、バケット権限、IP 許可リストを設定して、両方のクラスタと AppVault 間の WAN 経由の通信を有効にします。

> 多くの企業環境では、WAN 接続全体に厳格なファイアウォール ポリシーが実装されています。レプリケーションを設定する前に、これらのネットワーク要件をインフラストラクチャ チームに確認してください。

Trident Protectのインストールと設定

環境がTrident Protect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 Protectをインストールできます。Trident ProtectはNetAppから入手するか、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。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 レジストリからインストールすると便利です。

Trident Protectのインストール

NetAppからTrident Protectをインストールする

手順

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. Helmを使用してTrident Protectをインストールします。 `<name-of-cluster>`をクラスタ名に置き換えます。このクラスタ名はクラスタに割り当てられ、クラスタのバックアップとSnapshotを識別するために使用されます：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2510.0 --create
-namespace --namespace trident-protect
```

3. オプションで、デバッグ ログを有効にするには（トラブルシューティングに推奨）、次のコマンドを使用します：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2510.0 --create-namespace --namespace trident-protect
```

デバッグログは、ログ レベルの変更や問題の再現を必要とせずに、NetApp サポートが問題のトラブルシューティングを行うのに役立ちます。

プライベートレジストリから**Trident Protect**をインストールする

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。これらの例では、括弧内の値を環境の情報に置き換えます：

手順

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：

```
docker.io/netapp/controller:25.10.0
docker.io/netapp/restic:25.10.0
docker.io/netapp/kopia:25.10.0
docker.io/netapp/kopiablockrestore:25.10.0
docker.io/netapp/trident-autosupport:25.10.0
docker.io/netapp/exechook:25.10.0
docker.io/netapp/resourcebackup:25.10.0
docker.io/netapp/resourcerestore:25.10.0
docker.io/netapp/resourcedelete:25.10.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

例：

```
docker pull docker.io/netapp/controller:25.10.0
```

```
docker tag docker.io/netapp/controller:25.10.0 <private-registry-url>/controller:25.10.0
```

```
docker push <private-registry-url>/controller:25.10.0
```

> Helmチャートを取得するには、まず `helm pull trident-protect --version 100.2510.0 --repo https://netapp.github.io/trident-protect-helm-chart`を使用してインターネットにアクセスできるマシンにHelmチャートをダウンロードし、その結果の `trident-protect-100.2510.0.tgz`ファイルをオフライン環境にコピーして、最後のステップでリポジトリ参照の代わりに `helm install trident-protect ./trident-protect-100.2510.0.tgz`を使用してインストールします。

2. Trident Protect システム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. `protectValues.yaml`という名前のファイルを作成します。次のTrident Protect設定が含まれていることを確認してください：

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

> `imageRegistry`および `imagePullSecrets`の値は、 `resourcebackup`および `resourcerestore`を含むすべてのコンポーネントイメージに適用されます。レジストリ内の特定のリポジトリパスにイメージをプッシュする場合（例：`example.com:443/my-repo`）、レジストリフィールドに完全パスを含めてください。これにより、すべてのイメージが`<private-registry-url>/<image-name>:<tag>`から取得されます。

7. Helmを使用してTrident Protectをインストールします。`<name_of_cluster>`をクラスタ名に置き換えます。このクラスタ名はクラスタに割り当てられ、クラスタのバックアップとSnapshotを識別するために使用されます：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2510.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

8. オプションで、デバッグ ログを有効にするには（トラブルシューティングに推奨）、次のコマンドを使用します：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2510.0 --create-namespace --namespace trident-protect -f
protectValues.yaml
```

デバッグログは、ログ レベルの変更や問題の再現を必要とせずに、NetApp サポートが問題のトラブルシューティングを行うのに役立ちます。

AutoSupport設定や名前空間フィルタリングなどの追加のHelmチャート設定オプションについては、"Trident Protectのインストールをカスタマイズする"を参照してください。

Trident Protect CLI プラグインをインストールします

Trident Protect コマンドライン プラグインを使用できます。これは Trident `tridentctl` ユーティリティの拡張機能であり、Trident Protect カスタム リソース（CR）の作成と操作に使用できます。

Trident Protect CLI プラグインをインストールします

コマンドライン ユーティリティを使用する前に、クラスターにアクセスするために使用するマシンにそれをインストールする必要があります。マシンが x64 と ARM のどちらの CPU を使用しているかに応じて、次の手順に従ってください。

Linux AMD64 CPU 用プラグインをダウンロード

手順

1. Trident Protect 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/25.10.0/tridentctl-protect-linux-amd64
```

Linux ARM64 CPU用プラグインをダウンロード

手順

1. Trident Protect 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/25.10.0/tridentctl-protect-linux-arm64
```

Mac AMD64 CPU用プラグインをダウンロード

手順

1. Trident Protect 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/25.10.0/tridentctl-protect-macos-amd64
```

Mac ARM64 CPU用プラグインをダウンロード

手順

1. Trident Protect 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/25.10.0/tridentctl-protect-macos-arm64
```

1. プラグイン バイナリの実行権限を有効にします：

```
chmod +x tridentctl-protect
```

2. プラグインのバイナリを PATH 変数で定義されている場所にコピーします。例えば、 `/usr/bin`または `/usr/local/bin`（昇格した権限が必要になる場合があります）：

```
cp ./tridentctl-protect /usr/local/bin/
```

3. オプションで、プラグインのバイナリをホームディレクトリ内の場所にコピーすることもできます。この場合、場所が PATH 変数の一部であることを確認することをお勧めします：

```
cp ./tridentctl-protect ~/bin/
```

プラグインを PATH 変数の場所にコピーすると、 `tridentctl-protect`または `tridentctl protect`と入力することで、どこからでもプラグインを使用できるようになります。

Trident CLI プラグインのヘルプを表示

プラグインの機能に関する詳細なヘルプを取得するには、組み込みのプラグイン ヘルプ機能を使用できます：

手順

1. ヘルプ機能を使用して使用方法の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 Protect CLI プラグインをインストールしたら、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

手順

1. 完了スクリプトを作成します：

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. スクリプトを格納するための新しいディレクトリをホーム ディレクトリに作成します：

```
mkdir -p ~/.bash/completions
```

3. ダウンロードしたスクリプトを `~/.bash/completions`ディレクトリに移動します：

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. ホームディレクトリ内の `~/.bashrc`ファイルに次の行を追加します：

```
source ~/.bash/completions/tridentctl-completion.bash
```

Z シェルの自動補完を有効にする

手順

1. 完了スクリプトを作成します：

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. スクリプトを格納するための新しいディレクトリをホーム ディレクトリに作成します：

```
mkdir -p ~/.zsh/completions
```

3. ダウンロードしたスクリプトを `~/.zsh/completions`ディレクトリに移動します：

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. ホームディレクトリ内の `~/.zprofile`ファイルに次の行を追加します：

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

結果

次回のシェルログイン時には、tridentctl-protect プラグインによるコマンドの自動補完を使用できます。

Trident Protectのインストールをカスタマイズする

環境の特定の要件を満たすように Trident Protect のデフォルト設定をカスタマイズできます。

Trident Protectコンテナのリソース制限を指定する

Trident Protectのインストール後、設定ファイルを使用してTrident Protectコンテナのリソース制限を指定できます。リソース制限を設定すると、Trident Protect操作によって消費されるクラスタのリソース量を制御できます。

手順

1. `resourceLimits.yaml`という名前のファイルを作成します。
2. 環境のニーズに応じて、Trident Protectコンテナのリソース制限オプションをファイルに入力します。

   次の構成ファイルの例は、使用可能な設定を示しており、各リソース制限のデフォルト値が含まれています：

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

```
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. `resourceLimits.yaml`ファイルから値を適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

セキュリティコンテキスト制約をカスタマイズする

Trident Protect のインストール後、設定ファイルを使用して Trident Protect コンテナの OpenShift セキュリティコンテキスト制約（SCC）を変更できます。これらの制約は、Red Hat OpenShift クラスタ内のポッドのセキュリティ制限を定義します。

手順

1. `sccconfig.yaml`という名前のファイルを作成します。
2. ファイルに SCC オプションを追加し、環境のニーズに応じてパラメータを変更します。

   次の例は、SCC オプションのパラメータのデフォルト値を示しています：

```
scc:
  create: true
  name: trident-protect-job
  priority: 1
```

この表は、SCC オプションのパラメータについて説明しています：

パラメータ	概要	デフォルト
作成する	SCC リソースを作成できるかどうかを決定します。SCC リソースは、 `scc.create`が `true`に設定されていて、Helm インストールプロセスがOpenShift環境を識別した場合にのみ作成されます。OpenShiftで動作していない場合、または `scc.create`が `false`に設定されている場合、SCC リソースは作成されません。	true
名前	SCCの名前を指定します。	trident-protect-job
優先度	SCC の優先度を定義します。優先度の値が高い SCC は、値が低い SCC よりも先に評価されます。	1

3. `sccconfig.yaml`ファイルから値を適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-protect/trident-protect -f sccconfig.yaml --reuse-values
```

これにより、デフォルト値が `sccconfig.yaml`ファイルで指定された値に置き換えられます。

追加の**Trident Protect**ヘルムチャート設定を構成する

AutoSupportの設定と名前空間フィルタリングをカスタマイズして、特定の要件を満たすことができます。次の表に、使用可能な設定パラメータを示します：

パラメータ	タイプ	概要
autoSupport.proxy	string	NetApp AutoSupport接続のプロキシURLを設定します。これを使用して、プロキシ サーバ経由でサポートバンドルのアップロードをルーティングします。例：`http://my.proxy.url`。

パラメータ	タイプ	概要
autoSupport.insecure	ブーリアン	`true`に設定すると、`AutoSupport`プロキシ接続の`TLS`検証をスキップします。安全でないプロキシ接続にのみ使用してください。（デフォルト：`false`）
autoSupport.enabled	ブーリアン	Trident Protect AutoSupportバンドルの日次アップロードを有効または無効にします。 `false`に設定すると、スケジュールされた日次アップロードは無効になりますが、サポート バンドルを手動で生成することはできます。（デフォルト： `true`）
restoreSkipNamespaceAnnotations	string	バックアップおよびリストア処理から除外するネームスペースアノテーションのカンマ区切りリスト。アノテーションに基づいてネームスペースをフィルタリングできます。
restoreSkipNamespaceLabels	string	バックアップおよび復元操作から除外する名前空間ラベルのコンマ区切りリスト。ラベルに基づいて名前空間をフィルタリングできます。

これらのオプションは、YAML 構成ファイルまたはコマンドライン フラグを使用して設定できます：

YAMLファイルを使用する

手順

1. 設定ファイルを作成し、 `values.yaml`という名前を付けます。
2. 作成したファイルに、カスタマイズする設定オプションを追加します。

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
```

3. `values.yaml`ファイルに正しい値を入力したら、構成ファイルを適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

CLI フラグを使用する

手順

1. `--set`フラグを指定して次のコマンドを使用し、個々のパラメータを指定します。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set-string
restoreSkipNamespaceAnnotations="{annotation1,annotation2}" \
  --set-string restoreSkipNamespaceLabels="{label1,label2}" \
  --reuse-values
```

特定のノードへの**Trident Protect**ポッドの制限

Kubernetes の nodeSelector ノード選択制約を使用して、ノードラベルに基づいて Trident Protect ポッドを実行できるノードを制御できます。デフォルトでは、Trident Protect は Linux を実行しているノードに制限されます。ニーズに応じてこれらの制約をさらにカスタマイズできます。

手順

1. `nodeSelectorConfig.yaml`という名前のファイルを作成します。
2. nodeSelectorオプションをファイルに追加し、環境のニーズに応じて制限するノード ラベルを追加または変更するようにファイルを修正します。たとえば、次のファイルにはデフォルトの OS 制限が含まれてい

ますが、特定の地域とアプリ名もターゲットにしています：

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. `nodeSelectorConfig.yaml`ファイルから値を適用します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

これにより、デフォルトの制限が、 `nodeSelectorConfig.yaml`ファイルで指定した制限に置き換えられます。

Trident Protectの管理

Trident Protectの認証とアクセス制御を管理する

Trident Protectは、Kubernetesモデルのロールベースアクセス制御（RBAC）を使用します。デフォルトでは、Trident Protect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。多数のユーザーを抱える組織や特定のセキュリティニーズを持つ組織の場合は、Trident Protectのロールベースアクセス制御機能を使用して、リソースと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default `trident-protect`ネームスペース内のリソースにアクセスでき、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、追加のネームスペースを作成し、それらのネームスペース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 `trident-protect`ネームスペースでは、どのユーザーもアプリケーションデータ管理CRを作成できないことに注意してください。アプリケーションネームスペースにアプリケーションデータ管理CRを作成する必要があります（ベストプラクティスとして、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アプリケーションデータ管理CRを作成します）。

特権を持つTrident Protectカスタムリソースオブジェクトへのアクセスは、管理者のみが持つ必要があります。これには次のものが含まれます：

- **AppVault**：バケット認証情報データが必要です
- **AutoSupportBundle**：メトリクス、ログ、およびその他の機密性の高いTrident Protect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

ベストプラクティスとして、RBAC を使用して、特権オブジェクトへのアクセスを管理者に制限します。

RBACがリソースと名前空間へのアクセスをどのように制御するかの詳細については、 "Kubernetes RBAC ドキュメント"を参照してください。

サービスアカウントの詳細については、 "Kubernetes サービスアカウントのドキュメント"を参照してください。

例：**2**つのユーザーグループのアクセスを管理する

たとえば、組織にはクラスタ管理者、エンジニアリングユーザーのグループ、マーケティングユーザーのグループがあります。クラスタ管理者は、エンジニアリンググループとマーケティンググループがそれぞれの名前空間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するために、次のタスクを実行します（：）

ステップ**1**：各グループのリソースを格納する名前空間を作成する

名前空間を作成すると、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ーをより適切に制御できるようになります。

手順

1. エンジニアリンググループの名前空間を作成します：

```
kubectl create ns engineering-ns
```

2. マーケティンググループのネームスペースを作成します：

```
kubectl create ns marketing-ns
```

ステップ**2**：各名前空間内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

作成する新しいネームスペース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ますが、将来必要に応じてグループ間で権限をさらに分割できるように、ユーザー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. エンジニアリ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: 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、侵害された場合には簡単に削除して再作成できます。

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. marketing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
```

ステップ**4**：**RoleBinding**オブジェクトを作成して、**ClusterRole**オブジェクトを各新しいサービスアカウントにバインドします

Trident Protect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
```

ステップ**5**：権限をテストする

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
```

ステップ**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、ユーザーにAppVaultへのアクセス権を付与します。たとえば、次のCRは、ユーザー `eng-user`にAppVaultへのアクセス権を付与します：

```
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. ロール CR を作成して適用し、クラスター管理者が名前空間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、ユーザー eng-user に権限をバインドします。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てください。

   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. ユーザーがアクセス権限を持つようになったAppVault情報を取得できるかどうかをテストします：

```
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、承認されたAppVaultオブジェクトをアプリケーションデータ管理操作に使用できる必要があり、割り当てられた名前空間外のリソースにアクセスしたり、アクセス権の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。

Trident Protectリソースを監視

kube-state-metrics、Prometheus、およびAlertmanagerオープンソースツールを使用して、Trident Protectで保護されているリソースの健全性を監視できます。

kube-state-metricsサービスは、Kubernetes API通信からメトリックを生成します。Trident Protectと併用することで、環境内のリソースの状態に関する有用な情報が公開されます。

Prometheus は、kube-state-metrics によって生成されたデータを取り込み、これらのオブジェクトに関する読みやすい情報として提示できるツールキットです。kube-state-metrics と Prometheus を組み合わせることで、Trident Protect で管理しているリソースの健全性とステータスを監視できます。

Alertmanager は、Prometheus などのツールによって送信されたアラートを取り込み、設定した宛先にルーティングするサービスです。

> これらの手順に含まれる構成とガイダンスは単なる例であり、環境に合わせてカスタマイズする必要があります。具体的な手順とサポートについては、次の公式ドキュメントを参照してください：
>
> - "kube-state-metricsのドキュメント"
> - "Prometheusのドキュメント"
> - "Alertmanager ドキュメント"

ステップ**1**：監視ツールをインストールする

Trident Protect でリソース監視を有効にするには、kube-state-metrics、Prometheus、および Alertmanager をインストールして構成する必要があります。

kube-state-metricsをインストールする

Helm を使用して kube-state-metrics をインストールできます。

手順

1. kube-state-metrics Helm チャートを追加します。例：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. Prometheus ServiceMonitor CRD をクラスタに適用します：

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. Helmチャートの設定ファイルを作成します（例： `metrics-config.yaml`）。次の例の構成を、環境に合わせてカスタマイズできます：

metrics-config.yaml：kube-state-metrics Helm チャート設定

```
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: protect.trident.netapp.io
          kind: "Backup"
          version: "v1"
        labelsFromPath:
          backup_uid: [metadata, uid]
          backup_name: [metadata, name]
          creation_time: [metadata, creationTimestamp]
        metrics:
        - name: backup_info
          help: "Exposes details about the Backup state"
          each:
            type: Info
            info:
              labelsFromPath:
                appVaultReference: ["spec", "appVaultRef"]
                appReference: ["spec", "applicationRef"]
rbac:
  extraRules:
  - apiGroups: ["protect.trident.netapp.io"]
    resources: ["backups"]
    verbs: ["list", "watch"]

# Collect metrics from all namespaces
namespaces: ""

# Ensure that the metrics are collected by Prometheus
prometheus:
  monitor:
    enabled: true
```

4. Helm チャートをデプロイして kube-state-metrics をインストールします。例：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. "kube-state-metricsカスタムリソースのドキュメント"の手順に従って、Trident Protectで使用されるカスタムリソースのメトリックを生成するようにkube-state-metricsを設定します。

Prometheusのインストール

Prometheusは、 "Prometheusのドキュメント"の手順に従ってインストールできます。

Alertmanagerをインストールする

Alertmanagerは、 "Alertmanager ドキュメント"の手順に従ってインストールできます。

ステップ2：監視ツールを連携するように設定する

監視ツールをインストールした後、それらが連携するように構成する必要があります。

手順

1. kube-state-metricsをPrometheusと統合します。Prometheus設定ファイル(`prometheus.yaml`）を編集し、kube-state-metricsサービス情報を追加します。例：

prometheus.yaml：kube-state-metrics サービスと **Prometheus** の統合

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: trident-protect
data:
  prometheus.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.trident-protect.svc:8080']
```

2. アラートを Alertmanager にルーティングするように Prometheus を構成します。Prometheus 設定ファイル(`prometheus.yaml`を編集し、次のセクションを追加します：

prometheus.yaml：**Alertmanager**にアラートを送信する

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

結果

Prometheus は kube-state-metrics からメトリクスを収集し、Alertmanager にアラートを送信できるようになりました。これで、アラートをトリガーする条件とアラートの送信先を設定する準備が整いました。

ステップ**3**：アラートとアラートの送信先を設定する

ツールが連携するように構成したら、アラートをトリガーする情報の種類とアラートの送信先を構成する必要があります。

アラートの例：バックアップ失敗

次の例では、バックアップカスタムリソースのステータスが `Error`に設定されてから5秒以上経過したときにトリガーされる重要なアラートを定義します。この例を環境に合わせてカスタマイズし、このYAMLスニペットを `prometheus.yaml`設定ファイルに含めることができます：

rules.yaml：失敗したバックアップに対する **Prometheus** アラートを定義する

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

Alertmanager を設定して他のチャネルにアラートを送信する

Alertmanagerを設定して、他のチャネルに通知を送信することもできます。Eメール、PagerDuty、Microsoft Teams、またはその他の通知サービスを設定するには、 `alertmanager.yaml`ファイルでそれぞれの設定を指定します。

次の例では、Slack チャネルに通知を送信するように Alertmanager を構成します。この例を自分の環境に合わせてカスタマイズするには、 `api_url`キーの値をお使いの環境で使用されている Slack Webhook URL に置き換えます：

alertmanager.yaml：Slackチャンネルにアラートを送信する

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

Trident Protect サポートバンドルを生成する

Trident Protectは、管理者がログ、メトリック、管理下のクラスタやアプリに関するトポロジ情報など、NetApp Supportに役立つ情報を含むバンドルを生成できるようにします。インターネットに接続されている場合、カスタムリソース（CR）ファイルを使用して、サポートバンドルをNetApp Support Site（NSS）にアップロードできます。

CR を使用してサポートバンドルを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、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**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.triggerType**：（必須）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決定します。スケジュールされたバンドル生成は、UTC の午前 12 時に行われます。可能な値：
     - スケジュール済み
     - 手動
   - **spec.uploadEnabled**：（オプション）サポートバンドルの生成後にNetApp Support Siteにアップロードするかどうかを制御します。指定されていない場合、デフォルトは `false`です。使用可能な値：
     - true
     - false （デフォルト）
   - **spec.dataWindowStart**：（オプション）サポート バンドルに含まれるデータのウィンドウの開始日時を指定する 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ウィンドウ日付は 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. `trident-protect-support-bundle.yaml`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-support-bundle.yaml -n trident-protect
```

CLI を使用してサポートバンドルを作成する

手順

1. 括弧内の値を環境の情報に置き換えて、サポート バンドルを作成します。 `trigger-type`は、バンド

ルがすぐに作成されるか、作成時間がスケジュールによって決定されるかを決定します。`Manual`または `Scheduled`を指定できます。デフォルト設定は `Manual`です。

例：

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

サポートバンドルを監視して取得する

いずれかの方法を使用してサポート バンドルを作成した後、その生成の進行状況を監視し、ローカル システムに取得することができます。

手順

1. `status.generationState`が `Completed`状態になるまで待ちます。次のコマンドで生成の進行状況を監視できます：

```
kubectl get autosupportbundle trident-protect-support-bundle -n trident-protect
```

2. サポート バンドルをローカル システムに取得します。完了したAutoSupportバンドルからコピー コマンドを取得します：

```
kubectl describe autosupportbundle trident-protect-support-bundle -n trident-protect
```

出力から `kubectl cp`コマンドを見つけて実行し、宛先引数を希望のローカル ディレクトリに置き換えます。

Trident Protectのアップグレード

新しい機能やバグ修正を活用するには、Trident Protectを最新バージョンにアップグレードできます。

- バージョン24.10からアップグレードすると、アップグレード中に実行されるスナップショットが失敗する可能性があります。この障害により、手動またはスケジュールされた将来のスナップショットの作成が妨げられることはありません。アップグレード中にスナップショットが失敗した場合は、アプリケーションが保護されるように手動で新しいスナップショットを作成できます。

  潜在的な障害を回避するために、アップグレード前にすべてのスナップショット スケジュールを無効にして、アップグレード後に再度有効にすることができます。ただし、これにより、アップグレード期間中にスケジュールされたスナップショットが失われることになります。

- プライベート レジストリのインストールでは、ターゲット バージョンに必要な Helm チャートとイメージがプライベート レジストリで使用できることを確認し、カスタム Helm 値が新しいチャート バージョンと互換性があることを確認します。詳細については、"プライベートレジストリからTrident Protectをインストールする"を参照してください。

Trident Protect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. Trident Helm リポジトリを更新します：

```
helm repo update
```

2. Trident Protect CRD をアップグレードする：

   25.06より前のバージョンからアップグレードする場合は、CRDがTrident Protect Helmチャートに含まれているため、この手順が必要です。

   a. このコマンドを実行して、CRDの管理を `trident-protect-crds`から `trident-protect`に移行します：

```
kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
{"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
```

   b. このコマンドを実行して `trident-protect-crds`チャートのHelmシークレットを削除します：

   Helm を使用して `trident-protect-crds`チャートをアンインストールしないでください。CRD と関連データが削除される可能性があります。

```
kubectl delete secret -n trident-protect -l name=trident-protect-
crds,owner=helm
```

3. Trident Protectのアップグレード：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2510.0 --namespace trident-protect
```

アップグレード中にログレベルを設定するには、アップグレード コマンドに `--set logLevel=debug`を追加します。デフォルトのログレベルは `warn`です。デバッグログは、ログ レベルの変更や問題の再現を必要とせずにNetAppサポートが問題を診断するのに役立つため、トラブルシューティングに推奨されます。

アプリケーションの管理と保護

Trident Protect AppVaultオブジェクトを使用してバケットを管理する

Trident Protectのバケットカスタムリソース（CR）は、AppVaultとして知られています。AppVaultオブジェクトは、ストレージバケットの宣言的な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の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。管理者のみがAppVaultsを作成できます。

アプリケーションでデータ保護操作を実行するときは、AppVault CRを手動またはコマンドラインから作成する必要があります。AppVault CRは環境に固有のものであり、このページの例をAppVault CRを作成する際のガイドとして使用できます。

AppVault CRがTrident Protectがインストールされているクラスタ上にあることを確認してください。AppVault CRが存在しないか、アクセスできない場合は、コマンドラインにエラーが表示されます。

AppVault認証とパスワードを設定

AppVault CRを作成する前に、AppVaultと選択したデータムーバーがプロバイダおよび関連リソースで認証できることを確認してください。

データムーバーリポジトリのパスワード

CRやTrident Protect CLIプラグインを使用してAppVaultオブジェクトを作成する際、ResticおよびKopia暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。シークレットを指定しない場合、Trident Protectはデフォルトのパスワードを使用します。

- AppVault CR を手動で作成する場合は、**spec.dataMoverPasswordSecretRef** フィールドを使用してシークレットを指定します。
- Trident Protect CLIを使用してAppVaultオブジェクトを作成する場合は、 `--data-mover-password-secret-ref`引数を使用してシークレットを指定します。

データムーバーリポジトリのパスワードシークレットを作成する

次の例を使用して、パスワードシークレットを作成します。AppVaultオブジェクトを作成する際、Trident Protectにこのシークレットを使用してデータムーバーリポジトリで認証するように指示できます。

- 使用しているデータムーバーに応じて、そのデータムーバーに対応するパスワードのみを含める必要があります。たとえば、Resticを使用しており、将来Kopiaを使用する予定がない場合は、シークレットを作成するときにResticパスワードのみを含めることができます。
- パスワードは安全な場所に保管してください。同じクラスタまたは別のクラスタでデータをリストアするには、これが必要になります。クラスタまたは `trident-protect`ネームスペースが削除されると、パスワードなしでバックアップやSnapshotをリストアできなくなります。

CRを使用する

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

CLI を使用する

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

S3 互換ストレージの IAM 権限

Trident Protectを使用してAmazon S3、Generic S3、 "StorageGrid S3"、または "ONTAP S3"などのS3互換ストレージにアクセスする場合は、指定するユーザクレデンシャルにバケットへのアクセスに必要な権限があることを確認する必要があります。次に、Trident Protectでアクセスするために必要な最小限の権限を付与するポリシーの例を示します。このポリシーは、S3互換バケットポリシーを管理するユーザに適用できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon S3 ポリシーの詳細については、 "Amazon S3 ドキュメント"の例を参照してください。

Amazon S3（AWS）認証用のEKS Pod Identity

Trident Protect は、Kopia データ ムーバー操作用の EKS Pod Identity をサポートします。この機能により、Kubernetes シークレットに AWS 認証情報を保存せずに、S3 バケットへの安全なアクセスが可能になります。

Trident Protect を使用した **EKS Pod Identity** の要件

EKS Pod Identity を Trident Protect で使用する前に、次の点を確認してください：

- EKS クラスタで Pod Identity が有効になっています。
- 必要な S3 バケット権限を持つ IAM ロールを作成しました。詳細については、"S3 互換ストレージの IAM 権限"を参照してください。
- IAM ロールは次の Trident Protect サービス アカウントに関連付けられています：
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

Pod Identity を有効にして IAM ロールをサービスアカウントに関連付ける詳細な手順については、 "AWS EKS Pod Identity のドキュメント"を参照してください。

AppVault設定 EKS Pod Identityを使用する場合は、明示的なクレデンシャルの代わりに `useIAM: true`フラグを使用してAppVault CRを設定します：

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: eks-protect-vault
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-aws
      endpoint: s3.example.com
      useIAM: true
```

AppVault クラウド プロバイダ向けのキー生成例

AppVault CR を定義する際には、IAM 認証を使用していない限り、プロバイダによってホストされているリソースにアクセスするための認証情報を含める必要があります。認証情報のキーを生成する方法は、プロバイダによって異なります。以下は、いくつかのプロバイダのコマンドライン キー生成の例です。次の例を使用して、各クラウド プロバイダの認証情報のキーを作成できます。

Google Cloud

```
kubectl create secret generic <secret-name> \
--from-file=credentials=<mycreds-file.json> \
-n trident-protect
```

Amazon S3（AWS）

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<amazon-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

Microsoft Azure

```
kubectl create secret generic <secret-name> \
--from-literal=accountKey=<secret-name> \
-n trident-protect
```

汎用 S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<generic-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

ONTAP S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<ontap-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

StorageGrid S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<storagegrid-s3-trident-protect-src
-bucket-secret> \
-n trident-protect
```

AppVault 作成例

以下は、各プロバイダーの AppVault 定義の例です。

AppVault CR の例

以下のCRの例を使用して、各クラウド プロバイダのAppVaultオブジェクトを作成できます。

- オプションで、Restic および Kopia リポジトリの暗号化用のカスタム パスワードを含む Kubernetes シークレットを指定できます。詳細については、[データムーバーリポジトリのパスワード]を参照してください。
- Amazon S3（AWS）AppVaultオブジェクトの場合、オプションでsessionTokenを指定できます。これは、認証にシングル サインオン（SSO）を使用している場合に便利です。このトークンは、AppVault クラウド プロバイダ向けのキー生成例でプロバイダのキーを生成するときに作成されます。
- S3 AppVaultオブジェクトの場合、 `spec.providerConfig.S3.proxyURL`キーを使用して、アウトバウンドS3トラフィックの出力プロキシURLをオプションで指定できます。

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

Amazon S3（AWS）

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3-secret
```

Kopia データムーバーで Pod Identity を使用している EKS 環境の場合、`providerCredentials` セクションを削除し、代わりに `s3` 設定の下に `useIAM: true` を追加できます。

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

汎用 S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGrid S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

Trident Protect CLI を使用した**AppVault**作成例

次の CLI コマンド例を使用して、各プロバイダーの AppVault CR を作成できます。

- オプションで、Restic および Kopia リポジトリの暗号化用のカスタム パスワードを含む Kubernetes シークレットを指定できます。詳細については、[データムーバーリポジトリのパスワード]を参照してください。
- S3 AppVaultオブジェクトの場合、 `--proxy-url <ip_address:port>`引数を使用して、アウトバウンドS3トラフィックの出力プロキシURLをオプションで指定できます。

Google Cloud

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Amazon S3（AWS）

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

汎用 S3

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

StorageGrid S3

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

サポートされている `**providerConfig.s3**`設定オプション

S3 プロバイダの構成オプションについては、次の表を参照してください：

パラメータ	概要	デフォルト	例
`providerConfig.s3.skipCertValidation`	SSL/TLS 証明書の検証を無効にします。	false	"true"、"false"
`providerConfig.s3.secure`	S3 エンドポイントとの安全な HTTPS 通信を有効にします。	true	"true"、"false"
`providerConfig.s3.proxyURL`	S3 に接続するために使用されるプロキシ サーバの URL を指定します。	なし	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	SSL/TLS 検証用のカスタム ルート CA 証明書を提供します。	なし	"CN=MyCustomCA"
`providerConfig.s3.useIAM`	S3 バケットにアクセスするための IAM 認証を有効にします。EKS Pod Identity に適用可能です。	false	true、false

AppVault 情報を表示します

Trident Protect CLI プラグインを使用して、クラスタ上に作成した 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 \
-n trident-protect
```

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. オプションで、各リソースのAppVaultPathを表示するには、フラグ `--show-paths`を使用します。

表の最初の列のクラスタ名は、Trident Protect Helm のインストール時にクラスタ名が指定された場合にのみ使用できます。例： `--set clusterName=production1`。

AppVault を削除します

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 内の `finalizers`キーを削除しないでください。削除すると、AppVault バケット内に残留データが残り、クラスター内にリソースが孤立する可能性があります。

開始する前に

削除するAppVaultで使用されているすべてのスナップショットとバックアップCRが削除されていることを確認してください。

Kubernetes CLI を使用して **AppVault** を削除する

1. AppVaultオブジェクトを削除します。 `appvault-name`を削除するAppVaultオブジェクトの名前に置き換えます：

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

Trident Protect CLI を使用して **AppVault** を削除する

1. AppVaultオブジェクトを削除します。 `appvault-name`を削除するAppVaultオブジェクトの名前に置き換えます：

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

Trident Protectで管理用のアプリケーションを定義する

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と関連するAppVault CRを作成します。

AppVault CR を作成します

アプリケーションでデータ保護処理を実行する際に使用されるAppVault CRを作成する必要があります。AppVault CRは、Trident Protectがインストールされているクラスタに配置する必要があります。AppVault CRは環境に固有のものです。AppVault CRの例については、"AppVault カスタム リソース。"を参照してください。

アプリケーションを定義する

Trident Protectで管理する各アプリケーションを定義する必要があります。管理対象アプリケーションを定義するには、アプリケーションCRを手動で作成するか、Trident Protect CLIを使用します。

CRを使用してアプリケーションを追加する

手順

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、名前を付けます（例：`maria-app.yaml`）。
   b. 次の属性を設定します：
      - **metadata.name**：（必須）アプリケーションのカスタム リソースの名前。保護操作に必要な他の CR ファイルはこの値を参照するため、選択した名前を書き留めておいてください。
      - **spec.includedNamespaces**：（必須）ネームスペースとラベルセレクタを使用して、アプリケーションが使用するネームスペースとリソースを指定します。アプリケーションネームスペースはこのリストに含まれている必要があります。ラベルセレクタはオプションであり、指定された各ネームスペース内のリソースをフィルタリングするために使用できます。
      - **spec.includedClusterScopedResources**：（オプション）この属性を使用して、アプリケーション定義に含めるクラスタスコープのリソースを指定します。この属性を使用すると、グループ、バージョン、種類、ラベルに基づいてこれらのリソースを選択できます。
        - **groupVersionKind**：（必須）クラスタースコープのリソースの API グループ、バージョン、および種類を指定します。
        - **labelSelector**：（オプション）ラベルに基づいてクラスタースコープのリソースをフィルタリングします。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**：（オプション）このアノテーションは、KubeVirt環境など、スナップショットの前にファイルシステムのフリーズが発生する仮想マシンから定義されたアプリケーションにのみ適用されます。スナップショット中にこのアプリケーションがファイルシステムに書き込むことができるかどうかを指定します。trueに設定すると、アプリケーションはグローバル設定を無視し、スナップショット中にファイルシステムに書き込むことができます。falseに設定すると、アプリケーションはグローバル設定を無視し、スナップショット中にファイルシステムがフリーズされます。指定されていても、アプリケーション定義にアプリケーションの仮想マシンがない場合、アノテーションは無視されます。指定されていない場合、アプリケーションは"グローバルTrident Protect freeze設定"に従います。

アプリケーションがすでに作成された後にこのアノテーションを適用する必要がある場合は、次のコマンドを使用できます：

```
kubectl annotate application -n <application CR namespace> <application CR name> protect.trident.netapp.io/skip-vm-freeze="true"
```

+
YAMLの例：

+

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. （オプション）特定のラベルでマークされたリソースを含めるか除外するかを指定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須）`Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。含めるまたは除外するリソースを定義するには、以下のresourceMatchersパラメータを追加します：

     - **resourceFilter.resourceMatchers**：resourceMatcherオブジェクトの配列。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種類、バージョン）はAND演算として一致します。

       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
       - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。

- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例： `"trident.netapp.io/os=linux"`。

> ⓘ `resourceFilter`と `labelSelector`の両方が使用される場合、`resourceFilter`が最初に実行され、次に `labelSelector`が結果のリソースに適用されます。

例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

2. 環境に合わせてアプリケーション CR を作成したら、CR を適用します。例：

```
kubectl apply -f maria-app.yaml
```

手順

1. 次のいずれかの例を使用してアプリケーション定義を作成し、適用します。括弧内の値は、ご使用の環境の情報に置き換えてください。例に示す引数を含むコンマ区切りリストを使用して、アプリケーション定義に名前空間とリソースを含めることができます。

   アプリを作成するときにオプションで注釈を使用して、スナップショット中にアプリケーションがファイルシステムに書き込むことができるかどうかを指定できます。これは、KubeVirt環境など、スナップショットの前にファイルシステムのフリーズが発生する仮想マシンから定義されたアプリケーションにのみ適用されます。注釈を `true`に設定すると、アプリケーションはグローバル設定を無視し、スナップショット中にファイルシステムに書き込むことができます。 `false`に設定すると、アプ

リケーションはグローバル設定を無視し、スナップショット中にファイルシステムがフリーズされます。アノテーションを使用しても、アプリケーション定義にアプリケーションの仮想マシンがない場合、アノテーションは無視されます。アノテーションを使用しない場合、アプリケーションは"グローバルTrident Protect freeze設定"に従います。

CLIを使用してアプリケーションを作成するときにアノテーションを指定するには、`--annotation`フラグを使用できます。

- アプリケーションを作成し、ファイルシステムのフリーズ動作のグローバル設定を使用します：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace>
```

- アプリケーションを作成し、ファイルシステムのフリーズ動作のローカル アプリケーション設定を構成します：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --annotation protect.trident.netapp.io/skip-vm-freeze
=<"true"|"false">
```

`--resource-filter-include`および `--resource-filter-exclude`フラグを使用して、グループ、種類、バージョン、ラベル、名前、名前空間などの`resourceSelectionCriteria`に基づいてリソースを含めるか除外するかを指定できます。次に例を示します。

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-namespace>
--resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my-
deployment"],"Namespaces":["my-
namespace"],"LabelSelectors":["app=my-app"]}]'
```

Trident Protectを使用してアプリケーションを保護

Trident Protectで管理されているすべてのアプリを、自動化された保護ポリシーまたはアドホックベースでスナップショットとバックアップを作成して保護できます。

Trident Protectを設定して、データ保護操作中にファイルシステムをフリーズおよびフリーズ解除することができます。"Trident Protectを使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください".

オンデマンドスナップショットを作成する

オンデマンド スナップショットはいつでも作成できます。

クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CR を使用してスナップショットを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **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**：（必須）スナップショットの内容（メタデータ）を保存するAppVaultの名前。
   - **spec.reclaimPolicy**：（オプション）スナップショット CR が削除されたときに、スナップショットの AppArchive に対して何が起こるかを定義します。つまり、 `Retain`に設定されている場合でも、スナップショットは削除されます。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. `trident-protect-snapshot-cr.yaml`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-snapshot-cr.yaml
```

CLI を使用してスナップショットを作成する

手順

1. 括弧内の値を環境の情報に置き換えてスナップショットを作成します。例：

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

オンデマンド バックアップを作成する

アプリはいつでもバックアップできます。

> クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 バックアップ処理に十分であることを確認します。バックアップ処理中にトークンの有効期限が切れると、処理が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

CR を使用してバックアップを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.applicationRef**：（必須）バックアップするアプリケーションの Kubernetes 名。
   - **spec.appVaultRef**：（必須）バックアップコンテンツを保存するAppVaultの名前。
   - **spec.dataMover**：（オプション）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。可能な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia`（デフォルト）
   - **spec.reclaimPolicy**：（オプション）クレームから解放されたときにバックアップに何が起こるかを定義します。可能な値：
     - `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スナップショットの名前。指定しない場合は、一時ボリュームが作成され、バックアップされます。

     YAMLの例：

```
---
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. `trident-protect-backup-cr.yaml`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-backup-cr.yaml
```

CLIを使用してバックアップを作成する

手順

1. 括弧内の値を環境の情報に置き換えてバックアップを作成します。例：

```
tridentctl-protect create backup <my_backup_name> --appvault <my-vault-name> --app <name_of_app_to_back_up> --data-mover <Kopia_or_Restic> -n <application_namespace>
```

オプションで `--full-backup`フラグを使用して、バックアップを非増分にするかどうかを指定できます。デフォルトでは、すべてのバックアップは増分バックアップになります。このフラグを使用すると、バックアップは非増分になります。ベストプラクティスとしては、定期的にフル バックアップを実行し、その間に増分バックアップを実行して、リストアに伴うリスクを最小限に抑えます。

サポートされているバックアップアノテーション

次の表は、バックアップ CR を作成するときに使用できる注釈を示しています：

注釈	タイプ	概要	デフォルト値
protect.trident.netapp.io/full-backup	string	バックアップを非増分にするかどうかを指定します。 `true`に設定すると、非増分バックアップが作成されます。ベストプラクティスとしては、定期的にフル バックアップを実行し、フル バックアップの間に増分バックアップを実行して、リストアに伴うリスクを最小限に抑えることです。	"false"
protect.trident.netapp.io/snapshot-completion-timeout	string	スナップショット処理全体が完了するまでに許容される最大時間。	「60m」
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	"30分"
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	string	ボリューム スナップショットの作成に許可される最大時間。	「5m」
protect.trident.netapp.io/pvc-bind-timeout-sec	string	新しく作成されたPersistentVolumeClaims（PVC）が `Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。	「1200」（20分）

データ保護スケジュールを作成する

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してアプリを保護します。スナップショットとバックアップを時間ごと、日ごと、週ごと、月ごとに作成するように選択でき、保持するコピーの数を指定できます。full-backup-rule アノテーションを使用して、非増分フル バックアップをスケジュールできます。デフォルトでは、すべてのバックアップは増分バックアップになります。フル バックアップを定期的に実行し、その間に増分バックアップを実行することで、復元に伴うリスクを軽減できます。

- スナップショットのみのスケジュールを作成するには、 `backupRetention`をゼロに設定し、 `snapshotRetention`をゼロより大きい値に設定します。 `snapshotRetention`をゼロに設定すると、スケジュールされたバックアップでは引き続きスナップショットが作成されますが、これらは一時ボリュームであり、バックアップが完了するとすぐに削除されます。
- クラスタを対象としたリソースは、アプリケーション定義で明示的に参照されている場合、またはいずれかのアプリケーション名前空間への参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CRを使用してスケジュールを作成する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-schedule-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.dataMover**：（オプション）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。可能な値（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**：（必須）バックアップコンテンツを保存するAppVaultの名前。
   - **spec.backupRetention**：（必須）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（スナップショットのみ）。
   - **spec.backupReclaimPolicy**：（オプション）保持期間中にバックアップ CR が削除された場合に、バックアップに何が起こるかを決定します。保持期間が過ぎると、バックアップは常に削除されます。可能な値（大文字と小文字が区別されます）：
     - `Retain`（デフォルト）
     - `Delete`
   - **spec.snapshotRetention**：（必須）保持するスナップショットの数。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.snapshotReclaimPolicy**：（オプション）保持期間中にスナップショット CR が削除された場合にスナップショットに何が起こるかを決定します。保持期間が過ぎると、スナップショットは常に削除されます。可能な値（大文字と小文字が区別されます）：
     - `Retain`
     - `Delete`（デフォルト）
   - **spec.granularity**：スケジュールを実行する頻度。可能な値と、必須の関連フィールド：
     - `Hourly`（ `spec.minute`を指定する必要があります）
     - `Daily`（ `spec.minute`と `spec.hour`を指定する必要があります）
     - `Weekly`（ `spec.minute`, `spec.hour`、 `spec.dayOfWeek`を指定する必要があります）
     - `Monthly`（ `spec.minute`, `spec.hour`、 `spec.dayOfMonth`を指定する必要があります）
     - `Custom`
   - **spec.dayOfMonth**：（オプション）スケジュールを実行する月の日付（1 - 31）。粒度が `Monthly`に設定されている場合、このフィールドは必須です。値は文字列として提供する必要があります。
   - **spec.dayOfWeek**：（オプション）スケジュールを実行する曜日（0 - 7）。値 0 または 7 は日曜日を示します。粒度が `Weekly`に設定されている場合、このフィールドは必須です。値は文字列

として提供する必要があります。

- **spec.hour**: (オプション) スケジュールを実行する時刻（0 - 23）。粒度が `Daily`、 `Weekly`、または `Monthly`に設定されている場合、このフィールドは必須です。値は文字列として提供する必要があります。
- **spec.minute**：（オプション）スケジュールを実行する時刻の分（0～59）。このフィールドは、粒度が `Hourly`、 `Daily`、 `Weekly`、または `Monthly`に設定されている場合は必須です。値は文字列として指定する必要があります。

バックアップとスナップショットのスケジュールの YAML の例：

```
---
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: Daily
  hour: "0"
  minute: "0"
```

スナップショットのみのスケジュールの YAML の例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

3. `trident-protect-schedule-cr.yaml`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-schedule-cr.yaml
```

CLI を使用してスケジュールを作成する

手順

1. 括弧内の値を環境の情報に置き換えて、保護スケジュールを作成します。例：

   `tridentctl-protect create schedule --help`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 <my_schedule_name> \
  --appvault <my_appvault_name> \
  --app <name_of_app_to_snapshot> \
  --backup-retention <how_many_backups_to_retain> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_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-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

   次のフラグを使用すると、スケジュールをさらに制御できます：

   - フル バックアップのスケジュール設定： `--full-backup-rule`フラグを使用して、非増分フル バックアップをスケジュール設定します。このフラグは `--granularity Daily`でのみ機能します。指定可能な値：
     - `Always`：毎日フル バックアップを作成します。
     - 特定の曜日：1つ以上の曜日をカンマで区切って指定します（例： `"Monday,Thursday"`）。有効な値：Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。

       `--full-backup-rule`フラグは、時間単位、週単位、または月単位の粒度では機能しません。

   - スナップショットのみのスケジュール： `--backup-retention 0`を設定し、 `--snapshot-retention`に0より大きい値を指定します。

サポートされているスケジュール注釈

次の表は、スケジュール CR を作成するときに使用できる注釈を示しています：

注釈	タイプ	概要	デフォルト値
protect.trident.netapp.io/full-backup-rule	string	フル バックアップをスケジュールするためのルールを指定します。 `Always`に設定して常時フル バックアップを実行するか、要件に応じてカスタマイズできます。たとえば、日単位の粒度を選択した場合、フル バックアップを実行する曜日を指定できます（例：`"Monday,Thursday"`）。有効な曜日の値は、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sundayです。この注釈は、 `granularity`が `Daily`に設定されているスケジュールでのみ使用できます。	未設定（すべてのバックアップは増分）
protect.trident.netapp.io/snapshot-completion-timeout	string	スナップショット処理全体が完了するまでに許容される最大時間。	「60m」
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	"30分"
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	string	ボリューム スナップショットの作成に許可される最大時間。	「5m」
protect.trident.netapp.io/pvc-bind-timeout-sec	string	新しく作成されたPersistentVolumeClaims（PVC）が `Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。	「1200」（20分）

スナップショットを削除する

不要になったスケジュール済みスナップショットまたはオンデマンド スナップショットを削除します。

手順

1. スナップショットに関連付けられているスナップショット 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する

不要になったスケジュール バックアップまたはオンデマンド バックアップを削除します。

オブジェクト ストレージからすべてのバックアップ データを削除するために、回収ポリシーが `Delete`に設定されていることを確認してください。偶発的なデータ損失を回避するため、ポリシーのデフォルト設定は `Retain`です。ポリシーが `Delete`に変更されない場合、バックアップ データはオブジェクト ストレージに残るため、手動で削除する必要があります。

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
```

バックアップ操作のステータスを確認する

コマンド ラインを使用して、進行中、完了、または失敗したバックアップ操作のステータスを確認できます。

手順

1. バックアップ操作のステータスを取得するには、次のコマンドを使用します。括弧内の値は、ご使用の環境の情報に置き換えてください：

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

azure-netapp-files（ANF）操作のバックアップとリストアを有効にする

Trident Protectをインストールした場合、azure-netapp-filesストレージクラスを使用し、Trident 24.06より前に作成された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ボリュームで動作し、容量プールから追加のスペースを消費しません。

開始する前に

以下を確認してください。

- Trident Protectをインストールしました。
- Trident Protect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。
- `azure-netapp-files`がストレージ バックエンドのデフォルトのストレージ クラスとして選択されています。

設定手順を展開する

1. ANF ボリュームが Trident 24.10 へのアップグレード前に作成された場合は、Trident で以下の手順を実行してください：

   a. azure-netapp-files ベースでアプリケーションに関連付けられている各 PV のスナップショット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 に対してスナップショット 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"
   ```

   +
   スナップショット ディレクトリが有効になっていない場合、Trident Protect は通常のバックアップ機能を選択しますが、これにより、バックアップ プロセス中に容量プールのスペースが一時的に消費されます。この場合、バックアップされるボリュームのサイズの一時ボリュームを作成するために、容量プールに十分なスペースがあることを確認してください。

結果

アプリケーションは、Trident Protectを使用したバックアップとリストアの準備ができています。各PVCは、バックアップとリストアのために他のアプリケーションでも使用できます。

アプリケーションをリストアする

Trident Protectを使用してアプリケーションを復元する

Trident Protectを使用して、スナップショットまたはバックアップからアプリケーションを復元できます。アプリケーションを同じクラスターに復元する場合、既存のスナップショットからの復元の方が高速になります。

- アプリケーションを復元すると、アプリケーションに対して設定されているすべての実行フックもアプリとともに復元されます。復元後の実行フックが存在する場合、復元操作の一部として自動的に実行されます。
- バックアップから別の名前空間または元の名前空間への復元は、qtree ボリュームでサポートされています。ただし、スナップショットから別の名前空間または元の名前空間への復元は、qtree ボリュームではサポートされていません。
- 詳細設定を使用して復元操作をカスタマイズできます。詳細については、"高度な Trident Protect リストア設定を使用する"を参照してください。

バックアップから別の名前空間に復元する

BackupRestore CR を使用して異なる名前空間にバックアップを復元すると、Trident Protect は、アプリケーションを新しい名前空間に復元し、復元されたアプリケーションのアプリケーション CR を作成します。復元されたアプリケーションを保護するには、オンデマンドバックアップまたはスナップショットを作成するか、保護スケジュールを確立します。

- 既存のリソースを含む別の名前空間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、ターゲット名前空間を削除して再作成するか、バックアップを新しい名前空間にリストアします。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （オプション）復元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、特定のラベルでマークされ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：

   Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。たとえば、永続ボリュームクレームリソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は関連付けられているポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。含めるまたは除外するリソースを定義するには、以下のresourceMatchersパラメータを追加します：
     - **resourceFilter.resourceMatchers**：resourceMatcherオブジェクトの配列。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種

類、バージョン）はAND演算として一致します。

- **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
- **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
- **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. `trident-protect-backup-restore-cr.yaml`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-backup-restore-cr.yaml
```

CLI を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、バックアップを別の名前空間に復元します。 `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デスティネーション名前空間にマッピングします（形式は `source1:dest1,source2:dest2`）。例：

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

バックアップから元の名前空間に復元する

いつでもバックアップを元の名前空間に復元できます。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-ipr-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存される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）復元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、特定のラベルでマークされ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：

   Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。たとえば、永続ボリュームクレームリソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は関連付けられているポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。含めるまたは除外するリソースを定義するには、以下のresourceMatchersパラメータを追加します：
     - **resourceFilter.resourceMatchers**：resourceMatcherオブジェクトの配列。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種類、バージョン）はAND演算として一致します。
       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。

- **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション）"Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. `trident-protect-backup-ipr-cr.yaml`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-backup-ipr-cr.yaml
```

CLI を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、バックアップを元の名前空間に復元します。 `backup`引数は、名前空間とバックアップ名を `<namespace>/<name>`の形式で使用します。例：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

元のクラスタに問題がある場合は、バックアップを別のクラスタにリストアできます。

- Kopia をデータ ムーバーとして使用してバックアップを復元する場合、オプションで CR に注釈を指定するか、CLI を使用して Kopia が使用する一時ストレージの動作を制御できます。設定できるオプションの詳細については、 "Kopiaのドキュメント"を参照してください。Trident Protect CLI で注釈を指定する方法の詳細については、 `tridentctl-protect create --help`コマンドを使用してください。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

開始する前に

次の前提条件が満たされていることを確認してください：

- 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。
- デスティネーション クラスタは、バックアップが保存されているソース クラスタと同じAppVaultのバケット パスにアクセスできます。
- ローカル環境が、 `tridentctl-protect get appvaultcontent`コマンドの実行時にAppVault CRで定義されたオブジェクトストレージバケットに接続できることを確認してください。ネットワーク制限によりアクセスできない場合は、代わりにデスティネーション クラスタ上のポッド内からTrident Protect CLIを実行してください。
- AWS セッション トークンの有効期限が、長時間実行される復元操作に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。
  - 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
  - AWS リソースの認証情報の詳細については、 "AWSのドキュメント"を参照してください。

手順

1. Trident Protect CLIプラグインを使用して、デスティネーション クラスタ上のAppVault CRの可用性を確認します：

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

アプリケーションのリストアに使用する名前空間がデスティネーション クラスタに存在することを確認します。

2. デスティネーション クラスタから利用可能なAppVaultのバックアップコンテンツを表示します：

```
tridentctl-protect get appvaultcontent <appvault_name> \
--show-resources backup \
--show-paths \
--context <destination_cluster_name>
```

このコマンドを実行すると、AppVault内の利用可能なバックアップが表示されます。これには、元のクラスタ、対応するアプリケーション名、タイムスタンプ、アーカイブ パスが含まれます。

出力例：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |         TIMESTAMP
|    PATH     |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
```

3. AppVault名前とアーカイブパスを使用して、アプリケーションをデスティネーション クラスタに復元します：

CRを使用する

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-backup-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     BackupRestore CR が利用できない場合は、手順 2 に記載されているコマンドを使用してバックアップの内容を表示できます。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     例：

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-backup-path
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. `trident-protect-backup-restore-cr.yaml`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-backup-restore-cr.yaml
```

CLI を使用する

1. 次のコマンドを使用してアプリケーションを復元し、括弧内の値を環境の情報に置き換えます。namespace-mapping 引数は、コロンで区切られた名前空間を使用して、source1：dest1 、source2：dest2 の形式でソース名前空間を正しい宛先名前空間にマッピングします。例：

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--context <destination_cluster_name> \
-n <application_namespace>
```

スナップショットから別の **namespace** にリストアする

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データを別の名前空間または元のソース名前空間に復元できます。SnapshotRestore CR を使用してスナップショットを別の名前空間に復元すると、Trident Protect は新しい名前空間にアプリケーションを復元し、復元されたアプリケーションのアプリケーション CR を作成します。復元されたアプリケーションを保護するには、オンデマンド バックアップまたはスナップショットを作成するか、保護スケジュールを設定します。

- SnapshotRestoreは `spec.storageClassMapping`属性をサポートしていますが、ソース ストレージ クラスとデスティネーション ストレージ クラスが同じストレージ バックエンドを使用する場合のみです。異なるストレージ バックエンドを使用する `StorageClass`に復元しようとすると、復元処理は失敗します。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

     ```
     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からデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     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"}]
     ```

3. （オプション）復元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、特定のラベルでマークされ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：

   Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。たとえば、永続ボリュームクレームリソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は関連付けられているポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または `Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。含めるまたは除外するリソースを定義するには、以下のresourceMatchersパラメータを追加します：
     - **resourceFilter.resourceMatchers**：resourceMatcherオブジェクトの配列。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種

類、バージョン）はAND演算として一致します。

- **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
- **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
- **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。
- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. `trident-protect-snapshot-restore-cr.yaml`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-snapshot-restore-cr.yaml
```

CLI を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを別のネームスペースにリストアします。

   ◦ その `snapshot`引数は、名前空間とスナップショット名を次の形式で使用します `<namespace>/<name>`。

◦ `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`source1:dest1,source2:dest2`形式で正しいデスティネーション名前空間にマッピングします。

例：

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

スナップショットから元の名前空間に復元する

いつでもスナップショットを元の名前空間にリストアできます。

> アプリケーションが複数の名前空間を使用し、これらの名前空間に同じ名前のPVCがある場合、スナップショットの復元操作（インプレースおよび新しい名前空間への復元の両方）は正しく機能しません。復元されたすべてのボリュームには、名前空間ごとに正しいデータではなく、同じデータが含まれます。スナップショットの復元ではなくバックアップの復元を使用するか、この問題を修正するバージョン26.02以降にアップグレードしてください。

開始する前に

AWS セッショントークンの有効期限が、長時間実行される s3 復元処理に十分であることを確認します。復元操作中にトークンの有効期限が切れると、操作が失敗する可能性があります。

- 現在のセッショントークンの有効期限を確認する方法の詳細については、 "AWS API ドキュメント"を参照してください。
- AWS リソースの認証情報の詳細については、 "AWS IAM ドキュメント"を参照してください。

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-ipr-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

```
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）復元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、特定のラベルでマークされ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：

> Trident Protect は、選択したリソースとの関係に基づいて、一部のリソースを自動的に選択します。たとえば、永続ボリュームクレームリソースを選択し、それに関連付けられたポッドがある場合、Trident Protect は関連付けられているポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**：（フィルタリングに必須） `Include`または`Exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。含めるまたは除外するリソースを定義するには、以下のresourceMatchersパラメータを追加します：
     - **resourceFilter.resourceMatchers**：resourceMatcherオブジェクトの配列。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種類、バージョン）はAND演算として一致します。
       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
       - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョン。

- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. `trident-protect-snapshot-ipr-cr.yaml`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-snapshot-ipr-cr.yaml
```

CLI を使用する

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを元の名前空間に復元します。例：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
-n <application_namespace>
```

リストア処理のステータスを確認する

コマンド ラインを使用して、進行中、完了、または失敗した復元処理のステータスを確認できます。

手順

1. 復元操作のステータスを取得するには、次のコマンドを使用します。括弧内の値は、ご使用の環境の情報に置き換えてください：

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

高度な **Trident Protect** リストア設定を使用する

注釈、名前空間設定、ストレージ オプションなどの詳細設定を使用して、特定の要件を満たすように復元操作をカスタマイズできます。

リストアおよびフェイルオーバー処理中のネームスペースのアノテーションとラベル

復元およびフェイルオーバー処理中に、デスティネーションネームスペースのラベルとアノテーションは、ソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネームスペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、既存のラベルまたはアノテーションはソースネームスペースの値と一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルまたはアノテーションは変更されません。

Red Hat OpenShiftを使用する場合は、OpenShift環境における名前空間アノテーションの重要な役割に注意することが重要です。名前空間アノテーションにより、リストアされたポッドがOpenShiftセキュリティコンテキスト制約（SCC）で定義された適切な権限とセキュリティ設定に準拠し、権限の問題なくボリュームにアクセスできるようになります。詳細については、"OpenShiftセキュリティコンテキスト制約ドキュメント"を参照してください。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`を設定することで、デスティネーションネームスペース内の特定のアノテーションが上書きされるのを防ぐことができます。例：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

復元またはフェイルオーバー操作を実行する場合、 `restoreSkipNamespaceAnnotations`および `restoreSkipNamespaceLabels`で指定されたネームスペースのアノテーションとラベルは、復元またはフェイルオーバー操作から除外されます。これらの設定がHelmの初期インストール時に構成されていることを確認してください。詳細については、"追加のTrident Protectヘルムチャート設定を構成する"を参照してください。

Helm を使用して `--create-namespace`フラグを指定してソースアプリケーションをインストールした場合、`name`ラベルキーには特別な処理が適用されます。リストアまたはフェイルオーバープロセス中に、 Trident Protect はこのラベルをデスティネーションネームスペースにコピーしますが、ソースからの値がソースネームスペースと一致する場合は、値をデスティネーションネームスペースの値に更新します。この値がソースネームスペースと一致しない場合は、変更されずにデスティネーションネームスペースにコピーされます。

例

次の例は、それぞれ異なる注釈とラベルを持つソース名前空間と宛先名前空間を示しています。操作の前後の宛先名前空間の状態や、宛先名前空間で注釈とラベルがどのように結合または上書きされるかを確認できます。

リストアまたはフェイルオーバー処理の前に

次の表は、リストアまたはフェイルオーバー処理前のサンプル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状態を示しています：

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• annotation.one/key："updatedvalue" • annotation.two/key："true"	• environment=production • コンプライアンス＝hipaa • name=ns-1
ネームスペースns-2（宛先）	• annotation.one/key："true" • annotation.three/key："false"	• ロール＝database

リストア処理後

次の表は、復元またはフェイルオーバー操作後の宛先名前空間の例の状態を示しています。いくつかのキーが追加され、いくつかは上書きされ、 `name`ラベルは宛先名前空間と一致するように更新されました：

ネームスペース	アノテーション	ラベル
ネームスペースns-2（宛先）	• annotation.one/key："updatedvalue" • annotation.two/key："true" • annotation.three/key："false"	• name=ns-2 • コンプライアンス＝hipaa • environment=production • ロール＝database

サポートされているフィールド

このセクションでは、復元操作に使用できる追加のフィールドについて説明します。

ストレージクラスのマッピング

`spec.storageClassMapping`属性は、ソース アプリケーションに存在するストレージ クラスからターゲット クラスタ上の新しいストレージ クラスへのマッピングを定義します。異なるストレージ クラスを持つクラスタ間でアプリケーションを移行する場合や、BackupRestore操作のストレージ バックエンドを変更する場合にこれを使用できます。

例：

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

サポートされているアノテーション

このセクションでは、システム内のさまざまな動作を構成するためにサポートされているアノテーションを一覧表示します。ユーザーがアノテーションを明示的に設定しない場合、システムはデフォルト値を使用します。

注釈	タイプ	概要	デフォルト値
protect.trident.netapp.io/data-mover-timeout-sec	string	データ ムーバー処理が停止する最大時間（秒単位）。	"300"
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	string	Kopia コンテンツ キャッシュの最大サイズ制限（メガバイト単位）。	"1000"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	新しく作成されたPersistentVolumeClaims（PVC）が`Bound`フェーズに到達するまで待機する最大時間（秒単位）。この時間を超えると処理は失敗します。環境：すべてのリストアCRタイプ（BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore）。ストレージバックエンドまたはクラスターで処理に時間がかかることが多い場合は、より大きい値を使用してください。	「1200」（20分）

NetApp SnapMirrorとTrident Protectを使用したアプリケーションのレプリケート

Trident Protectを使用すると、NetApp SnapMirrorテクノロジーの非同期レプリケーション機能を使用して、同じクラスタ上または異なるクラスタ間で、あるストレージバックエンドから別のストレージバックエンドにデータとアプリケーションの変更を複製できます。

リストアおよびフェイルオーバー処理中のネームスペースのアノテーションとラベル

復元およびフェイルオーバー処理中に、デスティネーションネームスペースのラベルとアノテーションは、ソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネームスペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、既存のラベルまたはアノテーションはソースネームスペースの値と一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルまたはアノテーションは変更されません。

> Red Hat OpenShiftを使用する場合は、OpenShift環境における名前空間アノテーションの重要な役割に注意することが重要です。名前空間アノテーションにより、リストアされたポッドがOpenShiftセキュリティコンテキスト制約（SCC）で定義された適切な権限とセキュリティ設定に準拠し、権限の問題なくボリュームにアクセスできるようになります。詳細については、"OpenShiftセキュリティコンテキスト制約ドキュメント"を参照してください。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`を設定することで、デスティネーションネームスペース内の特定のアノテーションが上書きされるのを防ぐことができます。例：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

> 復元またはフェイルオーバー操作を実行する場合、 `restoreSkipNamespaceAnnotations`および `restoreSkipNamespaceLabels`で指定されたネームスペースのアノテーションとラベルは、復元またはフェイルオーバー操作から除外されます。これらの設定がHelmの初期インストール時に構成されていることを確認してください。詳細については、"追加のTrident Protectヘルムチャート設定を構成する"を参照してください。

Helm を使用して `--create-namespace`フラグを指定してソースアプリケーションをインストールした場合、 `name`ラベルキーには特別な処理が適用されます。リストアまたはフェイルオーバープロセス中に、 Trident Protect はこのラベルをデスティネーションネームスペースにコピーしますが、ソースからの値がソースネームスペースと一致する場合は、値をデスティネーションネームスペースの値に更新します。この値がソースネームスペースと一致しない場合は、変更されずにデスティネーションネームスペースにコピーされます。

例

次の例は、それぞれ異なる注釈とラベルを持つソース名前空間と宛先名前空間を示しています。操作の前後の宛先名前空間の状態や、宛先名前空間で注釈とラベルがどのように結合または上書きされるかを確認できます。

リストアまたはフェイルオーバー処理の前に

次の表は、リストアまたはフェイルオーバー処理前のサンプル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状態を示しています：

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• annotation.one/key："updatedvalue" • annotation.two/key："true"	• environment=production • コンプライアンス＝hipaa • name=ns-1
ネームスペースns-2（宛先）	• annotation.one/key："true" • annotation.three/key："false"	• ロール＝database

リストア処理後

次の表は、復元またはフェイルオーバー操作後の宛先名前空間の例の状態を示しています。いくつかのキーが追加され、いくつかは上書きされ、 `name`ラベルは宛先名前空間と一致するように更新されました：

ネームスペース	アノテーション	ラベル
ネームスペースns-2（宛先）	• annotation.one/key："updatedvalue" • annotation.two/key："true" • annotation.three/key："false"	• name=ns-2 • コンプライアンス＝hipaa • environment=production • ロール＝database

Trident Protectを設定して、データ保護操作中にファイルシステムをフリーズおよびフリーズ解除することができます。"Trident Protectを使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください".

フェイルオーバーおよびリバース操作中の実行フック

AppMirror関係を使用してアプリケーションを保護する場合、フェイルオーバーおよびリバース操作中に知っておくべき実行フックに関連する特定の動作があります。

- フェイルオーバー中、実行フックはソースクラスタからデスティネーション クラスタに自動的にコピーされます。手動で再作成する必要はありません。フェイルオーバー後、実行フックがアプリケーション上に存在し、関連するアクション中に実行されます。
- リバースまたはリバース再同期中に、アプリケーションの既存の実行フックはすべて削除されます。ソース アプリケーションがデスティネーション アプリケーションになると、これらの実行フックは無効になり、実行されないように削除されます。

実行フックの詳細については、"Trident Protect 実行フックを管理する"を参照してください。

レプリケーション関係を設定する

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 でアプリのスナップショット（アプリの Kubernetes リソースとアプリ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）を取得する頻度を選択します
- 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）

- スナップショットを作成する時間を設定する

手順

1. ソースクラスタで、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、"AppVault カスタムリソース"の例を環境に合わせて変更します：

CR を使用して **AppVault** を作成する

a. カスタムリソース（CR）ファイルを作成し、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**: (必須) 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 ファイルはこの値を参照するため、選択した名前をメモしておいてください。
- **spec.providerConfig**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な設定を保存します。bucketNameおよびプロバイダーに必要なその他の詳細を選択してください。レプリケーション関係に必要な他のCRファイルがこれらの値を参照するため、選択した値をメモしておいてください。他のプロバイダーを使用したAppVault CRの例については、"AppVault カスタムリソース"を参照してください。
- **spec.providerCredentials**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な認証情報への参照を保存します。
  - **spec.providerCredentials.valueFromSecret**：（必須）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**：（必須）選択するシークレットの有効なキー。
    - **name**：（必須）このフィールドの値を含むシークレットの名前。同じ名前空間に存在する必要があります。
  - **spec.providerCredentials.secretAccessKey**：（必須）プロバイダーにアクセスするために使用するアクセス キー。**name** は **spec.providerCredentials.valueFromSecret.name** と一致する必要があります。
- **spec.providerType**：（必須）バックアップを提供するものを決定します。例：NetApp ONTAP S3、汎用S3、Google Cloud、またはMicrosoft Azure。可能な値：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. `trident-protect-appvault-primary-source.yaml`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-appvault-primary-source.yaml -n
trident-protect
```

CLIを使用して**AppVault**を作成する

a. AppVaultを作成します。括弧内の値は、実際の環境の情報に置き換えてください：

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. ソース クラスタで、ソース アプリケーション CR を作成します：

CRを使用してソースアプリケーションを作成する

a. カスタムリソース（CR）ファイルを作成し、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**：（必須）アプリケーションのカスタム リソースの名前。レプリケーション関係に必要な他の CR ファイルはこの値を参照するため、選択した名前をメモしておいてください。
- **spec.includedNamespaces**：（必須）ネームスペースと関連付けられたラベルの配列。ネームスペース名を使用し、必要に応じてラベルでネームスペースの範囲を絞り込んで、ここにリストされているネームスペース内に存在するリソースを指定します。アプリケーション ネームスペースは、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: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. `trident-protect-app-source.yaml`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-app-source.yaml -n my-app-namespace
```

CLI を使用してソースアプリケーションを作成する

a. ソース アプリケーションを作成します。例：

```
tridentctl-protect create app <my-app-name> --namespaces
<namespaces-to-be-included> -n <my-app-namespace>
```

3. 必要に応じて、ソース クラスターでソース アプリケーションのスナップショットを取得します。このスナップショットは、デスティネーション クラスタ上のアプリケーションの基盤として使用されます。この手順をスキップした場合は、最新のスナップショットを取得するために、次にスケジュールされたスナップショットが実行されるまで待つ必要があります。オンデマンド スナップショットを作成するには、"オンデマンドスナップショットを作成する"を参照してください。

4. ソース クラスタで、レプリケーション スケジュール CR を作成します：

下記のスケジュールに加えて、ピアリングされたONTAPクラスタ間で共通のスナップショットを維持するために、7日間の保存期間を持つ個別の日次スナップショットスケジュールを作成することをお勧めします。これにより、スナップショットは最大7日間利用できるようになりますが、保存期間はユーザーの要件に基づいてカスタマイズできます。

フェイルオーバーが発生した場合、システムはこれらのスナップショットを最大7日間使用して逆の操作を行うことができます。このアプローチでは、すべてのデータではなく、最後のスナップショット以降に行われた変更のみが転送されるため、逆のプロセスがより高速かつ効率的になります。

アプリケーションの既存のスケジュールがすでに必要な保持要件を満たしている場合は、追加のスケジュールは必要ありません。

CR を使用してレプリケーションスケジュールを作成する

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、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**：（必須）スケジュールカスタムリソースの名前。
   - **spec.appVaultRef**：（必須）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**：（必須）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**：（必須）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**：`Custom`に設定する必要があります。
   - **spec.recurrenceRule**：UTC 時間での開始日と繰り返し間隔を定義します。
   - **spec.snapshotRetention**：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
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

   i. `trident-protect-schedule.yaml`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-schedule.yaml -n my-app-namespace
```

CLI を使用してレプリケーションスケジュールを作成する

a. 括弧内の値を環境の情報に置き換えて、レプリケーション スケジュールを作成します：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

例：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. デスティネーション クラスタで、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を作成し、名前を付けます（例： `trident-protect-appvault-primary-destination.yaml`）。
6. CR を適用します：

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
trident-protect
```

7. 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用のデスティネーションAppVault CRを作成します。ストレージプロバイダに応じて、"AppVault カスタムリソース"の例を環境に合わせて変更します：
   a. カスタムリソース（CR）ファイルを作成し、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**: (必須) 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 ファイルはこの値を参照するため、選択した名前をメモしておいてください。
      - **spec.providerConfig**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な設定を保存します。 `bucketName`および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ファイルがこれらの値を参照するため、選択した値をメモしておいてください。他のプロバイダーを使用したAppVault CRの例については、"AppVault カスタムリソース"を参照してください。
      - **spec.providerCredentials**：（必須）指定されたプロバイダーを使用してAppVaultにアクセスするために必要な認証情報への参照を保存します。
        - **spec.providerCredentials.valueFromSecret**：（必須）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**：（必須）選択するシークレットの有効なキー。
- **name**：（必須）このフィールドの値を含むシークレットの名前。同じ名前空間に存在する必要があります。

- **spec.providerCredentials.secretAccessKey**：（必須）プロバイダーにアクセスするために使用するアクセス キー。**name** は **spec.providerCredentials.valueFromSecret.name** と一致する必要があります。

- **spec.providerType**：（必須）バックアップを提供するものを決定します。例：NetApp ONTAP S3、汎用S3、Google Cloud、またはMicrosoft Azure。可能な値：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. `trident-protect-appvault-secondary-destination.yaml`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-appvault-secondary-destination.yaml
-n trident-protect
```

8. デスティネーション クラスタで、AppMirrorRelationship CR ファイルを作成します。

> CR を使用する場合は、CR を適用する前に、デスティネーション ネームスペースを手動で作成します。Trident Protect は、CLI を使用する場合にのみネームスペースを自動的に作成します。

CRを使用して**AppMirrorRelationship**を作成する

a. カスタムリソース（CR）ファイルを作成し、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**：（必須）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**：（必須）デスティネーション名前空間とソース名前空間は、それぞれのアプリケーション CR で定義されたアプリケーション名前空間と一致する必要があります。
- **spec.sourceAppVaultRef**：（必須）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**：（必須）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.sourceApplicationUID**：（必須）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 UID と一致する必要があります。
- **spec.storageClassName**：（オプション）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、ソース環境とピアリングされているONTAPストレージVMにリンクされている必要があります。ストレージクラスが指定されていない場合は、クラスタのデフォルトのストレージクラスがデフォルトで使用されます。
- **spec.recurrenceRule**：UTC 時間での開始日と繰り返し間隔を定義します。

  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: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. `trident-protect-relationship.yaml`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-relationship.yaml -n my-app-
namespace
```

CLI を使用して **AppMirrorRelationship** を作成する

a. AppMirrorRelationshipオブジェクトを作成して適用し、括弧内の値を環境の情報に置き換えます：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --source-app-vault <my_vault_name> --recurrence
-rule <rule> --namespace-mapping <ns_mapping> --source-app-id
<source_app_UID> --source-app <my_source_app_name> --storage
-class <storage_class_name> -n <application_namespace>
```

例：

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. （オプション）デスティネーション クラスタで、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 Protect 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。この手順では、レプリケーション関係を停止し、デスティネーション クラスタでアプリをオンラインにします。Trident Protect は、ソース クラスタ上のアプリが稼働中であっても停止しません。

手順

1. デスティネーション クラスタで、AppMirrorRelationship CR ファイル（例： `trident-protect-relationship.yaml`）を編集し、**spec.desiredState** の値を `Promoted`に変更します。
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）fail over 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
```

フェイルオーバーしたレプリケーション関係を再同期する

再同期操作により、レプリケーション関係が再確立されます。再同期操作を実行すると、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 クラスタ上の実行中のアプリケーションに対して行われ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. オプション：ソース クラスターで、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。これにより、ソース クラスターからの最新の変更が確実にキャプチャされます。
2. デスティネーション クラスタで、AppMirrorRelationship CR ファイル（例： `trident-protect-relationship.yaml`）を編集し、spec.desiredState の値を `Established`に変更します。
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、それらを削除します。スケジュールが残っていると、ボリューム スナップショットが失敗します。

フェイルオーバーしたレプリケーション関係を逆再同期する

フェー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、それらを削除します。
2. 元々関係を設定するために使用した CR ファイルを反対側のクラスタに適用して、レプリケーション関係を設定します。
3. 新しいデスティネーション（元のソースクラスタ）が両方のAppVault CRで設定されていることを確認します。
4. 逆方向の値を設定して、反対側のクラスタにレプリケーション関係を設定します。

アプリケーションのレプリケーション方向を逆にする

レプリケーションの方向を逆にすると、Trident Protect は、元のソース ストレージ バックエンドへのレプリケーションを継続しながら、アプリケーションをデスティネーション ストレージ バックエンドに移動します。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複製します。

この状況では、ソースとデスティネーションが入れ替わっています。

手順

1. ソース クラスタで、シャットダウン スナップショットを作成します：

CRを使用してシャットダウンスナップショットを作成する

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、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**：（必須）カスタム リソースの名前。
- **spec.AppVaultRef**：（必須）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**：（必須）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: my-app-name
```

c. `trident-protect-shutdownsnapshot.yaml`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-shutdownsnapshot.yaml -n my-app-namespace
```

CLI を使用してシャットダウン **Snapshot** を作成する

a. 括弧内の値を環境の情報に置き換えて、シャットダウン スナップショットを作成します。例：

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot> -n <application_namespace>
```

2. ソース クラスタで、シャットダウン Snapshot が完了したら、シャットダウン 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. ソース クラスタで、次のコマンドを使用して **shutdownsnapshot.status.appArchivePath** の値を見つけ、ファイル パスの最後の部分（ベース名とも呼ばれます。最後のスラッシュの後のすべてです）を記録します：

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 新しいデスティネーション クラスタから新しいソース クラスタへのフェイルオーバーを実行します。次の変更を加えます：

> フェイルオーバー手順のステップ 2 で、AppMirrorRelationship CR ファイルに `spec.promotedSnapshot`フィールドを含め、その値を上記の手順 3 で記録したベース名に設定します。

5. [フェイルオーバーしたレプリケーション関係を逆再同期する]で逆再同期の手順を実行します。
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

逆レプリケーションにより、次のアクションが発生します：

- 元のソース アプリの Kubernetes リソースのスナップショットが取得されます。
- 元のソース アプリのポッドは、アプリの 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 はそのまま残ります）。
- 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レプリケートされたボリューム データを使用して、シャットダウン前の Snapshot からリストアされます。
- レプリケーションは逆方向に再確立されます。

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する

Trident Protect を使用すると、次の一連の操作を使用して、フェイルオーバー操作後に「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 Protect は、レプリケーションの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複製（再同期）します。

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を完了した関係から開始され、次の手順が含まれます：

- 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。フェイルオーバー手順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. [フェイルオーバーしたレプリケーション関係を逆再同期する]の手順を実行します。
2. [アプリケーションのレプリケーション方向を逆にする]の手順を実行します。

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、関係のない 2 つの別個のアプリケーションが作成されます。

手順

1. 現在の宛先クラスタで、AppMirrorRelationship CRを削除します：

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

Trident Protectを使用してアプリケーションを移行する

バックアップ データを復元することで、アプリケーションをクラスター間または異なるストレージ クラスに移行できます。

> アプリケーションを移行すると、アプリケーション用に設定されたすべての実行フックも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存在する場合、リストア処理の一部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およびリストア処理を実行するには、特定のバックアップおよびリストアタスクを自動化できます。

同じクラスタにクローンする

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. "スナップショットを作成する"。
   b. "バックアップを作成する"。

2. 同じクラスタで、スナップショットを作成したかバックアップを作成したかに応じて、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復元する"。
   b. "バックアップからデータを復元する"。

別のクラスタにクローンする

アプリケーションを別のクラスターに複製するには（クラスター間複製を実行する）、ソース クラスターでバックアップを作成し、そのバックアップを別のクラスターに復元します。Trident Protectがデスティネーション クラスタにインストールされていることを確認してください。

異なるクラスタ間でアプリケーションを複製するには、"SnapMirrorレプリケーション"を使用します。

手順

1. "バックアップを作成する"。
2. バックアップ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. デスティネーション クラスタで、"バックアップからデータを復元する"。

アプリケーションをあるストレージ クラスから別のストレージ クラスに移行する

バックアップをデスティネーション ストレージ クラスに復元することで、アプリケーションを 1 つのストレージ クラスから別のストレージ クラスに移行できます。

たとえば（復元 CR からシークレットを除外）：

```
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を使用してスナップショットを復元する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-snapshot-restore-cr.yaml`という名前を付けます。
2. 作成したファイルで、次の属性を設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.appArchivePath**：AppVault内のパスで、スナップショットの内容が保存される場所。このパスを見つけるには、次のコマンドを使用できます：

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**：（必須）スナップショットの内容が保存されるAppVaultの名前。
   - **spec.namespaceMapping**：リストア処理のソースネームスペースからデスティネーションネームスペースへのマッピング。 `my-source-namespace`と `my-destination-namespace`を環境の情報に置き換えます。

     ```
     ---
     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. オプションとして、復元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、特定のラベルでマークされ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須） `include or exclude`を使用して、resourceMatchersで定義されたリソースを含めるか除外します。以下のresourceMatchersパラメータを追加して、含める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。この配列に複数の要素を定義すると、それらはOR演算として一致し、各要素内のフィールド（グループ、種類、バージョン）はAND演算として一致します。
       - **resourceMatchers[].group**：（オプション）フィルタリングするリソースのグループ。
       - **resourceMatchers[].kind**：（オプション）フィルタリングするリソースの種類。
       - **resourceMatchers[].version**：（オプション）フィルタリングするリソースのバージョ

ン。

- **resourceMatchers[].names**：（オプション）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**：（オプション）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**：（オプション） "Kubernetesドキュメント"で定義されているリソースのKubernetesメタデータ.nameフィールドのラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. `trident-protect-snapshot-restore-cr.yaml`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-snapshot-restore-cr.yaml
```

CLI を使用してスナップショットを復元する

手順

1. 括弧内の値を環境の情報に置き換えて、スナップショットを別のネームスペースにリストアします。
   - その `snapshot`引数は、名前空間とスナップショット名を次の形式で使用します `<namespace>/<name>`。
   - `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`source1:dest1,source2:dest2`形式で正しいデスティネーション名前空間にマッピングします。

     例：

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

Trident Protect 実行フックを管理する

実行フックは、管理対象アプリ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、スナップショットが完了した後にトランザクションを再開できます。これにより、アプリケーションの一貫性のあるスナップショットが保証されます。

実行フックのタイプ

Trident Protect は、実行できる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ています。

- 事前スナップショット
- スナップショット後
- バックアップ前
- バックアップ後
- 復元後
- フェイルオーバー後

実行順序

データ保護操作が実行されると、実行フック イベントが次の順序で発生します：

1. 適用可能なカスタム操作前実行フックは、適切なコンテナで実行されます。必要な数だけカスタム操作前フックを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されておらず、構成もできません。
2. 該当する場合、ファイルシステムのフリーズが発生します。"Trident Protectを使用したファイルシステムのフリーズの設定の詳細については、こちらをご覧ください"。
3. データ保護操作が実行されます。
4. 該当する場合、凍結されたファイルシステムは凍結解除されます。
5. 適用可能なカスタム操作後実行フックは、適切なコンテナで実行されます。必要な数だけカスタム操作後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ておらず、構成もできません。

同じタイプ（たとえば、pre-snapshot）の実行フックを複数作成する場合、そ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ます。たとえば、さまざまな種類のフックがすべて含まれる構成の実行順序は次のとおりです：

1. スナップショット前のフックが実行されました

2. スナップショット後のフックが実行されました
3. バックアップ前のフックが実行されました
4. バックアップ後のフックが実行されました

上記の順序の例は、既存のスナップショットを使用しないバックアップを実行する場合にのみ適用されます。

本番環境で実行フックスクリプトを有効にする前に、必ず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後、結果のスナップショットとバックアップをテストして、一貫性があることを確認します。これを行うには、アプリを一時的な名前空間に複製し、スナップショットまたはバックアップを復元してから、アプリをテストします。

スナップショット前の実行フックによって Kubernetes リソースが追加、変更、または削除された場合、それらの変更はスナップショットまたはバックアップと、その後のすべてのリストア処理に含まれます。

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、アクションを実行するためにスクリプトを使用する必要があります。複数の実行フックが同じスクリプトを参照できます。
- Trident Protect 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 Protect は実行フックの設定と一致する基準を使用して、スナップショット、バックアップ、または復元操作に適用可能なフックを決定します。

実行フックは、多くの場合、実行対象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ため、カスタム実行フックの実行にかかる時間を常に最小限に抑えるようにする必要があります。関連する実行フックを使用してバックアップまたは Snapshot 操作を開始したが、その後キャンセルした場合でも、バックアップまたは Snapshot 操作がすでに開始されている場合は、フックは引き続き実行できます。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では、バックアップが完了したと想定することはできません。

実行フックフィルター

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するが、各イメージを異なる目的で使用する可能性があるアプリケーション（Elasticsearchなど）に役立ちます。フィルタを使用すると、実行フックが必ずしもすべての同一コンテナではなく一部のコンテナで実行されるシナリオを作成できます。単一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演算子で結合されます。実行フックごとに最大10個のアクティブフィルタを設定できます。

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。フックがコンテナに一致すると、フックはそのコンテナ上で関連付けられたスクリプトを実行します。フィルタの正規表現では正規表現 2（RE2）構文が使用されますが、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。Trident Protect が実行フックフィルタの正規表現でサポートする構文については、 "正規表現2（RE2）構文のサポート"を参照してください。

リストアまたはクローン処理の後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

`https://github.com/NetApp/Verda["NetApp Verda GitHubプロジェクト "]`にアクセスして、`Apache Cassandra` や `Elasticsearch` などの一般的なアプリの実際の実行フックをダウンロードしてください。また、例を参照したり、独自のカスタム実行フックを構成するためのアイデアを入手したりすることもできます。

実行フックを作成する

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、所有者、管理者、またはメンバーの権限が必要です。

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-hook.yaml`という名前を付けます。
2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.applicationRef**：（必須）実行フックを実行するアプリケーションの Kubernetes 名。
   - **spec.stage**：（必須）アクション中に実行フックを実行するステージを示す文字列。可能な値：
     - プレ
     - 投稿
   - **spec.action**: (必須) 指定された実行フック フィルター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。可能な値：
     - Snapshot
     - バックアップ
     - リストア
     - フェイルオーバー
   - **spec.enabled**: (オプション) 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 になります。
   - **spec.hookSource**：（必須）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**：（オプション）実行フックの実行が許可される時間（分）を定義する数値。最小値は 1 分で、指定されていない場合のデフォルト値は 25 分です。
   - **spec.arguments**: （オプション）実行フックに指定できる引数の YAML リスト。
   - **spec.matchingCriteria**：（オプション）条件キーと値のペアのオプションのリスト。各ペアは実行フック フィルターを構成します。実行フックごとに最大 10 個のフィルターを追加できます。
   - **spec.matchingCriteria.type**：（オプション）実行フックのフィルタータイプを識別する文字列。可能な値：
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value**：（オプション）実行フックのフィルター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 を使用する

手順

1. 実行フックを作成し、括弧内の値を環境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> -n <application_namespace>
```

実行フックを手動で実行する

テスト目的で、または失敗後にフックを手動で再実行する必要がある場合は、実行フックを手動で実行できます。実行フックを手動で実行するには、Owner、Admin、またはMemberの権限が必要です。

実行フックを手動で実行するには、次の 2 つの基本的な手順を実行します：

1. リソースのバックアップを作成します。これはリソースを収集してそれらのバックアップを作成し、フックが実行される場所を決定します
2. バックアップに対して実行フックを実行する

ステップ**1**：リソースのバックアップを作成する

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-resource-backup.yaml`という名前を付けます。
2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.applicationRef**：（必須）リソース バックアップを作成するアプリケーションの Kubernetes 名。
   - **spec.appVaultRef**：（必須）バックアップコンテンツが保存されるAppVaultの名前。
   - **spec.appArchivePath**：AppVault内でバックアップコンテンツが保存されるパス。このパスを見つけるには、次のコマンドを使用できます：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     YAMLの例：

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: ResourceBackup
     metadata:
       name: example-resource-backup
     spec:
       applicationRef: my-app-name
       appVaultRef: my-appvault-name
       appArchivePath: example-resource-backup
     ```

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-resource-backup.yaml
   ```

CLI を使用する

手順

1. 括弧内の値を環境の情報に置き換えてバックアップを作成します。例：

   ```
   tridentctl protect create resourcebackup <my_backup_name> --app
   <my_app_name> --appvault <my_appvault_name> -n
   <my_app_namespace> --app-archive-path <app_archive_path>
   ```

2. バックアップのステータスを表示します。操作が完了するまで、この例のコマンドを繰り返し使用できます：

```
tridentctl protect get resourcebackup -n <my_app_namespace>
<my_backup_name>
```

3. バックアップが成功したことを確認します：

```
kubectl describe resourcebackup <my_backup_name>
```

ステップ**2**：実行フックを実行する

CRを使用する

手順

1. カスタムリソース（CR）ファイルを作成し、 `trident-protect-hook-run.yaml`という名前を付けます。
2. 以下の属性を Trident Protect 環境とクラスタ構成に合わせて設定します：
   - **metadata.name**：（必須）このカスタム リソースの名前。環境に合わせて一意かつ適切な名前を選択してください。
   - **spec.applicationRef**：（必須）この値が、手順 1 で作成した ResourceBackup CR のアプリケーション名と一致していることを確認してください。
   - **spec.appVaultRef**：（必須）この値が、手順1で作成したResourceBackup CR のappVaultRefと一致していることを確認してください。
   - **spec.appArchivePath**：この値が、手順1で作成したResourceBackup CR のappArchivePathと一致していることを確認してください。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.action**: (必須) 指定された実行フック フィルター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。可能な値：
     - Snapshot
     - バックアップ
     - リストア
     - フェイルオーバー
   - **spec.stage**：（必須）アクション中に実行フックを実行するステージを示す文字列。このフック実行では、他のステージのフックは実行されません。可能な値：
     - プレ
     - 投稿

     YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

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-run.yaml
```

CLI を使用する

手順

1. 手動実行フックの実行リクエストを作成します：

```
tridentctl protect create exechooksrun <my_exec_hook_run_name>
-n <my_app_namespace> --action snapshot --stage <pre_or_post>
--app <my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. 実行フックの実行ステータスを確認します。操作が完了するまでこのコマンドを繰り返し実行できます：

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. 最終的な詳細とステータスを確認するには、exechooksrun オブジェクトを記述します：

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

Trident Protectをアンインストールする

試用版から製品の完全版にアップグレードする場合は、Trident Protectコンポーネントを削除する必要があるかもしれません。

Trident Protectを削除するには、次の手順を実行します。

手順

1. Trident Protect CR ファイルを削除します：

   バージョン 25.06 以降ではこの手順は必要ありません。

   ```
   helm uninstall -n trident-protect trident-protect-crds
   ```

2. Trident Protectを削除：

   ```
   helm uninstall -n trident-protect trident-protect
   ```

3. Trident Protect ネームスペースを削除します：

   ```
   kubectl delete ns trident-protect
   ```

TridentおよびTrident Protectのブログ

ここでは、優れたNetApp TridentおよびTrident Protectのブログをご覧いただけます：

Tridentブログ

- 2025年10月16日："Kubernetes向けの高度なストレージソリューション"
- 2025年8月19日："データの整合性の強化：OpenShift 仮想化における Trident を使用したボリュームグループスナップショット"
- 2025年5月9日:"Amazon EKSアドオンを使用したFSx for ONTAPの自動Tridentバックエンド構成"
- 2025年4月15日："SMBプロトコル対応のGoogle Cloud NetApp Volumesを使用したNetApp Trident"
- 2025年4月14日："Trident 25.02 でファイバチャネルプロトコルを活用した Kubernetes の永続ストレージ"
- 2025年4月14日："Kubernetes ブロックストレージ向けNetApp ASA r2 システムの能力を引き出す"
- 2025年3月31日:"新しい認定オペレーターによるRed Hat OpenShiftへのTridentインストールの簡素化"
- 2025年3月27日："Google Cloud NetApp Volumes を使用した SMB 向け Trident のプロビジョニング"
- 2025年3月5日："シームレスな iSCSI ストレージ統合を実現：AWS 向け ROSA クラスター上の FSxN ガイド"
- 2025年2月27日："Trident、GKE、Google Cloud NetApp Volumes を使用したクラウド ID の導入"
- 2024年12月12日："Tridentでのファイバーチャネルサポートの導入"
- 2024年11月11日："NetApp TridentとGoogle Cloud NetApp Volumes"
- 2024年10月29日："Trident を使用した Amazon FSx for NetApp ONTAP と Red Hat OpenShift Service on AWS（ROSA）"
- 2024年10月29日："ROSA上のOpenShift VirtualizationとAmazon FSx for NetApp ONTAPを使用したVMのライブマイグレーション"
- 2024年7月8日:"NVMe/TCPを使用してAmazon EKS上の最新のコンテナ化されたアプリ用のONTAPストレージを消費する"
- 2024年7月1日："Google Cloud NetApp Volumes Flex と Astra Trident によるシームレスな Kubernetes ストレージ"
- 2024年6月11日："OpenShiftの統合イメージレジストリのバックエンドストレージとしてのONTAP"

Trident Protectブログ

- 2025年5月16日:"Trident Protectの復元後フックを使用した災害復旧のためのレジストリフェイルオーバーの自動化"
- 2025年5月16日:"OpenShift 仮想化ディザスタリカバリと NetApp Trident Protect"
- 2025年5月13日:"Trident Protectのバックアップ"
- 2025年5月9日:"Trident Protectのリストア後フックを使用してKubernetesアプリケーションをリスケールする"

- 2025年4月3日："Trident Protect Power Up：保護と災害復旧のための Kubernetes レプリケーション"
- 2025年3月13日："OpenShift Virtualization VMのクラッシュ整合性バックアップおよびリストア処理"
- 2025年3月11日:"NetApp Trident を使用したアプリケーションデータ保護への GitOps パターンの拡張"
- 2025年3月3日："Trident 25.02：エキサイティングな新機能でRed Hat OpenShiftエクスペリエンスを向上"
- 2025年1月15日："Trident Protectのロールベースアクセス制御のご紹介"
- 2024年11月11日："Kubernetesによるデータ管理：Trident Protectによる新時代"

ナレッジとサポート

よくある質問

Tridentのインストール、構成、アップグレード、トラブルシューティングに関するよくある質問への回答を見つけます。

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か月ごとにリリースされます。

Trident は **Kubernetes** の特定のバージョンでリリースされる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リリース内でベータ機能をサポートする場合があります。

Tridentが機能するために、他の**NetApp**製品に依存していますか？

Tridentは、他のNetAppソフトウェア製品に依存せず、スタンドアロンアプリケーションとして動作します。ただし、NetAppバックエンドストレージデバイスが必要です。

完全な**Trident**設定の詳細を入手するにはどうすればよいですか？

`tridentctl get`コマンドを使用して、Trident構成の詳細情報を取得します。

Trident によってストレージがどのようにプロビジョニングされているかの指標を取得できますか？

はい。Prometheusエンドポイントを使用して、管理対象のバックエンドの数、プロビジョニングされたボリュームの数、消費されたバイト数など、Tridentの動作に関する情報を収集できます。"Cloud Insights"を監視や分析に使用することもできます。

Trident を **CSI** プロビジョナーとして使用する場合、ユーザーエクスペリエンスは変わりますか？

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ナー名は`csi.trident.netapp.io`です。このTridentのインストール方法は、現在のリリース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したい場合に推奨されます。

Kubernetes クラスタに Trident をインストールして使用する

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。"Tridentのインストールについて"を参照してください。

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、クラスタへの `kubectl`アクセスを持つ任意のマシンからのリモートインストール機能をサポートしています。 `kubectl`アクセスが確認されたら（例えば、リモートマシンから `kubectl get nodes`コマンドを実行して確認）、インストール手順に従ってください。

Tridentで高可用性を構成できますか？

Trident は Kubernetes デプロイメント（ReplicaSet）として 1 つのインスタンスで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さないでください。Trident がインストールされているノードが失われたり、ポッドにアクセスできなくなったりした場合、Kubernetes はポッドをクラスター内の正常なノードに自動的に再デプロイします。Trident はコントロールプレーンのみであるため、Trident が再デプロイされても、現在マウントされているポッドは影響を受けません。

Tridentは**kube-system**名前空間にアクセスする必要がありますか？

Trident は、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に Kubernetes API サーバーから読み取りを行うため、kube-system へのアクセスが必要です。

Tridentで使用される役割と権限は何ですか？

Tridentインストーラは、KubernetesのClusterRoleを作成します。これにより、KubernetesクラスタのPersistentVolume、PersistentVolumeClaim、StorageClass、およびSecretリソースへの特定のアクセス権が付与されます。"tridentctl インストールのカスタマイズ"を参照してください。

Trident がインストールに使用する正確な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正確な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。"tridentctl インストールのカスタマイズ"を参照してください。

2 つの別々の **Kubernetes** クラスター用の **2** つの別々の **Trident** インスタンスに対して、同じ **ONTAP** バックエンド **SVM** を共有できますか？

推奨はされませんが、2つのTridentインスタンスで同じバックエンドSVMを使用することは可能です。インストール時に各インスタンスごとに一意のボリューム名を指定するか、または `StoragePrefix`ファイル内で一意の `setup/backend.json`パラメータを指定してください。これは、同じFlexVol volumeが両方のインスタンスで使用されないようにするためです。

ContainerLinux（旧 **CoreOS**）に **Trident** をインストールすることは可能ですか？

Trident は単なる Kubernetes ポッドであり、Kubernetes が実行されている場所であればどこにでもインストールできます。

Trident を **NetApp Cloud Volumes ONTAP** で使用できますか？

はい、Trident は AWS、Google Cloud、Azure でサポートされています。

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が完全にサポートします。

サポートケースを提出するにはどうすればよいですか？

サポートケースを提出するには、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. "NetAppサポート"に連絡してサポートケースを提起してください。

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？

サポートバンドルを作成するには、 `tridentctl logs -a`を実行します。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ログを取得する手順は、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提出する必要がある場合はどうすればよいですか？

`https://github.com/NetApp/trident["Trident Github"^]`で問題を作成し、問題の件名と説明に `*RFE*` を記載してください。

不具合はどこに報告すればよいですか？

`https://github.com/NetApp/trident["Trident Github"^]`で問題を作成します。問題に関連する必要な情報とログをすべて含めるようにしてください。

Tridentについて明確にする必要がある簡単な質問がある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 "Discordチャンネル"またはGitHubからお問い合わせください。

ストレージシステムのパスワードが変更され、**Trident**が動作しなくなりました。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`で更新します。例の `myBackend`をバックエンド名に置き換え、 `/path/to_new_backend.json`を正しい `backend.json`ファイルへのパスに置き換えます。

Trident で **Kubernetes** ノードが見つかりません。これをどうすれば修正できますか？

Trident が Kubernetes ノードを見つけられない理由として、2 つのシナリオが考えられます。これは、Kubernetes 内のネットワークの問題または DNS の問題が原因である可能性があります。各 Kubernetes ノードで実行される Trident ノードデーモンセットは、Trident コントローラと通信してノードを Trident に登録できる必要があります。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、この問題は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ポッドが破壊された場合、データは失われますか？

Tridentポッドが破壊されてもデータが失われることはありません。Tridentメタデータは CRD オブジェクトに保存されます。Tridentによってプロビジョニングされたすべての PV は正常に機能します。

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（いくつかのバージョンをスキップして）アップグレードできますか？

NetAppは、Tridentのあるメジャーリリースから次のメジャーリリースへのアップグレードをサポートしています。バージョン18.xxから19.xx、19.xxから20.xxといったようにアップグレードできます。実稼働環境に展開する前に、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することは可能ですか？

アップグレード後に発見されたバグ、依存関係の問題、またはアップグレードの失敗や不完全さに対する修正が必要な場合は、"Tridentをアンインストールします"そのバージョン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。これは以前のバージョンにダウングレードする場合に推奨される唯一の方法です。

バックエンドとボリュームを管理する

ONTAP バックエンド定義ファイルで管理 **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。DataLIF は次のように異なります：

- ONTAP SAN：iSCSI の場合は指定しないでください。Trident は"ONTAP セレクティブLUNマップ"を使用して、マルチパスセッションを確立するために必要な iSCSI LIF を検出します。 `dataLIF`が明示的に定義されている場合、警告が生成されます。詳細については、"ONTAP SAN 構成オプションと例"を参照してください。
- ONTAP NAS：NetAppでは、 `dataLIF`を指定することを推奨しています。指定しない場合、TridentはSVMからdataLIFを取得します。NFSマウント操作に使用する完全修飾ドメイン名（FQDN）を指定することで、複数のdataLIF間で負荷を分散するラウンドロビンDNSを作成できます。詳細については"ONTAP NAS 構成オプションと例"を参照してください。

Trident は **ONTAP** バックエンドに **CHAP** を設定できますか？

はい。TridentはONTAPバックエンドの双方向CHAPをサポートしています。これには、バックエンド設定で`useCHAP=true`を設定する必要があります。

Trident で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バージョン20.04以降では、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で1つ以上のCIDRブロックを提供し、Tridentがこれらの範囲内にあるノードIPを、作成するエクスポートポリシーに追加します。このように、Tridentは指定されたCIDR内の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。

管理およびデータ **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 バックエンド用）。
- ONTAP SAN バックエンドの`managementLIF`。ONTAP SAN バックエンドでは `dataLIF`を指定できません。

Tridentは、IPv6経由で機能させるために、フラグ --use-ipv6（ tridentctl`インストールの場合）、 `IPv6（Tridentオペレータの場合）、または tridentTPv6（Helmインストールの場合）を使用してインストールする必要があります。

バックエンドで管理 **LIF** を更新することは可能ですか**?**

はい、 `tridentctl update backend`コマンドを使用してバックエンド管理LIFを更新することができます。

バックエンドで **DataLIF** を更新することは可能ですか？

DataLIFを更新できるのは `ontap-nas`と `ontap-nas-economy`のみです。

Kubernetes 用の **Trident** で複数のバックエンドを作成できますか？

Tridentは、同じドライバーでも異なるドライバーでも、同時に複数のバックエンドをサポートできます。

Trident はバックエンドのクレデンシャルをどのように保存しますか？

Tridentは、バックエンドのクレデンシャルをKubernetes Secretsとして保存します。

Tridentはどのように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、 `storagePools`および `additionalStoragePools`パラメータを使用して特定のプールの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しないようにするにはどうすればよいですか？

`excludeStoragePools`パラメータは、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するために使用され、一致するプールが削除されます。

同じ種類のバックエンドが複数ある場合、**Trident**はどのバックエンドを使用するかをどのように選択しますか？

同じタイプのバックエンドが複数設定されている場合、Tridentは `StorageClass`と `PersistentVolumeClaim`に存在する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。たとえば、ontap-nasドライババックエンドが複数ある場合、Tridentは `StorageClass`と `PersistentVolumeClaim`のパラメータを組み合わせて一致させ、 `StorageClass`と `PersistentVolumeClaim`にリストされている要件を満たすバックエンドと一致させようとします。リクエスト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 は **ONTAP** ボリュームに **Qtree** をどのように展開しますか？**1** つのボリュームにいくつの **Qtree** を展開できますか？

`ontap-nas-economy`ドライバは、同じFlexVolボリュームに最大200個のQtree（50～300の間で設定可能）、クラスタノードあたり100,000個のQtree、クラスタあたり2.4MのQtreeを作成します。economyドライバによって処理される新しい`PersistentVolumeClaim`を入力すると、ドライバは新しいQtreeを処理できるFlexVolボリュームがすでに存在するかどうかを確認します。Qtreeを処理できるFlexVolボリュームが存在しない場合は、新しいFlexVolボリュームが作成されます。

ONTAP NAS上でプロビジョニングされたボリュームの**Unix**権限を設定するにはどうすればよいですか？

バックエンド定義ファイルに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に Unix 権限を設定できます。

ボリュームのプロビジョニング時に **ONTAP NFS** マウントオプションの明示的なセットを設定するにはどうすればよいですか。

デフォルトでは、Trident は Kubernetes ではマウントオプションを任意の値に設定しません。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、"ここをクリックしてください。"に示されている例に従ってください。

プロビジョニングされ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で設定された`exportPolicy`パラメータを使用します。

Tridentと**ONTAP**を使用してボリューム暗号化を設定するにはどうすればよいですか？

Trident でプロビジョニングされたボリュームに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 の連携"

Trident を通じて **ONTAP** に **QoS** を実装する最良の方法は何ですか？

`StorageClasses`を使用して、ONTAPのQoSを実装します。

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のいずれかをサポートします。ONTAP ドライバはデフォルトでシンプロビジョニングを使用します。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 `StorageClass`を設定する必要があります。両方が設定されている場合、 `StorageClass`が優先されます。ONTAP に対して以下を設定します：

1. `StorageClass`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 parameter`をボリュームとして設定することで、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 バージョン 1.10 以降では、 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を拡張することができます。ボリュームの自動拡張はONTAPの機能であり、Tridentには適用されないことに注意してください。

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場合、インポートできますか？

外部ボリュームが DP モードまたはオフラインの場合、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.
```

リソース割り当ては**NetApp**クラスタにどのように変換されますか？

Kubernetesストレージリソースクォータは、NetAppストレージに容量がある限り機能します。NetAppストレージが容量不足のためKubernetesのクォータ設定を満たせない場合、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スナップショットを作成できますか？

はい。オンデマンドボリュームスナップショットの作成とSnapshotからのPersistent Volumeの作成は、Tridentでサポートされています。SnapshotからPVを作成するには、 `VolumeSnapshotDataSource`機能ゲートが有効になっていることを確認してください。

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は何ですか？

本日より、オンデマンドスナップショットサポートは、 `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、および `azure-netapp-files`バックエンドドライバで利用できます。

Trident と **ONTAP** でプロビジョニングされたボリュームのスナップショットバックアップを取得するにはどうすればよいですか。

これは `ontap-nas`、 `ontap-san`、および `ontap-nas-flexgroup`ドライバで利用できます。また、FlexVolレベルで `ontap-san-economy`ドライバ用の `snapshotPolicy`を指定することもできます。

これは `ontap-nas-economy`ドライバでも使用できますが、qtreeレベルの粒度ではなく、FlexVol volumeレベルの粒度で使用できます。Tridentでプロビジョニングされたボリュームのスナップショット機能を有効にするには、バックエンドパラメータオプション `snapshotPolicy`をONTAPバックエンドで定義されている目的のスナップショットポリシーに設定します。ストレージコントローラによって作成されたスナップショットは、Tridentでは認識されません。

Trident を使用してプロビジョニングされたボリュームのスナップショット予約率を設定できますか？

はい、バックエンド定義ファイルで `snapshotReserve`属性を設定することで、Trident を使用してスナップショットコピーを保存するためのディスクスペースの特定の割合を予約できます。バックエンド定義ファイルで `snapshotPolicy`と `snapshotReserve`を設定した場合、スナップショットリザーブの割合は、バックエンドファイルに記載されている `snapshotReserve`の割合に従って設定されます。 `snapshotReserve`の割合が記載されていない場合、ONTAP はデフォルトでスナップショットリザーブの割合を 5 とします。 `snapshotPolicy`オプションが none に設定されている場合、スナップショットリザーブの割合は 0 に設定されます。

ボリューム **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 `snapshotDir`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に対して**SnapMirror**を設定できますか？

現在、SnapMirrorはONTAP CLIまたはOnCommand System Managerを使用して外部から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スナップショットに復元するには、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. 必要なスナップショットに戻すには、ONTAP CLI または OnCommand System Manager を使用します。
3. アプリケーションポッドを再起動します。

Trident は、ロードシェアリングミラーが設定されている **SVM** にボリュームをプロビジョニングできますか？

NFS 経由でデータを提供する SVM のルートボリュームに対して負荷共有ミラーを作成できます。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。これにより、ボリュームのマウントに遅延が発生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、ボリュームのプロビジョニングは ONTAP による負荷共有ミラーの更新に依存します。

各顧客/テナントのストレージクラスの使用を分離するにはどうすればよいですか？

Kubernetes では、名前空間内のストレージクラスは許可されません。ただし、Kubernetes では、名前空間ごとのストレージリソースクォータを使用して、名前空間ごとに特定のストレージクラスの使用を制限できます。特定のストレージへの特定の名前空間の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ついては、ここに示すヒントを参照してください。

Tridentのヘルプについては、 `tridentctl logs -a -n trident`を使用してサポートバンドルを作成し、NetAppサポートに送信してください。

一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（例えば、Tridentポッドが `ContainerCreating`フェーズで準備完了コンテナが2つ未満の状態で停止している場合）、 `kubectl -n trident describe deployment trident`および `kubectl -n trident describe pod trident-**`を実行すると、追加の情報が得られます。kubeletログの取得（例： `journalctl -xeu kubelet`経由）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`フラグを渡すことで、Tridentのデバッグモードを有効にすることができます。

  次に、 `./tridentctl logs -n trident`を使用してデバッグが設定されていることを確認し、ログ内で `level=debug msg`を検索します。

  Operatorを使用してインストール

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

  これにより、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。これは、 `kubectl get pod -n trident`の出力の「AGE」列を観察することで確認できます。

  Trident 20.07および20.10の場合は、 `tprov`を `torc`の代わりに使用してください。

  Helmでインストール

  ```
  helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
  tridentDebug=true`
  ```

  tridentctlでインストール

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

- 各バックエンドごとにデバッグログを取得するには、バックエンド定義に `debugTraceFlags`を含めることもできます。例えば、TridentログでAPIコールやメソッドのトラバーサルを取得するには、 `debugTraceFlags: {"api":true, "method":true,}`を含めてください。既存のバックエンドには、 `debugTraceFlags`を `tridentctl backend update`で設定できます。
- Red Hat Enterprise Linux CoreOS (RHCOS)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てください。これは、OpenShift MachineConfigsを使用するか、ignitionテンプレートを変更することで実行できます。
- Tridentを "Azure NetApp Files"で使用する際に発生する可能性のある一般的な問題は、テナントシークレットとクライアントシークレットが、権限が不十分なアプリ登録から取得された場合です。Trident要件の完全なリストについては、"Azure NetApp Files"構成を参照してください。
- PVをコンテナにマウントする際に問題がある場合は、 `rpcbind`がインストールされ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ーを使用して、 `rpcbind`が実行中かどうかを確認してください。 `rpcbind`サービスのステータスは、 `systemctl status rpcbind`またはそれと同等のものを実行することで確認できます。
- Tridentバックエンドが以前は動作していたにもかかわらず `failed`状態であると報告する場合は、バック

エンドに関連付けられたSVM/管理者の資格情報が変更されたことが原因である可能性があります。`tridentctl update backend`を使用してバックエンド情報を更新するか、Tridentポッドをバウンスすることで、この問題は解決されます。

- コンテナランタイムとしてDockerを使用してTridentをインストールする際に権限の問題が発生した場合は、`--in cluster=false`フラグを使用してTridentのインストールを試みてください。これにより、インストーラポッドが使用されず、`trident-installer`ユーザーが原因で発生する権限の問題を回避できます。
- `uninstall parameter <Uninstalling Trident>`を使用して、実行が失敗した後のクリーンアップを行います。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、実行中のデプロイメントでも安全にアンインストールして再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、まず`tridentctl uninstall`コマンドを実行してTridentを削除します。目的の"Tridentバージョン"をダウンロードし、`tridentctl install`コマンドを使用してインストールします。
- インストールが成功した後、PVCが`Pending`フェーズでスタックしている場合、`kubectl describe pvc`を実行すると、TridentがこのPVCのPVをプロビジョニングできなかった理由についての追加情報を提供できます。

オペレータを使用したTrident展開の失敗

オペレータを使用してTridentを導入する場合、`TridentOrchestrator`のステータスが`Installing`から`Installed`に変わります。`Failed`ステータスが表示され、オペレータが自力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コンテナのログを追跡すると、問題がどこにあるかがわかります。たとえば、エア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という問題が考えられます。

Tridentのインストールが失敗した理由を理解するには、`TridentOrchestrator`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ています。各KubernetesクラスタにはTridentのインスタンスを1つしか配置できないため、オペレータは、任意の時点でアクティブな `TridentOrchestrator`が1つだけ存在するようにします。

さらに、Tridentポッドのステータスを確認することで、何かが正常でない場合にそれを知らせることがよくあります。

```
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取得されなかったため、podを完全に初期化できないことがはっきりとわかります。

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。または、 `TridentOrchestrator`を削除して、修正された正確な定義で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`

何が問題だったのかを解明するために、`-d`引数を使用してインストーラーを再度実行できます。これによりデバッグモードがオンになり、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します：

```
./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および CRD を完全に削除

Trident と作成されたすべての CRD および関連するカスタムリソースを完全に削除できます。

これを元に戻すことはできません。Trident を完全に新規インストールする場合以外は、これを実行しないでください。CRD を削除せずに Trident をアンインストールするには、"Tridentのアンインストール"を参照してください。

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：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

Helm

Trident をアンインストールし、Helm を使用して 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>

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
```

Kubernetes 1.26 の RWX raw ブロック名前空間での NVMe ノードのアンステージング失敗

Kubernetes 1.26 を実行している場合、RWX 生のブロック名前空間で 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可能性があります。次のシナリオは、失敗に対する回避策を提供します。あるいは、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を削除しました

Trident で管理されている名前空間（NVMe 永続ボリューム）がポッドに接続されているシナリオを考えてみます。ONTAP バックエンドから直接名前空間を削除すると、ポッドを削除しようとした後、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

それぞれのノードから永続ボリューム（その名前空間に対応）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
完全な機能を復元するには、dataLIFS を起動します。

名前空間マッピングを削除しました

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
`hostNQN`をサブシステムに追加し直します。

NFSv4.2クライアントは、「v4.2-xattrs」が有効になっていることを期待している場合、ONTAPのアップグレード後に「invalid argument」を報告します

ONTAP のアップグレード後、NFSv4.2 クライアントが NFSv4.2 エクスポートをマウントしようとすると「無効な引数」エラーを報告する場合があります。この問題は、SVM で `v4.2-xattrs`オプションが有効になっていない場合に発生します。.回避策：SVM で `v4.2-xattrs`オプションを有効にするか、ONTAP 9.12.1 以降にアップグレードしてください。このバージョンでは、このオプションがデフォルトで有効になっています。

サポート

NetApp は、さまざまな方法で Trident のサポートを提供しています。ナレッジベース（KB）記事や Discord チャネルなど、広範な無料のセルフサポートオプションが 24 時間年中無休でご利用いただけます。

Tridentサポートライフサイクル

Trident 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ます。"NetApp ソフトウェアバージョンの定義のサポート"を参照してください。

完全サポート
: Tridentは、リリース日から12か月間の完全サポートを提供します。

限定的なサポート
: Trident は、リリース日から 13 ～ 24 か月間、限定サポートを提供します。

セルフサポート
: Tridentのドキュメントは、リリース日から25～36カ月間利用できます。

version	完全サポート	限定的なサポート	セルフサポート
"25.10"	2026年10月	2027年10月	2028年10月
"25.06"	2026年6月	2027年6月	2028年6月
"25.02"	2026年2月	2027年2月	2028年2月
"24.10"	—	2026年10月	2027年10月
"24.06"	—	2026年6月	2027年6月
"24.02"	—	2026年2月	2027年2月
"23.10"	—	—	2026年10月
"23.07"	—	—	2026年7月
"23.04"	—	—	2026年4月
"23.01"	—	—	2026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、 "NetApp Knowledgebase（ログインが必要です）"を参照してください。

コミュニティサポート

"Discordチャンネル"には、コンテナユーザー（Trident開発者を含む）の活発なパブリックコミュニティがあります。ここは、プロジェクトに関する一般的な質問をしたり、同じ考えを持つ仲間と関連トピックについて話し合ったりするのに最適な場所です。

NetApp テクニカルサポート

Tridentのヘルプについては、 `tridentctl logs -a -n trident`を使用してサポートバンドルを作成し、 `NetApp Support <Getting Help>`に送信してください。

詳細情報

- "Trident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、こちらをご覧ください。

概要

Tridentは、Kubernetesクラスタ内およびストレージバックエンドとの通信にさまざまなポートを使用します。以下は、主要なポート、その目的、およびセキュリティに関する考慮事項の概要です。

- アウトバウンドフォーカス：Kubernetesノード（コントローラとワーカー）は主にストレージLIF/IPへのトラフィックを開始するため、iptablesルールではこれらのポート上のノードIPから特定のストレージIPへのアウトバウンドを許可する必要があります。広範な「any-to-any」ルールは避けてください。
- 受信制限：内部Tridentポートをクラスタ内部トラフィックに制限します（たとえば、CalicoなどのCNIを使用）。ホストファイアウォール上で不要な受信露出はありません。
- プロトコルセキュリティ：
  - 可能な場合は TCP を使用します（信頼性が高くなります）。
  - 機密の場合はiSCSIにCHAP/IPsecを有効にし、管理にはTLS/HTTPSを有効にします（ポート443/8443）。
  - NFSv4（Tridentのデフォルト）の場合、必要ない場合はUDP/古いNFSv3ポート（例：4045～4049）を削除します。
  - 信頼できるサブネットに制限し、Prometheus（オプションのポート8001）などのツールを使用して監視します。

コントローラノードのポート

これらのポートは主にTridentオペレータ（バックエンド管理）用です。すべての内部ポートはポッドレベルです。ホストファイアウォールがCNIに干渉する場合にのみノードで許可します。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 8000	受信/送信（クラスタ内部）	Trident RESTサーバー（オペレーターとコントローラー間の通信）	すべて	ポッドCIDRに制限、外部への公開はありません。
TCP 8443	受信/送信（クラスタ内部）	バックチャネル HTTPS（安全な内部 API）	すべて	TLS暗号化。使用する場合はKubernetesサービスメッシュに制限されます。
TCP 8001	受信（クラスタ内部、オプション）	Prometheusメトリクス	すべて	監視ツール（RBACの使用など）にのみ公開し、使用しない場合は無効にします。
TCP 443	アウトバウンド	HTTPSからONTAP SVM/クラスタ管理LIF	ONTAP（すべて）、ANF	TLS証明書の検証が必要です。管理LIF IPのみに制限します。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 8443	アウトバウンド	HTTPSからEシリーズWeb Services Proxy	Eシリーズ（iSCSI）	デフォルトREST API；証明書を使用します。バックエンドYAMLで構成可能です。

ワーカーノードのポート

これらのポートは、CSI ノードデーモンセットとポッドマウント用です。データポートはストレージデータ LIF へのアウトバウンドです。NFSv3 を使用する場合は NFSv3 エクストラを含めます（NFSv4 の場合はオプション）。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 17546	受信（ポッドへのローカル）	CSI ノードの liveness / readiness プローブ	すべて	設定可能（--probe-port）；ホストの競合がないことを確認する；ローカルのみ。
TCP 8000	受信/送信（クラスタ内部）	Trident REST サーバー	すべて	上記と同様、pod-internal。
TCP 8443	受信/送信（クラスタ内部）	バックチャネル HTTPS	すべて	上記の通りです。
TCP 8001	受信（クラスタ内部、オプション）	Prometheusメトリクス	すべて	上記の通りです。
TCP 443	アウトバウンド	HTTPSからONTAP SVM/クラスタ管理LIF	ONTAP（すべて）、ANF	上記と同様、検出に使用されます。
TCP 8443	アウトバウンド	HTTPSからEシリーズWeb Services Proxy	Eシリーズ（iSCSI）	上記の通りです。
TCP/UDP 111	アウトバウンド	RPCBIND／portmapper	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	v3では必須、v4（ファイアウォールオフロード）ではオプション、NFSv4のみを使用する場合は制限されます。
TCP/UDP 2049	アウトバウンド	NFSデーモン	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	コアデータ、よく知られている、信頼性のためにTCPを使用。
TCP/UDP 635	アウトバウンド	マウントデーモン	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	マウント。双方向コールバックが可能です（必要に応じて着信の一時コールを許可します）。
UDP 4045	アウトバウンド	NFSロックマネージャー（nlockmgr）	ONTAP-NAS（NFSv3）	ファイルロック。v4（pNFSハンドル）の場合はスキップ。UDP のみ。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
UDP 4046	アウトバウンド	NFSステータスモニター（statd）	ONTAP-NAS（NFSv3）	通知。コールバックには受信一時ポート（1024～65535）が必要になる場合があります。
UDP 4049	アウトバウンド	NFSクォータデーモン（rquotad）	ONTAP-NAS（NFSv3）	クォータ。v4の場合はスキップします。
TCP 3260	アウトバウンド	iSCSIターゲット（検出/データ/CHAP）	ONTAP-SAN（iSCSI）、Eシリーズ（iSCSI）	既知のポート。このポートでCHAP 認証を実行。セキュリティのために相互 CHAP を有効にします。
TCP 445	アウトバウンド	SMB / CIFS	ONTAP-NAS（SMB）、ANF（SMB）	よく知られている、暗号化されたSMB3を使用する（Tridentアノテーション netapp.io/smb-encryption=true）。
TCP/UDP 88（オプション）	アウトバウンド	Kerberos認証	ONTAP（NFS/SMB/iSCSI と Kerb）	Kerberos を使用する場合（デフォルトではない）、ストレージではなく AD サーバーに接続します。
TCP/UDP 389（オプション）	アウトバウンド	LDAP	ONTAP（LDAP を使用した NFS/SMB）	同様に、名前解決/認証の場合、ADに制限します。

ライブネス/レディネスプローブポートは、インストール中に `--probe-port`フラグを使用して変更できます。このポートがワーカーノード上の別のプロセスによって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"は Trident REST API と対話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を使用する場合

REST API は、Kubernetes 以外のデプロイメントで 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便利です。

セキュリティ強化のため、Trident `REST API`はポッド内で実行する場合、デフォルトでは localhost に制限されます。この動作を変更するには、Tridentの `-address`引数をポッド構成内で設定する必要があります。

REST APIの使用

これらの API の呼び出し例については、デバッグ(`-d`フラグを渡してください。詳細については、"tridentctl を使用してTridentを管理する"を参照してください。

API は次のように動作します：

GET

`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>`

名前付きオブジェクトの詳細を取得します。

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、"tridentctlを使用してTridentを管理する"を参照してください。
- オブジェクトがすでに存在する場合、動作は異なります。バックエンドは既存のオブジェクトを更新しますが、他のすべてのオブジェクトタイプは操作に失敗します。

DELETE

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`

名前付き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たボリュームは引き続き存在するため、これらは個別に削除する必要があります。詳細については、"tridentctlを使用してTridentを管理する"を参照してください。

コマンドラインオプション

Tridentは、Tridentオーケストレーター用のいくつかのコマンドラインオプションを公開します。これらのオプションを使用して、デプロイメントを変更できます。

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`を使用して、Kubernetesサポートを有効にします。これを設定すると、Tridentは、含まれているポッドの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、Tridentがサービスアカウントが有効になっているKubernetesクラスタ内の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 `-k8s_pod`を使用して、Kubernetesサポートを有効にします。指定すると、Tridentは、提供された安全でないアドレスとポートを使用してKubernetes APIサーバーに接続します。これにより、Tridentをポッドの外部に導入できますが、APIサーバーへの安全でない接続のみがサポートされます。安全に接続するには、 `-k8s_pod`オプションを使用してポッド内にTridentを導入します。

Docker

-volume_driver <name>

Docker プラグインを登録するときに使用するドライバー名。デフォルトは `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このパスを指定する必要があります。

REST

-address <ip-or-host>

Trident の REST サーバーがリッスンするアドレスを指定します。デフォルトは localhost です。localhost でリッスンし、Kubernetes ポッド内で実行している場合、ポッドの外部から REST インターフェイスに直接アクセスすることはできません。 `-address ""`を使用して、ポッド IP アドレスから REST インターフェイスにアクセスできるようにします。

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）でのみリッスンおよびサービスを提供するように構成できます。

-port <port-number>

Trident の REST サーバーがリッ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とTridentオブジェクト

Kubernetes と Trident を使用して REST API でリソース オブジェクトを読み書きすることで対話できます。Kubernetes と Trident 、 Trident とストレージ、および Kubernetes とストレージの関係を規定するリソース 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一部は Kubernetes を通じて管理され、その他は Trident を通じて管理されます。

オブジェクトは互い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、相互作用の仕方を理解する最も簡単な方法は、Kubernetes ユーザーからの単一のストレージ要求を追跡することです：

1. ユーザーは、管理者によって事前に設定されたKubernetes `StorageClass`から、特定のサイズの新しい`PersistentVolume`を要求する `PersistentVolumeClaim`を作成します。
2. Kubernetes `StorageClass`は、Trident をプロビジョナとして識別し、要求されたクラスのボリュームを Trident がプロビジョニングする方法を指示するパラメータを含みます。
3. Tridentは、同じ名前を持つ独自の `StorageClass`を参照し、クラスのボリュームのプロビジョニングに使用できる、一致する `Backends`と `StoragePools`を識別します。
4. Tridentは、一致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でボリュームの検索、マウント、および処理方法をKubernetesに指示する`PersistentVolume`と、 `PersistentVolume`と実際のストレージ間の関係を保持する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。`PersistentVolumeClaim`を含むポッドは、実行されるどのホストでもそのPersistentVolumeをマウントします。
6. ユーザーは、既存のPVCの `VolumeSnapshot`を作成し、Tridentを指す `VolumeSnapshotClass`を使用します。
7. Trident は PVC に関連付けられているボリュームを識別し、そのバックエンドにボリュームのスナップショットを作成します。また、 `VolumeSnapshotContent`を作成し、 Kubernetes にスナップショットを識別する方法を指示します。
8. ユーザーは、 `PersistentVolumeClaim`を作成し、 `VolumeSnapshot`をソースとして使用できます。
9. Tridentは必要なスナップショットを識別し、 `PersistentVolume`と `Volume`の作成に関連する同じ一連の手順を実行します。

Kubernetes オブジェクトの詳細については、Kubernetes ドキュメントの "永続ボリューム"セクションをお読みになることを強くお勧めします。

Kubernetes `PersistentVolumeClaim` オブジェクト

Kubernetes `PersistentVolumeClaim` オブジェクトは、Kubernetes クラスタ ユーザによるストレージの要求です。

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を上書きする場合、Tridentではユーザーは次のボリューム固有の注釈を指定できます：

注釈	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	fileSystem	ontap-san、solidfire-san、ontap-san-economy
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas、ontap-san、solidfire-san、azure-netapp-files、ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ontap-nas、ontap-san

注釈	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/protocol	プロトコル	any
trident.netapp.io/exportPolicy	exportPolicy	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	snapshotPolicy	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotReserve	snapshotReserve	ontap-nas、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotDirectory	snapshotDirectory	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/blockSize	blockSize	solidfire-san
trident.netapp.io/skipRecoveryQueue	skipRecoveryQueue	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、ontap-san-economy

作成されたPVに `Delete`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を削除すると）、PVとバッキングボリュームの両方を削除します。削除アクションが失敗した場合、TridentはPVをそのようにマークし、操作が成功するかPVが手動で削除されるまで定期的に操作を再試行します。PVが `Retain`ポリシーを使用している場合、Tridentはこれを無視し、管理者がKubernetesとバックエンドからこれをクリーンアップし、ボリュームを削除する前にバックアップまたは検査できるようにすることを想定します。PVを削除しても、Tridentはバッキングボリュームを削除しないことに注意してください。REST API(`tridentctl`を使用して削除する必要があります。

Trident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をサポートしています。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を複製できます。このようにして、PVのポイントインタイムコピーをスナップショットの形式でKubernetesに公開できます。スナップショットを使用して新しいPVを作成できます。 `On-Demand Volume Snapshots`を参照して、これがどのように機能するかを確認してください。

Tridentは、クローンを作成するための `cloneFromPVC`および `splitOnClone`アノテーションも提供します。これらのアノテーションを使用すると、CSI実装を使用せずにPVCをクローニングできます。

例：ユーザーがすでにPVC `mysql`を持っている場合、アノテーション `trident.netapp.io/cloneFromPVC: mysql`を使用して、新しいPVC `mysqlclone`を作成できます。このアノテーションが設定されていると、Tridentはmysql PVCに対応するボリュームをクローンし、新規にボリュームをプロビジョニングするのではなく複製します。

次の点を考慮してください：

- NetApp では、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ています。
- PVC とそのクローンは同じ Kubernetes ネームスペースにあり、同じストレージクラスを持つ必要があります。
- `ontap-nas`および `ontap-san`ドライバーを使用する場合、 `trident.netapp.io/splitOnClone`PVCアノテーションを `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.netapp.io/splitOnClone`が `true`に設定されている場合、Tridentはクローンボリュームを親ボリュームから分離し、その結果、クローンボリュームのライフサイクルが親ボリュームから完全に切り離され

ますが、一部のストレージ効率が失われます。 `trident.netapp.io/splitOnClone`を設定しない、または`false`に設定すると、親ボリュームとクローンボリューム間に依存関係が生じ、クローンが先に削除されない限り親ボリュームを削除できなくなる代わりに、バックエンドでのスペース消費が削減されます。クローンの分離が有効なシナリオとしては、空のデータベースボリュームをクローンし、そのボリュームとクローンが大きく分岐し、ONTAPが提供するストレージ効率の恩恵を受けないことが予想される場合が挙げられます。

`sample-input`ディレクトリには、Tridentで使用するPVC
定義の例が含まれています。Tridentボリュームに関連するパラメータと設定の詳細については
、を参照してください。

Kubernetes `PersistentVolume`オブジェクト

Kubernetes `PersistentVolume`オブジェクトは、Kubernetesクラスタで使用できるストレージの一部を表します。これには、それを使用するポッドから独立したライフサイクルがあります。

Tridentは、プロビジョニングしたボリュームに基づいて `PersistentVolume`オブジェクトを作成し、Kubernetesクラスタに自動的に登録します。自分で管理する必要はありません。

Tridentベースの `StorageClass`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、そのボリュームの新しい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を構成する際、Tridentは以下のルールに従います：

- Tridentは、Kubernetesの PV 名と、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、名前がその範囲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可能な限り一致しますが、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 `StorageClass`オブジェクトは `PersistentVolumeClaims`で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は、使用するプロビジョナーを識別し、プロビジョナーが理解できる用語でそのプロパティセットを定義します。

これは、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 1 つです。もう一つは Trident バックエンドオブジェクトです。

Tridentを使用するKubernetes `StorageClass`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固有のもので、クラスのボリュームをプロビジョニングする方法を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	タイプ	必須	概要
attributes	map[string]string	いいえ	下記の属性セクションを参照してください
storagePools	map[string]StringList	いいえ	バックエンド名と内部のストレージプールのリストのマッピング
additionalStoragePools	map[string]StringList	いいえ	バックエンド名と内部のストレージプールのリストのマッピング
excludeStoragePools	map[string]StringList	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可能な値は、ストレージプール選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択属性

これらのパラメータは、特定のタイプのボリュームをプロビジョニングするためにどのTrident管理ストレージプールを使用するかを決定します。

属性	タイプ	値	オファー	要求	サポート対象
メディア[1]	string	HDD、ハイブリッド、SSD	プールにはこのタイプのメディアが含まれます。ハイブリッドとは両方を意味します	指定されたメディアタイプ	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san

属性	タイプ	値	オファー	要求	サポート対象
provisioningType	string	薄い、厚い	プールはこのプロビジョニング方法をサポートしています	プロビジョニング方法が指定されました	thick：すべての ONTAP、thin：すべての ONTAP および solidfire-san
backendType	string	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、azure-netapp-files、ontap-san-economy	プールはこのタイプのバックエンドに属します	バックエンドが指定されました	すべてのドライバー
Snapshot	ブール値	true、false	プールは Snapshot 付きのボリュームをサポートします	スナップショットが有効になっているボリューム	ontap-nas 、ontap-san 、solidfire-san
クローン	ブール値	true、false	プールはボリュームのクローン作成をサポート	クローンが有効なボリューム	ontap-nas 、ontap-san 、solidfire-san
暗号化	ブール値	true、false	プールは暗号化されたボリュームをサポートします	暗号化が有効になっているボリューム	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroups、ontap-san
IOPS	int	正の整数	プールはこの範囲の IOPS を保証できる	ボリュームで保証されるIOPS	solidfire-san

[1]: ONTAP Select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すると、シックプロビジョニングされたボリュームが生成されます。ただし、Elementストレージプールは、要求された値ではなく、提供されたIOPSの最小値と最大値を使用して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にのみ使用されます。

理想的には、 `attributes`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品質をモデル化できます。Tridentは、指定した `attributes`の_すべて_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したプールを自動的に選択できない場合は、 `storagePools`および `additionalStoragePools`パラメータを使用して、プールをさらに絞り込んだり、特定のプールのセットを選択したりできます。

`storagePools`パラメータを使用して、指定された`attributes`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、Tridentは、プロビジョニングのために `attributes`パラメータと`storagePools`パラメータによって識別されるプールの共通部分を使用します。いずれかのパラメータを単独で使用することも、両方を一緒に使用することもできます。

`additionalStoragePools`パラメータを使用して、 `attributes`および`storagePools`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。

`storagePools`および `additionalStoragePools`パラメータでは、各エントリは`<backend>:<storagePoolList>`の形式をとります。ここで、`<storagePoolList>`は指定され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。これらのリストは、バックエンドとリスト値の両方に対して正規表現値を受け入れます。 `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、Trident による動的プロビジョニング中のストレージプール/バックエンドの選択には影響しません。代わりに、これらの属性は 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が必要になる場合があります。

属性	タイプ	値	概要	関連するドライバー	**Kubernetes**バージョン
fsType	string	ext4、ext3、xfs	ブロックボリュームのファイルシステムタイプ	solidfire-san、ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、ontap-san-economy	すべて

属性	タイプ	値	概要	関連するドライバー	**Kubernetes**バージョン
allowVolumeExpansion	ブーリアン	true、false	PVC サイズの拡張のサポートを有効または無効にする	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、ontap-san-economy、solidfire-san、azure-netapp-files	1.11+
volumeBindingMode	string	即座、WaitForFirstConsumer	ボリュームバインディングと動的プロビジョニングを実行するタイミングを選択する	すべて	1.19 - 1.26

- `fsType` パラメータは、SAN LUN用の希望するファイルシステムタイプを制御するために使用されます。さらに、Kubernetesはストレージクラス内に `fsType` が存在することでファイルシステムが存在することを示します。ボリュームの所有権は、 `fsGroup` セキュリティコンテキストをポッドで使用し、 `fsType` が設定されている場合のみ制御できます。"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する" を参照して、 `fsGroup` コンテキストを使用したボリューム所有権の設定概要をご覧ください。Kubernetesは、次の場合にのみ `fsGroup` の値を適用します：

  - `fsType`がストレージクラスに設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバーの場合、NFS エクスポートの一部として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。 `nfs`または null 以外の値に設定できます。

- ボリューム拡張の詳細については、"ボリュームを拡張する"を参照してください。
- Tridentインストーラバンドルは、`sample-input/storage-class-*.yaml`でTridentと併用するためのストレージクラス定義例をいくつか提供しています。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 `StorageClasses`に類似しています。これらは複数のストレージ クラスを定義するのに役立ち、ボリューム スナップショットによって参照されて、スナップショットを必要なスナップショット クラスに関連付けます。各ボリューム スナップショットは、単一のボリューム スナップショット クラスに関連付けられます。

`VolumeSnapshotClass`は、`Snapshot`を作成するために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SnapshotのリクエストがTridentによって処理されることをKubernetesに指定します。 `deletionPolicy`は、Snapshotを削除する必要があるときに実行するアクションを指定します。 `deletionPolicy`が `Delete`に設定されている場合、Snapshotが削除されると、ボリュームSnapshotオブジェクトとストレージクラスター上の基礎となるSnapshotが削除されます。または、 `Retain`に設定すると、 `VolumeSnapshotContent`と物理Snapshotが保持されます。

Kubernetes VolumeSnapshot オブジェクト

Kubernetes `VolumeSnapshot`オブジェクトは、ボリュームのSnapshotを作成する要求です。PVCがボリュームに対するユーザーからの要求を表すのと同様に、ボリュームSnapshotは既存のPVCのSnapshotを作成するためのユーザーからの要求です。

ボリュームスナップショットのリクエストが届くと、Tridentはバックエンドのボリュームのスナップショットの作成を自動的に管理し、一意の`VolumeSnapshotContent`オブジェクトを作成し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を作成する際にその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から独立しています。ソースPVCが削除された後もスナップショットは保持されます。スナップショットが関連付けられているPVCを削除する場合、TridentはこのPVCのバッキング ボリュームを*削除中*状態にマークしますが、完全には削除しません。関連付けられているすべてのスナップショットが削除されると、ボリュームは削除されます。

Kubernetes VolumeSnapshotContent オブジェクト

Kubernetes `VolumeSnapshotContent`オブジェクトは、すでにプロビジョニングされた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`に類似しており、ストレージクラスタ上にプロビジョニングされたスナップショットを示します。 `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、 `VolumeSnapshotContent`オブジェクトは、スナップショットの作成を要求した `VolumeSnapshot`オブジェクトとの1対1のマッピングを維持します。

`VolumeSnapshotContent`オブジェクトには、 `snapshotHandle`などのSnapshotを一意に識別する詳細が含まれています。 `snapshotHandle`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値です。

スナップショットのリクエストが来ると、Tridentはバックエンドにスナップショットを作成します。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、スナップショットをKubernetes APIに公開します。

通常、 `VolumeSnapshotContent`オブジェクトを管理する必要はありません。例外は、Tridentの外部で作成された"ボリューム Snapshot をインポートする"を使用する場合です。

Kubernetes `VolumeGroupSnapshotClass` オブジェクト

Kubernetes `VolumeGroupSnapshotClass`オブジェクトは `VolumeSnapshotClass`に類似しています。これらは複数のストレージ クラスを定義するのに役立ち、ボリューム グループ Snapshotによって参照されて、Snapshotを必要なSnapshotクラスに関連付けます。各ボリューム グループ Snapshotは、単一のボリューム グループ Snapshotクラスに関連付けられます。

`VolumeGroupSnapshotClass`は、スナップショットのグループを作成するために管理者が定義する必要があります。ボリューム グループ スナップショット
クラスは、次の定義で作成されます：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

`driver`は、 `csi-group-snap-class`クラスのボリュームグループSnapshot
の要求がTridentによって処理されることをKubernetesに指定します。
`deletionPolicy`は、グループSnapshotを削除する必要があるときに実行するアクションを指定します。 `deletionPolicy`が `Delete`に設定されている場合、
Snapshotが削除されると、ボリュームグループSnapshotオブジェクトとストレージクラスタ上の基盤となるSnapshotが削除されます。または、 `Retain`に設定すると、
`VolumeGroupSnapshotContent`と物理Snapshotが保持されます。

Kubernetes `VolumeGroupSnapshot` オブジェクト

Kubernetes `VolumeGroupSnapshot`オブジェクトは、複数の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ーの要求を表すのと同様に、ボリューム グループ スナップショットは、既存のPVCのスナップショットを作成するためのユーザーの要求です。

ボリュームグループのスナップショット要求が来ると、Tridentはバックエンドのボリュームのグループスナップショットの作成を自動的に管理し、一意の `VolumeGroupSnapshotContent`オブジェクトを作成し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を作成するときにそのスナップショットをDataSourceとして使用できます。

VolumeGroupSnapshotのライフサイクルは、ソースPVCから独立しています。ソースPVCが削除された後もスナップショットは保持されます。スナップショットが関連付けられているPVCを削除する場合、TridentはこのPVCのバッキングボリュームを*削除中*状態にマークしますが、完全には削除しません。関連するすべてのスナップショットが削除されると、ボリュームグループのスナップショットも削除されます。

Kubernetes `VolumeGroupSnapshotContent` オブジェクト

Kubernetes `VolumeGroupSnapshotContent`オブジェクトは、すでにプロビジョニングされたボリュームから取得されたグループSnapshotを表します。これは `PersistentVolume`に類似しており、ストレージ クラスター上にプロビジョニングされたSnapshotを示します。 `PersistentVolumeClaim`および `PersistentVolume`オブジェクトと同様に、Snapshotが作成されると、 `VolumeSnapshotContent`オブジェクトは、Snapshotの作成を要求した `VolumeSnapshot`オブジェクトとの1対1のマッピングを維持します。

`VolumeGroupSnapshotContent`オブジェクトには、`volumeGroupSnapshotHandle`やストレージシステムに存在する個々の`volumeSnapshotHandles`など、スナップショットグループを識別する詳細が含まれます。

スナップショットのリクエストが来ると、Tridentはバックエンドにボリューム グループのスナップショットを作成します。ボリューム グループのスナップショットが作成されると、Tridentは `VolumeGroupSnapshotContent`オブジェクトを構成し、スナップショットをKubernetes APIに公開します。

Kubernetes `CustomResourceDefinition`オブジェクト

Kubernetes カスタム リソースは、管理者によって定義され、類似のオブジェクトをグループ化するために使用される Kubernetes API のエンドポイントです。Kubernetes は、オブジェクトのコレクションを保存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は、以下を実行することで取得できます `kubectl get crds`。

カスタム リソース定義（CRD）とそれに関連付けられたオブジェクト メタデータは、Kubernetesによってメタデータ ストアに保存されます。これにより、Trident用の別のストアが不要になります。

Tridentは、 `CustomResourceDefinition`オブジェクトを使用して、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オブジェクトのアイデンティティを保持します。これらのオブジェクトはTridentによって管理されます。さらに、CSIボリュームSnapshotフレームワークでは、ボリュームSnapshotを定義するために必要ないくつかのCRDが導入されています。

CRDはKubernetesの構造です。上記で定義され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 `tridentbackends` CRDオブジェクトがKubernetesで使用するために作成されます。

以下に、TridentのCRDについて留意すべき点をいくつか示します：

- Tridentがインストールされると、CRDのセットが作成され、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。
- `tridentctl uninstall`コマンドを使用してTridentをアンインストールすると、Tridentポッドは削除されますが、作成されたCRDはクリーンアップされません。"Tridentのアンインストール"を参照して、Tridentを完全に削除して最初から再構成する方法を理解してください。

Trident `StorageClass`オブジェクト

Tridentは、プロビジョナー フィールドで `csi.trident.netapp.io`を指定するKubernetes `StorageClass`オブジェクト用に対応するストレージクラスを作成します。ストレージクラス名は、それが表すKubernetes `StorageClass`オブジェクトの名前と一致します。

> Kubernetesでは、Tridentをプロビジョナーとして使用するKubernetes `StorageClass`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、ボリュームの要件のセットから構成されます。Tridentはこれらの要件を各ストレージ プールに存在する属性と照合します。一致する場合、そのストレージ プールはそのストレージ クラスを使用してボリュームをプロビジョニングするため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構成を作成できます。ただし、Kubernetesのデプロイメントの場合、新しいKubernetes `StorageClass`オブジェクトを登録するときに作成されることを想定しています。

Tridentバックエンドオブジェクト

バックエンドは、Tridentがボリュームをプロビジョニングするストレージプロバイダーを表します。単一のTridentインスタンスは任意の数のバックエンドを管理できます。

> これは、自分で作成して管理する 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 `StorageClass` オブジェクトです。

これらのオブジェクトの構築方法の詳細については、"バックエンドの設定"を参照してください。

Trident `StoragePool`オブジェクト

ストレージ プールは、各バックエンドでプロビジョニングできる個別の場所を表します。ONTAP の場合、これらは SVM のアグリゲートに相当します。NetApp HCI/SolidFire の場合、これらは管理者が指定した QoS 帯域に対応します。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一連の個別のストレージ属性があります。

ここでの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、管理されます。

Trident `Volume`オブジェクト

ボリュームはプロビジョニングの基本単位であり、NFS 共有、iSCSI、FC LUN などのバックエンド エンドポイントで構成されます。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、ボリュームをプロビジョニングできる場所とサイズを決定するストレージ クラスがあることを確認します。

> - Kubernetes では、これらのオブジェクトは自動的に管理されます。それらを表示して、Trident がプロビジョニングした内容を確認できます。
> - 関連するスナップショットを持つPVを削除する場合、対応するTridentボリュームは*削除中*状態に更新されます。Tridentボリュームを削除するには、ボリュームのスナップショット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	タイプ	必須	概要
version	string	いいえ	Trident APIのバージョン（「1」）
名前	string	はい	作成するボリュームの名前
storageClass	string	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	string	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	string	いいえ	使用するプロトコルタイプ：「file」または「block」
internalName	string	いいえ	ストレージシステム上のオブジェクトの名前。Tridentによって生成されます
cloneSourceVolume	string	いいえ	ontap (nas、san) & solidfire-*：クローン元のボリュームの名前
splitOnClone	string	いいえ	ONTAP (NAS、SAN): クローンを親から分離する
snapshotPolicy	string	いいえ	ONTAP-*：使用するSnapshotポリシー
snapshotReserve	string	いいえ	ontap-*：Snapshot用に予約されているボリュームの割合
exportPolicy	string	いいえ	ontap-nas*：使用するエクスポートポリシー
snapshotDirectory	ブール値	いいえ	ontap-nas*：Snapshotディレクトリが表示されるかどうか
unixPermissions	string	いいえ	ontap-nas*：初期UNIX権限
blockSize	string	いいえ	SolidFire-*：ブロック/セクターサイズ
fileSystem	string	いいえ	ファイルシステムの種類
skipRecoveryQueue	string	いいえ	ボリュームの削除中は、ストレージ内のリカバリキューをバイパスし、ボリュームを直ちに削除します。

Tridentは、ボリュームを作成するときに `internalName`を生成します。これは2つのステップで構成されま

す。まず、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設定のプレフィックス）をボリューム名の前に付加して、 `<prefix>-<volume-name>`という形式の名前にします。次に、バックエンドで許可されていない文字を置き換えて、名前をサニタイズします。ONTAPバックエンドの場合、ハイフンをアンダースコアに置き換えます（したがって、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。

ボリューム構成を使用すると、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の導入では、ほとんどのユーザーが標準のKubernetes `PersistentVolumeClaim`方法を使用することが想定されています。Tridentは、プロビジョニング プロセスの一部として、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

スナップショットはボリュームのポイントインタイム コピーであり、新しいボリュームのプロビジョニングや状態の復元に使用できます。Kubernetesでは、これらは `VolumeSnapshotContent`オブジェクトに直接対応します。各スナップショットは、スナップショットのデータのソースであるボリュームに関連付けられています。

各 `Snapshot`オブジェクトには、以下のプロパティが含まれます：

属性	タイプ	必須	概要
version	文字列	はい	Trident APIのバージョン（「1」）
名前	文字列	はい	Tridentスナップショットオブジェクトの名前
internalName	文字列	はい	ストレージシステム上のTridentスナップショットオブジェクトの名前
volumeName	文字列	はい	Snapshotが作成される永続ボリュームの名前
volumeInternalName	文字列	はい	ストレージシステム上の関連する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は自動的に管理されます。それらを表示して、Tridentがプロビジョニングした内容を確認できます。

Kubernetes `VolumeSnapshot`オブジェクトリクエストが作成されると、`Trident`はバックエンド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で動作します。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 `UID`、`VolumeSnapshot`オブジェクトの`snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`を組み合わせて生成されます。 `volumeName`および`volumeInternalName`は、バックエンドボリュームの詳細を取得することで設定されます。

Trident `ResourceQuota`オブジェクト

Tridentデーモンセットは `system-node-critical`優先度クラス（Kubernetesで利用可能な最高の優先度クラス）を使用します。これにより、Tridentはノードの正常なシャットダウン中にボリュームを識別してクリーンアップでき、リソースの負荷が高いクラスターでは、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ます。

これを実現するために、Tridentは `ResourceQuota`オブジェクトを使用して、Trident daemonsetで「system-node-critical」優先度クラスが満たされるようにします。導入とdaemonsetの作成前に、Tridentは `ResourceQuota`オブジェクトを検索し、検出されない場合は適用します。

デフォルトのResource QuotaとPriority Classをさら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`オブジェクトを設定できます。

以下は、Tridentデーモンセットを優先する`ResourceQuota`オブジェクトの例です。

```
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、"Kubernetes：リソースクォータ"を参照してください。

インストールに失敗した場合はクリーンアップします `ResourceQuota`

`ResourceQuota`オブジェクトの作成後にインストールが失敗するまれなケースでは、まずlink:../trident-managing-k8s/uninstall-trident.html["アンインストール"]を試してから再インストールしてください。

それでも解決しない場合は、手動で `ResourceQuota`オブジェクトを削除してください。

削除 `ResourceQuota`

自分でリソースの割り当てを制御したい場合は、次のコマンドを使用してTrident `ResourceQuota`オブジェクトを削除できます：

```
kubectl delete quota trident-csi -n trident
```

Pod Security Standards（PSS）とSecurity Context Constraints（SCC）

Kubernetes ポッド セキュリティ標準（PSS）とポッド セキュリティ ポリシー（PSP）

は、権限レベルを定義し、ポッドの動作を制限します。OpenShift セキュリティ コンテキスト制約（SCC）も同様に、OpenShift Kubernetes エンジンに固有のポッド制限を定義します。このカスタマイズを提供するために、Trident はインストール中に特定の権限を有効にします。以下のセクションでは、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 Security Policies（PSP）に代わるものです。PSPはKubernetes v1.21で非推奨となり、v1.25で削除されます。詳細については、"Kubernetes：セキュリティ"を参照してください。

必要な **Kubernetes** セキュリティコンテキストと関連フィールド

権限	概要
特権	CSI ではマウントポイントが双方向であることが求められており、Trident ノード ポッドは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、"Kubernetes：マウント伝播"を参照してください。
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、プロセス間通信（IPC）を使用して NFSD と通信します。
ホスト PID	NFS の場合は `rpc-statd`を開始する必要があります。Trident は、NFS ボリュームをマウントする前に `rpc-statd`が実行されているかどうかを判断するためにホストプロセスを照会します。

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` は特権コンテナに対して次の機能を設定します：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccompプロファイルは特権コンテナでは常に「Unconfined」であるため、Tridentでは有効にできません。
SELinux	OpenShift では、特権コンテナは `spc_t`（「Super Privileged Container」）ドメインで実行され、非特権コンテナは `container_t`ドメインで実行されます。 `containerd`では、 `container-selinux`がインストールされている場合、すべてのコンテナは `spc_t`ドメインで実行され、SELinux が事実上無効になります。したがって、Trident はコンテナに `seLinuxOptions`を追加しません。
DAC	特権コンテナは root として実行する必要があります。非特権コンテナは、CSI に必要な Unix ソケットにアクセスするために root として実行されます。

Pod Security Standards（PSS）

ラベル	概要	デフォルト
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	Trident コントローラとノードがインストール名前空間に許可されるようにします。名前空間ラベルを変更しないでください。	`enforce: privileged enforce-version: <version of the current cluster or highest version of PSS tested.>`

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、「作成エラー：...」または「警告：trident-csi-...」が表示される場合があります。このような場合は、 `privileged`の名前空間ラベルが変更されたかどうかを確認してください。変更されている場合は、Tridentを再インストールしてください。

Pod Security Policies（PSP）

フィールド	概要	デフォルト
`allowPrivilegeEscalation`	特権コンテナは特権の昇格を許可する必要があります。	`true`
`allowedCSIDrivers`	Tridentはインライン CSI 一時ボリュームを使用しません。	空
`allowedCapabilities`	非特権Tridentコンテナはデフォルトセット以上の機能を必要とせず、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
`allowedFlexVolumes`	Tridentは"FlexVolumeドライバ"を使用しないため、許可されるボリュームのリストには含まれません。	空
`allowedHostPaths`	Tridentノード ポッドはノードのルート ファイルシステムをマウントするため、このリストを設定する利点はありません。	空
`allowedProcMountTypes`	Tridentは使用しません `ProcMountTypes`	空
`allowedUnsafeSysctls`	Tridentでは、安全でないものは必要ありません `sysctls`。	空
`defaultAddCapabilities`	特権コンテナに機能を追加する必要はありません。	空
`defaultAllowPrivilegeEscalation`	権限昇格の許可はそれぞれのTridentポッドで処理されます。	`false`
`forbiddenSysctls`	`sysctls`は許可されていません。	空
`fsGroup`	Trident コンテナは root として実行されます。	`RunAsAny`

フィールド	概要	デフォルト
hostIPC	NFSボリュームをマウントするには、ホストIPCとの通信が必要です nfsd	true
hostNetwork	iscsiadm では、iSCSI デーモンと通信するためにホストネットワークが必要です。	true
hostPID	ホストPIDは、 `rpc-statd`がノード上で実行されているかどうかを確認するために必要です。	true
hostPorts	Trident はホストポートを使用しません。	空
privileged	ボリュームをマウントするには、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。	true
readOnlyRootFilesystem	Trident ノード ポッドはノード ファイル システムに書き込む必要があります。	false
requiredDropCapabilities	Tridentノードポッドは特権コンテナを実行するため、機能を削除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
seLinux	Tridentは `seLinuxOptions`を設定しません。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を処理する方法に現在違いがあるためです。	空
supplementalGroups	Trident コンテナは root として実行されます。	RunAsAny
volumes	Tridentポッドにはこれらのボリュームプラグインが必要です。	hostPath, projected, emptyDir

Security Context Constraints（SCC）

ラベル	概要	デフォルト
allowHostDirVolumePlugin	Trident ノードポッドは、ノードのルートファイルシステムをマウントします。	true

ラベル	概要	デフォルト
`allowHostIPC`	NFSボリュームをマウントするには、ホストIPCと通信する必要があります `nfsd`。	`true`
`allowHostNetwork`	iscsiadm では、iSCSI デーモンと通信するためにホストネットワークが必要です。	`true`
`allowHostPID`	ホストPIDは、 `rpc-statd`がノード上で実行されているかどうかを確認するために必要です。	`true`
`allowHostPorts`	Trident はホストポートを使用しません。	`false`
`allowPrivilegeEscalation`	特権コンテナは特権の昇格を許可する必要があります。	`true`
`allowPrivilegedContainer`	ボリュームをマウントするには、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。	`true`
`allowedUnsafeSysctls`	Tridentでは、安全でないものは必要ありません `sysctls`。	`none`
`allowedCapabilities`	非特権Tridentコンテナはデフォルトセット以上の機能を必要とせず、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
`defaultAddCapabilities`	特権コンテナに機能を追加する必要はありません。	空
`fsGroup`	Trident コンテナは root として実行されます。	`RunAsAny`
`groups`	このSCCはTrident専用であり、そのユーザーにバインドされています。	空
`readOnlyRootFilesystem`	Trident ノード ポッドはノード ファイル システムに書き込む必要があります。	`false`
`requiredDropCapabilities`	Tridentノードポッドは特権コンテナを実行するため、機能を削除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`
`seLinuxContext`	Tridentは `seLinuxOptions`を設定しません。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を処理する方法に現在違いがあるためです。	空

ラベル	概要	デフォルト
`seccompProfiles`	特権コンテナは常に"Unconfined"で実行されます。	空
`supplementalGroups`	Trident コンテナは root として実行されます。	`RunAsAny`
`users`	このSCCをTrident名前空間のTridentユーザーにバインドするためのエントリが1つ提供されています。	N/A
`volumes`	Tridentポッドにはこれらのボリュームプラグインが必要です。	`hostPath, downwardAPI, projected, emptyDir`

法律上の表示

法的通知から、著作権情報、商標、特許などを確認できます。

著作権

"https://www.netapp.com/company/legal/copyright/"

商標

NetApp、NetAppのロゴ、NetAppの商標一覧のページに掲載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

現在NetAppが所有する特許の一覧は以下のページから閲覧できます。

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ソフトウェアのTridentで使用されているサードパーティの著作権とライセンスは、各リリースの通知ファイル（ https://github.com/NetApp/trident/）で確認できます。