Trident 25.06 ドキュメント

Trident

NetApp
March 05, 2026

目次

Trident 25.06 ドキュメント 1
リリース ノート 2
新着情報 2
25.06.2 の新機能 2
25.06.1 の変更点 2
25.06の変更点 2
25.02.1 の変更点 5
25.02の変更点 5
24.10.1 の変更点 7
24.10の変更点 7
24.06の変更点 9
24.02の変更点 10
23.10の変更点 11
23.07.1 の変更点 11
23.07の変更点 11
23.04の変更点 12
23.01.1 の変更点 13
23.01の変更点 14
22.10の変更点 15
22.07の変更点 16
22.04の変更点 17
22.01.1 の変更点 18
22.01.0 の変更点 18
21.10.1 の変更点 18
21.10.0 の変更点 19
既知の問題 20
詳細情報の参照 21
以前のバージョンのドキュメント 21
既知の問題 21
大きなファイルのResticバックアップの復元が失敗する可能性がある 21
始めましょう 22
Tridentについて学ぶ 22
Tridentについて学ぶ 22
Tridentアーキテクチャ 23
概念 26
Tridentのクイックスタート 30
次の手順 31
要件 31
Tridentに関する重要な情報 31

サポートされているフロントエンド（オーケストレーター） 32
サポートされているバックエンド（ストレージ） 32
KubeVirt と OpenShift Virtualization のTridentサポート 33
機能の要件 33
テスト済みのホストオペレーティングシステム 34
ホスト構成 34
ストレージシステムの構成 35
Tridentポート 35
コンテナイメージと対応するKubernetesバージョン 35
Tridentをインストールする 37
Tridentオペレーターを使用してインストールする 37
tridentctlを使用してインストールする 37
OpenShift認定オペレーターを使用してインストールする 37
Tridentを使用する 38
ワーカーノードを準備する 38
適切なツールの選択 38
ノードサービス検出 38
NFSボリューム 39
iSCSIボリューム 39
NVMe/TCPボリューム 43
SCSI over FCボリューム 44
バックエンドの構成と管理 47
バックエンドを構成する 47
Azure NetApp Files 47
Google Cloud NetApp Volumes 67
Google Cloud バックエンド用のCloud Volumes Serviceを構成する 84
NetApp HCIまたはSolidFireバックエンドを構成する 96
ONTAP SAN ドライバー 101
ONTAP NAS ドライバー 131
Amazon FSx for NetApp ONTAP 167
kubectl でバックエンドを作成する 203
バックエンドを管理する 210
ストレージクラスの作成と管理 220
ストレージクラスを作成する 220
ストレージクラスの管理 223
ボリュームのプロビジョニングと管理 225
ボリュームをプロビジョニングする 225
ボリュームを拡張する 229
輸入量 240
ボリューム名とラベルをカスタマイズする 251
名前空間間でNFSボリュームを共有する 254

名前空間を越えてボリュームを複製する 258
SnapMirrorを使用してボリュームを複製する 260
CSIトポロジを使用する 267
スナップショットの操作 274
ボリュームグループのスナップショットを操作する 282
Tridentの管理と監視 287
Tridentをアップグレード 287
Tridentをアップグレード 287
オペレーターによるアップグレード 288
tridentctl によるアップグレード 293
tridentctl を使用してTrident を管理する 294
コマンドとグローバルフラグ 294
コマンドオプションとフラグ 296
プラグインのサポート 302
モニターTrident 302
概要 302
ステップ1: Prometheusターゲットを定義する 302
ステップ2: Prometheus ServiceMonitorを作成する 303
ステップ3: PromQLでTridentメトリクスをクエリする 303
Trident AutoSupportテレメトリについて学ぶ 304
Tridentメトリックを無効にする 305
Tridentをアンインストールする 305
元のインストール方法を決定する 306
Tridentオペレータのインストールをアンインストールする 306
アンインストール `tridentctl`インストール 307
Docker 用のTrident 308
展開の前提条件 308
要件を確認する 308
NVMeツール 310
FCツール 311
Tridentを展開する 313
Docker マネージド プラグイン方式 (バージョン 1.13/17.03 以降) 313
従来の方法（バージョン1.12以前） 315
システム起動時にTridentを起動する 317
Trident のアップグレードまたはアンインストール 317
Upgrade 318
アンインストール 319
ボリュームの操作 319
ボリュームの作成 319
ボリュームを削除する 320
ボリュームをクローニングする 320

外部で作成されたボリュームにアクセスする 322
ドライバー固有のボリュームオプション 322
ログを収集する 328
トラブルシューティングのためにログを収集する 328
一般的なトラブルシューティングのヒント 329
複数のTridentインスタンスを管理する 329
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) 329
従来の手順（バージョン1.12以前） 330
ストレージ構成オプション 330
グローバル設定オプション 330
ONTAP構成 331
要素ソフトウェア構成 339
既知の問題と制限事項 341
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。 341
ボリューム名は 2 文字以上である必要があります。 342
Docker Swarm には、 Trident があらゆるストレージとドライバーの組み合わせをサポートできない原因となる特定の動作があります。 342
FlexGroupがプロビジョニングされている場合、2 番目のFlexGroup にプロビジョニングされているFlexGroupと共通の 1 つ以上のアグリゲートが存在すると、 ONTAP は2 番目のFlexGroup をプロビジョニングしません。 342
ベストプラクティスと推奨事項 343
導入 343
専用の名前空間にデプロイする 343
クォータと範囲制限を使用してストレージ消費を制御する 343
ストレージ構成 343
プラットフォームの概要 343
ONTAPとCloud Volumes ONTAP のベストプラクティス 343
SolidFireのベストプラクティス 348
さらに詳しい情報はどこで見つかりますか? 350
Tridentを統合 350
ドライバーの選択と展開 350
ストレージクラスの設計 354
仮想プールの設計 355
ボリューム操作 356
メトリクスサービス 359
データ保護とディザスタ リカバリ 360
Tridentの複製と回復 361
SVMのレプリケーションとリカバリ 361
ボリュームの複製と回復 362

スナップショットデータ保護 363
セキュリティ 363
セキュリティ 363
Linux 統合キー設定 (LUKS) 364
Kerberos のインフライト暗号化 370
Trident Protectでアプリケーションを保護する 379
Tridentプロテクトについて学ぶ 379
次の手順 379
Trident Protectをインストールする 379
Tridentプロテクトの要件 379
Trident Protectのインストールと設定 383
Trident Protect CLIプラグインをインストールする 386
Trident Protectのインストールをカスタマイズする 390
Trident Protectの管理 395
Trident Protectの認証とアクセス制御を管理する 395
Trident Protectリソースを監視する 402
Trident Protect サポートバンドルを生成する 407
Tridentプロテクトのアップグレード 409
アプリケーションの管理と保護 410
Trident Protect AppVault オブジェクトを使用してバケットを管理する 410
Trident Protectで管理するアプリケーションを定義する 424
Trident Protectを使用してアプリケーションを保護する 428
アプリケーションを復元する 438
NetApp SnapMirrorとTrident Protectを使用してアプリケーションを複製する 456
Trident Protectを使用してアプリケーションを移行する 472
Trident Protect実行フックを管理する 476
Trident Protectをアンインストールする 488
TridentとTridentプロテクトのブログ 489
Tridentブログ 489
Trident Protectブログ 489
知識とサポート 491
よくある質問 491
一般的な質問 491
Kubernetes クラスターにTrident をインストールして使用する 491
トラブルシューティングとサポート 492
Tridentをアップグレード 494
バックエンドとボリュームを管理する 494
トラブルシューティング 498
一般的なトラブルシューティング 498
オペレーターを使用したTridentの展開は失敗に終わった 500
Trident配備失敗 `tridentctl` 502

TridentとCRDを完全に削除する . . . 502
Kubernetes 1.26 の RWX 生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 . . . 503
ONTAPをアップグレードした後、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にもかかわらず、「無効な引数」を報告します。 . . . 504
サポート . . . 504
Tridentのサポート . . . 504
自立 . . . 505
コミュニティサポート . . . 505
NetAppテクニカルサポート . . . 505
詳細情報 . . . 505
参照 . . . 506
Tridentポート . . . 506
Tridentポート . . . 506
TridentREST API . . . 506
REST APIを使用する場合 . . . 506
REST APIの使用 . . . 506
コマンドラインオプション . . . 507
ロギング . . . 507
Kubernetes . . . 507
Docker . . . 508
REST . . . 508
KubernetesとTridentオブジェクト . . . 508
オブジェクトは互いにどのように相互作用しますか? . . . 508
Kubernetes `PersistentVolumeClaim`オブジェクト . . . 509
Kubernetes `PersistentVolume`オブジェクト . . . 510
Kubernetes `StorageClass`オブジェクト . . . 511
Kubernetes `VolumeSnapshotClass`オブジェクト . . . 514
Kubernetes `VolumeSnapshot`オブジェクト . . . 515
Kubernetes `VolumeSnapshotContent`オブジェクト . . . 515
Kubernetes `VolumeGroupSnapshotClass`オブジェクト . . . 516
Kubernetes `VolumeGroupSnapshot`オブジェクト . . . 516
Kubernetes `VolumeGroupSnapshotContent`オブジェクト . . . 517
Kubernetes `CustomResourceDefinition`オブジェクト . . . 517
Trident `StorageClass`オブジェクト . . . 517
Tridentバックエンドオブジェクト . . . 518
Trident `StoragePool`オブジェクト . . . 518
Trident `Volume`オブジェクト . . . 518
Trident `Snapshot`オブジェクト . . . 520
Trident `ResourceQuota`物体 . . . 520
ポッドセキュリティ標準（PSS）とセキュリティコンテキスト制約（SCC） . . . 521
必要な Kubernetes セキュリティコンテキストと関連フィールド . . . 522

ポッドセキュリティ標準（PSS） 522
ポッドセキュリティポリシー（PSP） 523
セキュリティコンテキスト制約（SCC） 524
法律上の表示 527
著作権 527
商標 527
特許 527
プライバシー ポリシー 527
オープンソース 527

Trident 25.06 ドキュメント

リリース ノート

新着情報

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

その `tridentctl`インストーラーの zip ファイルで提供される Linux 用のバイナリは、テストされサポートされているバージョンです。注意してください `macos`バイナリ提供 `/extras`zip ファイルの一部はテストもサポートもされていません。

25.06.2 の新機能

新機能の概要には、 TridentとTrident Protect の両方のリリースの機能強化、修正、廃止に関する詳細が記載されています。

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発生する既知の問題のため、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プ解除される前に NQN がチェックされない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 クラス間のボリュームのクローン作成をブロックします。
- **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をクローンする際のタイムアウトを増加しました（"問題 #1008" ）。

25.06の変更点

Trident

機能強化

- **Kubernetes**:
  - 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。 `v1beta1` ONTAP-SAN iSCSI ドライバー用のボリューム グループ スナップショット Kubernetes API。見る"ボリュームグループのスナップショットを操作する"。

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

  - iSCSI に加えて、NVMe/TCP 用のONTAP ASA r2 のサポートが追加されました。見るlink:"ONTAP SAN 構成オプションと例"。
  - 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。見る"安全なSMBを有効にする"。
  - 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。
  - 追加した `--allow-discards`LUKS ボリュームを開いて、スペース再利用のための破棄/TRIM コマンドを許可する場合。
  - 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 クリーンアップが強化されました。
  - NVMe ボリュームの接続と切断に対するTridentノードの冪等性が強化されました。
  - 追加した `internalID`ONTAP -SAN-Economy ドライバーのTridentボリューム構成にフィールドを追加します。
  - NVMe バックエンドのSnapMirrorを使用したボリューム レプリケーションのサポートが追加されました。見る"SnapMirrorを使用してボリュームを複製する"。

実験的な機能強化

実稼働環境では使用しないでください。

- [テクニカルプレビュー] Tridentコントローラーの同時操作を、 `--enable-concurrency`機能フラグ。これにより、コントローラーの操作を並行して実行できるようになり、混雑した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。

- [技術プレビュー] 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**:

- 基礎となる 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一致しない可能性がある CSI NodeExpandVolume の問題を修正しました。
- ONTAP-NAS およびONTAP-NAS-Economy ドライバーの重複したエクスポート ポリシーをクリーンアップできない問題を修正しました。
- GCNVボリュームがNFSv3にデフォルト設定される問題を修正 `nfsMountOptions`設定が解除されました。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。もし `nfsMountOptions`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。
- Kustomize を使用してTrident をインストールする際のデプロイメントの問題を修正しました ("第831号" ）。
- 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 の増分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。

- SolidFireバックエンドを使用してボリュームをクローンする際のタイムアウトを修正しました（"問題 #1008" ）。
- 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 を使用しているときにクローン分割操作が同期的に実行される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プロテクト

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) をNetApp SnapMirrorと併用する場合、完全な PVC レプリケーションを回避するために、効率的な再同期とリバース レプリケーションを実行します。
- 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で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、ソース 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、Bound 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がある AppVault で AppVaultContent を取得するときにエラーが表示される問題を修正しました。
- 障害を回避するために、KubeVirt VMSnapshots は復元およびフェイルオーバー操作から除外されます。
- 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 ("983号" ）。
  - 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が回復に失敗する問題を修正しました（"第961号" ）。

25.02の変更点

Trident 25.02 以降、「新機能」の概要には、 TridentとTrident Protect の両方のリリースの機能強化、修正、廃止に関する詳細が記載されています。

Trident

機能強化

- **Kubernetes**:
  - iSCSI 用のONTAP ASA r2 のサポートが追加されました。
  - 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 ボリュームでは、 Tridentによって管理されるボリュームごとのエクスポート ポリシーが利用されるようになりました。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複製のサポートが追加されました。
  - iSCSI 自己修復スキャン修復が強化され、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。

- オープンシフト:
  - ROSA クラスター上の RHCOS の自動 iSCSI ノード準備のサポートが追加されました。
  - ONTAPドライバー用の OpenShift Virtualization のサポートが追加されました。
- ONTAP-SAN ドライバーにファイバー チャネル 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 ("第961号" ）。
- 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リカバリ キューをスキップできるようにするサポートが追加されました。
- 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。
- tridentctl 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（"第965号" ）。
  - 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ためにバックエンド構成の認証情報を修正しました ("913号" ）。
  - Tridentオペレータで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 ("第924号" ）。
  - csi-resizer コンテナの securityContext を追加しました ("976号" ）。

Tridentプロテクト

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

- volumeMode: File および volumeMode: Block (raw デバイス) ストレージの両方に対して、KubeVirt / OpenShift Virtualization VM のバックアップと復元のサポートが追加されました。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 を備えたNetApp SnapMirrorを使用してストレージを複製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プロキシ接続を構成するためのサポートが追加されました。
- 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 のインストール中にセキュリティ コンテキスト制約 (SCC) を構成する機能が追加されました。
- Trident Protect のインストール中に nodeSelector を構成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が追加されました。
- クラスタースコープのリソースを除外できるように ResourceFilter を拡張しました。
- S3 AppVault 認証情報に AWS セッション トークンのサポートが追加されました。
- 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

- ONTAPボリューム リカバリ キューをスキップするために一時ボリュームの管理が改善されました。
- 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 (#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（#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不健全な非 CHAP ポータルのログアウトを削除しました (#961) 。

24.10の変更点

機能強化

- Google Cloud NetApp Volumesドライバが NFS ボリュームで一般提供され、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 を使用したGoogle Cloud NetApp Volumesの Cloud Identity として使用されます。
- 追加した `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て、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す。
- 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。

- 追加した `denyNewVolumePools`ONTAP-NAS-Economy およびONTAP-SAN-Economy ドライバーを既存の Flexvol プールに制限する構成パラメータ。
- すべてのONTAPドライバーにわたる 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、ステージの失敗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 ジェネレーターが追加され、顧客がONTAPでTridentの最小限のロールを作成できるようになりました。
- 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`（"第792号"）。

Kubernetes

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

    "Tridentプロテクトについて詳しくはこちら"。
- 新しいフラグを追加しました `--k8s-api-qps`インストーラーに、 Tridentが Kubernetes API サーバーと通信するために使用する QPS 値を設定します。
- 追加した `--node-prep`Kubernetes クラスター ノード上のストレージ プロトコル依存関係を自動管理するためのインストーラーへのフラグ。 Amazon Linux 2023 iSCSI ストレージプロトコルとの互換性をテストおよび検証済み
- 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、使用時にqtreeごとのエクスポートポリシーを使用します。 `autoExportPolicy`バックエンドオプション。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。 Trident がアクティブなワークロードに影響を与えずにボリュームをすべてのノードから非公開にすると、既存の qtree は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 ドライバーのファイバー チャネル 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 のアドミッション Webhook がTrident Helm のインストールを妨げている問題を修正 ("第839号"）。

- ヘルムチャート値のアフィニティキーを修正しました（"第898号" ）。
- tridentControllerPluginNodeSelector/tridentNodePluginNodeSelector が「true」値で動作しない問題を修正しました ("第899号" ）。
- 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（"901号" ）。

- Windows Server 2019 のサポートが追加されました。
- Tridentリポジトリの `go mod tidy` を修正しました ("号 #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) にブランド名が変更されます。このブランド変更は、 Tridentの機能、サポートされるプラットフォーム、相互運用性には影響しません。

24.06の変更点

機能強化

- 重要: `limitVolumeSize`パラメータにより、 ONTAPエコノミー ドライバの qtree/LUN サイズが制限されるようになりました。新しい `limitVolumePoolSize`これらのドライバーの Flexvol サイズを制御するパラメーター。 （"第341号" ）。
- 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 iSCSI 自己修復機能を追加しました ("号 #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の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コントローラーのユーザー構成のログ設定をTridentノード ポッドに伝播する機能を追加しました。
- ONTAPバージョン 9.15.1 以降では、デフォルトで ONTAPI (ZAPI) の代わりに REST を使用するためのサポートがTridentに追加されました。
- 新しい永続ボリュームのONTAPストレージ バックエンドにカスタム ボリューム名とメタデータのサポートが追加されました。
- 強化された `azure-netapp-files`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、(ANF) ドライバーは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します。
- 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 サポートが追加されました。

Kubernetes

- Kubernetes 1.30 のサポートが追加されました。

- Trident DaemonSet に起動時にゾンビマウントと残留追跡ファイルを消去する機能を追加 ("号 #883" ）。
- PVC注釈を追加しました `trident.netapp.io/luksEncryption`LUKSボリュームを動的にインポートするため（"第849号" ）。
- 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による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（"第892号" ）。
- Tridentコントローラーを変更しました `SecurityContextConstraint`優先する `0`（"号 #887" ）。
- ONTAPドライバーは、20 MiB 未満のボリューム サイズ ("問題[#885" ）。
- ONTAP -SAN ドライバーのサイズ変更操作中にFlexVolボリュームが縮小されないようにTrident を修正しました。
- 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 クラスターにTrident をアドオンとしてインストールするためのサポートが追加されました。
- iSCSI 自己修復を構成および無効化する機能を追加 ("号 #864" ）。
- ONTAPドライバーにAmazon FSxパーソナリティを追加して、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 ("第453号" ）。

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 ("号 #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分割を実行する前に 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 ("第834号" ）。
- 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 ("号 #722" ）。
- ONTAP -NAS-Economy のFlexVol名変換を修正しました。
- ノードを再起動したときの Windows ノード上の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 ドライバーの 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 ("第558号" ）。

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 ("第740号" ）。

23.07の変更点

修正

Kubernetes

- 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Trident のアップグレードを修正しました ("第740号" ）。
- 「transient-trident-version-pod」定義に許容範囲を追加しました（"第795号" ）。

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（"号 #816" ）。
- use-rest=true でONTAPドライバーを使用する際のクォータのサイズ変更を修正しました。
- ontap-san-economy での LUN クローンの作成を修正しました。

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

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（"第817号" ）。

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、バックエンドの状態を更新します（"問題 #801" 、"＃543" ）。
- LUN シリアル番号は、ControllerVolumePublish ワークフロー中に常に取得され、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ロジックを追加しました。
- 正しいマルチパス デバイスがステージングされていないことを確認するための iSCSI ボリュームの追加検証。

実験的な強化

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。強制デタッチはインストール時に有効にする必要があります `--enable-force-detach`Tridentインストーラー フラグ。

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 のクラスタロール権限がバンドル権限と同期するように修正されました ("第799号" ）。

- 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。
- FlexGroupクローンのサポートと SMB ボリュームのボリューム インポートを修正しました。
- Tridentコントローラーがすぐにシャットダウンできない問題を修正しました (号 #811" ）。
- ontap-san-* ドライバーでプロビジョニングされた指定された LUN に関連付けられているすべての igroup 名を一覧表示する修正を追加しました。
- 外部プロセス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（"第537号" ）。
- 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（"第781号" ）。
- 潜在的な型アサーションエラーを修正しました（"号 #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 モードのサポートが追加されました。
  - 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、アクティブなワークロードに影響を与えずに安全であるとTridentが判断した場合、新しい igroup スキームに自動的に切り替えられます ("第758号" ）。
  - 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により、Trident のセキュリティが向上しました。
- ontap-nas-economy および ontap-nas-flexgroup ストレージ ドライバーに、 Amazon FSxを使用した SMB ボリュームのサポートが追加されました。
- ontap-nas、ontap-nas-economy、ontap-nas-flexgroup ストレージ ドライバーによる 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（"第732号" ）。
- 最初に API サーバーを非アクティブ化することでTrident のシャットダウン手順を改善しました ("号 #811" ）。
- Makefile に Windows および arm64 ホストのクロスプラットフォーム ビルド サポートを追加しました。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 ("第758号" ）。

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 のクラスタ ロール権限がバンドル権限と同期するように修正されました"第799号"。
- 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。
- FlexGroupクローンのサポートと SMB ボリュームのボリューム 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 経由でTrident のインストールを修正するために、Pod 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 ("第783号、第794号" ）。

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 リソース利用率の向上 ("号 #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資格情報ローテーション サポートが追加されました。

Trident

- ontap-nas ストレージ ドライバーに、 Amazon FSx for NetApp ONTAPを使用した SMB ボリュームのサポートが追加されました。
- 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 ドライバーを使用して FlexGroup を作成するときに、flexgroupAggregateList をオプションで使用できるようにするサポートが追加されました。
- 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 ドライバーの dataLIF 更新が有効になりました。
- ホスト ノード OS を反映するように、 Tridentデプロイメントと DaemonSet の命名規則を更新しました。

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。
- 設定時にDataLIFを指定しなくなりました `ontap-san`または `ontap-san-economy`ドライバー。

22.10の変更点

- Trident 22.10 にアップグレードする前に、次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は現在、SAN環境でのマルチパス構成の使用を厳格に強制しており、推奨値は `find_multipaths: no`multipath.conf ファイル内。
>
> 非マルチパス構成の使用または `find_multipaths: yes`または `find_multipaths: smart`multipath.conf ファイルの値が小さいとマウントが失敗します。`Trident`は、`find_multipaths: no` 21.07 リリース以降。

修正

- ONTAPバックエンドで作成された特定の問題を修正 `credentials`22.07.0 アップグレード中にフィールドがオンラインにならない (["第759号"] ）。
- **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（["第548号"]そして["第760号"]）。
- 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かい再試行を削除し、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デプロイメントと DaemonSet に個別の ServiceAccount、ClusterRole、および ClusterRoleBinding を追加しました。
  - サポートを追加["クロスネームスペースボリューム共有"]。
- すべてのTrident `ontap-*`ストレージ ドライバーが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`）なし `PodSecurityPolicy`Kubernetes 1.25 をサポートするため。

- 追加した"LUKS暗号化ボリュームのサポート"のために `ontap-san`そして `ontap-san-economy`ストレージ ドライバー。
- Windows Server 2019 ノードのサポートが追加されました。
- 追加した"Windows ノード上の SMB ボリュームのサポート"を通して `azure-netapp-files`ストレージ ドライバー。
- 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。
- サポートを削除 `yes`そして `smart`オプション `find_multipaths`iSCSI のワーカー ノード マルチパスを構成する場合。

22.07の変更点

修正

Kubernetes

- Helm またはTrident Operator を使用してTrident を構成するときに、ノード セレクターのブール値と数値を処理する問題を修正しました。("GitHub の問題 #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 の問題 #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 ボリュームでは、ノードごとの igroup が使用されるようになり、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。Trident がアクティブなワークロードに影響を与えずに安全であると判断すると、既存のボリュームは新しい igroup スキームに自動的に切り替えられます。
- PriorityClass の消費がデフォルトで制限されている場合にTrident DaemonSet がスケジュールされるように、 Tridentインストールに ResourceQuota を追加しました。
- Azure NetApp Filesドライバーにネットワーク機能のサポートが追加されました。("GitHub の問題 #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 の問題 #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。

撤去

- 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"ポッドセキュリティ標準"(PSS) インストール時にTridentによって有効になる権限の詳細を説明するセクションが追加されました。

22.04の変更点

NetAppは、製品とサービスを継続的に改善および強化しています。以下はTridentの最新機能の一部です。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" 。

> 以前のTridentリリースからアップグレードし、 Azure NetApp Files を使用する場合は、`location`構成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 の問題 #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 の問題 #598")
- Trident CRD 内の重複キー宣言を修正しました。("GitHub の問題 #671")
- 不正確な CSI スナップショット ログを修正しました。 （"GitHub の問題 #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 の問題 #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 の問題 #656")
- 設定時に自動サポート画像を取得する問題を修正しました `imageRegistry`インストール中にフラグを設定します。("GitHub の問題 #715")
- Azure NetApp Filesドライバーが複数のエクスポート ルールを持つボリュームの複製に失敗する問題を修正しました。

機能強化

- Trident の安全なエンドポイントへの受信接続には、最低でも TLS 1.3 が必要になりました。("GitHub の問題 #698")
- Trident は、安全な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 アクセス許可機能を自動的に有効にしようとするようになりました。
- **Kubernetes**: Tridentデーモンセットが system-node-critical 優先度クラスで実行されるようになりました。("GitHub の問題 #694")

撤去

E シリーズ 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 の問題 #691")
- ONTAP API 応答でアグリゲート スペースの nil フィールドにアクセスするとき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:** 大規模クラスターのノード登録バックオフ再試行時間を増加します。
- 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 の問題 #489")
- SolidFireボリュームに 32 を超えるスナップショットが作成されるときに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。
- 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Azure NetApp Filesドライバーを修正しました。
- 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Azure NetApp Filesドライバーを修正しました。

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 経由でインストールされる場合、 Tridentポッドのスケジュール オプションを追加します。("GitHub の問題 #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 の問題 #633")
- Azure NetApp Filesボリュームに 'unixPermissions' オプションのサポートが追加されました。("GitHub の問題 #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 クラスターに再度追加されたときに、 Tridentコントローラーが CrashLoopBackOff 状態になる可能性がある問題があります。この問題は v21.10.1 (GitHub 問題 669) で修正されています。

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、インポートが失敗するようになりました。
- ノードが削除されてから Kubernetes クラスターに再度追加されたときにTridentコントローラーが CrashLoopBackOff 状態になる可能性がある問題を修正しました (GitHub の問題 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 の問題 514)。
- Trident がシャットダウン時に致命的なエラーを記録する問題を修正しました (GitHub 問題 597)。
- Kubernetes 関連の修正:
  - スナップショットを作成するときに、ボリュームの使用済みスペースを最小復元サイズとして返します。 `ontap-nas`そして `ontap-nas-flexgroup`ドライバー (GitHub の問題 645)。
  - 問題を修正しました `Failed to expand filesystem`ボリュームのサイズ変更後にエラーが記録されました (GitHub の問題 560)。
  - ポッドがスタックする問題を修正 `Terminating`状態 (GitHub の問題 572)。
  - ケースを修正しました `ontap-san-economy`FlexVol がスナップショット LUN でいっぱいになっている可能性があります (GitHub の問題 533)。
  - 異なるイメージでのカスタム YAML インストーラーの問題を修正しました (GitHub の問題 613)。
  - スナップショット サイズの計算を修正しました (GitHub の問題 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 の問題 599)。

機能強化

- サポートを追加 `unixPermissions`GCP-CVS パフォーマンス ボリュームのオプション。
- 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。
  - Tridentオペレーターと Helm チャートが Kubernetes 1.22 で動作するようにしました (GitHub の問題 628)。

- オペレータ画像を追加しました `tridentctl`images コマンド (GitHub の問題 570)。

実験的な機能強化

- 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-san`ドライバ。
- 技術プレビュー RESTサポートを追加しました `ontap-nas-flexgroup`、 `ontap-san` 、 そして `ontap-nas-economy`ドライバー。

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は、values.yamlを更新して設定する必要があります。 `excludePodSecurityPolicy`に `true`または追加 `--set excludePodSecurityPolicy=true`に `helm upgrade`クラスターをアップグレードする前にコマンドを実行する必要があります。
- Tridentは空白を強制するようになった `fsType` (`fsType=""`）がないボリュームの場合は `fsType`StorageClass` で指定されます。 `Kubernetes 1.17`以降を使用する場合、 `Trident`は空白の提供をサポートします。 `fsType` NFS ボリューム用。 iSCSIボリュームの場合は、 `fsType`ストレージクラスを強制する際に `fsGroup`セキュリティコンテキストを使用します。
- 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は異なる `storagePrefix`ONTAPバックエンドの値を変更するか、別の `TenantName`SolidFireバックエンド用。 Trident は、 Tridentの他のインスタンスが作成したボリュームを検出できません。 Trident はボリュームの作成をべき等操作として扱うため、 ONTAPまたはSolidFireバックエンドのいずれかで既存のボリュームを作成しようとすると成功します。もし `storagePrefix`または `TenantName`異なる場合、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演算子）を使用して `tridentctl`Tridentを管理するには、 `KUBECONFIG`環境変数が設定されています。これはKubernetesクラスタを示すために必要です `tridentctl`反対に働くはずです。複数のKubernetes環境で作業する場合は、 `KUBECONFIG`ファイルのソースは正確です。
- 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" ; 破棄マウントオプションが[StorageClass^] オンラインでのブロック破棄をサポートします。
- Kubernetes クラスターごとにTridentのインスタンスが複数ある場合、 Trident は他のインスタンスと通信できず、作成された他のボリュームを検出できません。そのため、クラスター内で複数のインスタンスが実行されると、予期しない誤った動作が発生します。 Kubernetes クラスターごとにTridentのインスタンスが 1 つだけ存在する必要があります。
- トライデントベースの場合 `StorageClass`Tridentがオフラインの間に Kubernetes からオブジェクトが削除された場合、 Trident は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を削除するには、 `tridentctl`または REST API。
- 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 はバックアップボリュームを自動的に削除しません。ボリュームを削除するには、 `tridentctl`または REST API。
- 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、 ONTAP は一度に複数の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、以下を指定する必要があります。 `managementLIF`そして `dataLIF`バ

ックエンド定義では角括弧で囲みます。例えば、`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。

> ⓘ 指定できません `dataLIF`ONTAP SAN バックエンド上。 Trident は利用可能なすべての 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。

- 使用する場合 `solidfire-san`OpenShift 4.5 のドライバーでは、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 を使用することを確認します。 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

- "TridentGitHub"
- "Tridentブログ"

以前のバージョンのドキュメント

Trident 25.06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は、"Tridentサポート ライフサイクル"。

- "Trident25.02"
- "Trident24.10"
- "Trident24.06"
- "Trident24.02"
- "Trident23.10"
- "Trident23.07"
- "Trident23.04"
- "Trident23.01"
- "Trident22.10"

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**バックアップの復元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 30 GB 以上のファイルを復元すると、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)。参照 "Trident Protectを使用してアプリケーションを保護する"手順についてはこちらをご覧ください。

始めましょう

Tridentについて学ぶ

Tridentについて学ぶ

Trident は、NetAppによって管理されている、完全にサポートされているオープン ソース プロジェクトです。これは、コンテナ ストレージ インターフェース (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とは何ですか？

Netapp Trident をNetAppすると、オンプレミスの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、Google Cloud 上のCloud Volumes Service。

Tridentは、コンテナストレージインターフェース（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、ネイティブに統合されています。"Kubernetes" 。 Trident は、クラスター内の各ワーカー ノード上で単一のコントローラー ポッドと 1 つのノード ポッドとして実行されます。参照 "Tridentアーキテクチャ" 詳細については。

Trident は、 NetAppストレージ プラットフォームの Docker エコシステムとの直接統合も提供します。 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。参照 "Docker 用のTrident をデプロイする" 詳細については。

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

NetApp製品とのKubernetes統合

NetAppのストレージ製品ポートフォリオは、Kubernetes クラスターのさまざまな側面と統合され、Kubernetes 展開の機能、性能、パフォーマンス、可用性を強化する高度なデータ管理機能を提供します。

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 システムを搭載したファイル システムを起動および実行できる、完全に管理された AWS サービスです。

Azure NetApp Files

"Azure NetApp Files"は、 NetAppを搭載したエンタープライズ グレードの 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を備え、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"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のみのストレージ アプライアンスです。

Google Cloud NetApp Volumes

"Google Cloud NetApp Volumes"は、高性能でエンタープライズ グレードのファイル ストレージを提供する、Google Cloud のフルマネージド ファイル ストレージ サービスです。

エレメントソフトウェア

"要素"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"日常的なタスクを自動化し、インフラストラクチャ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、基盤となるNetApp HCIストレージ プラットフォームに対して、コンテナ化されたアプリケーション用のストレージ デバイスを直接プロビジョニングおよび管理できます。

NetApp ONTAP

"NetApp ONTAP"は、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する、 NetApp のマルチプロトコル統合ストレージ オペレーティング システムです。

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、オンプレミスのFAS、AFA、 ASAクラスター、 ONTAP Select、 Cloud Volumes ONTAP、さまざまな導入モデルが用意されています。 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、クラスター内の各ワーカー ノード上で単一のコントローラー ポッドと 1 つのノード 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があるすべてのホストで実行されている必要があります。

コントローラーポッドとノードポッドについて

Tridentは単体で展開Tridentコントローラー ポッド1つ以上のTridentノードポッドKubernetes クラスター上で、標準の Kubernetes *CSI* サイドカー コンテナー を使用して CSI プラグインのデプロイメントを簡素化します。"Kubernetes CSI サイドカーコンテナ" Kubernetes ストレージ 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"そして"寛容と汚点"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 のインストール中に、コントローラーとノード ポッドのノード セレ

クターと許容値を構成できます。

- 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。

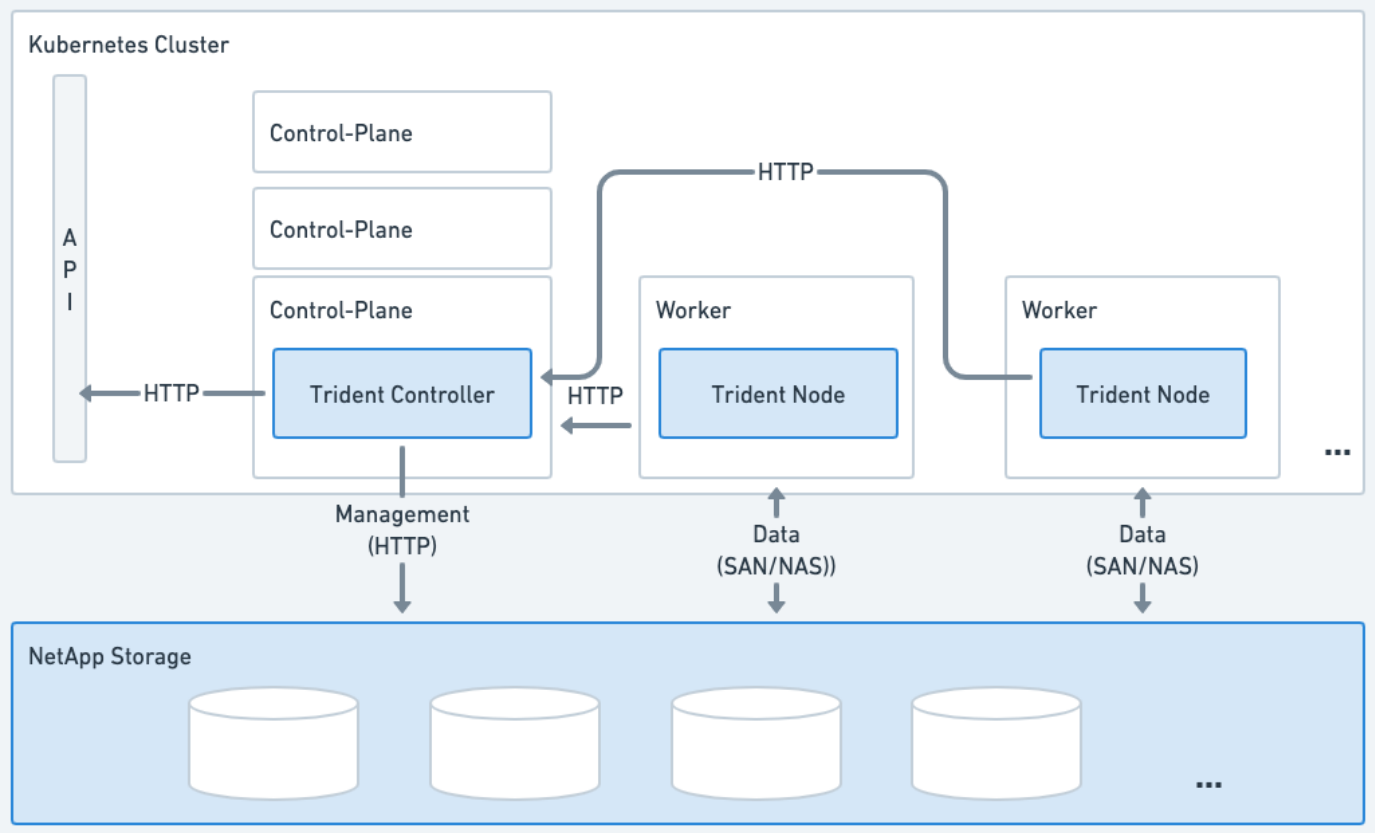


図 1. **Kubernetes** クラスターにデプロイされた**Trident**

Tridentコントローラー ポッド

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

- NetAppストレージのボリュームのプロビジョニングと管理を担当
- Kubernetesデプロイメントによって管理
- 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。

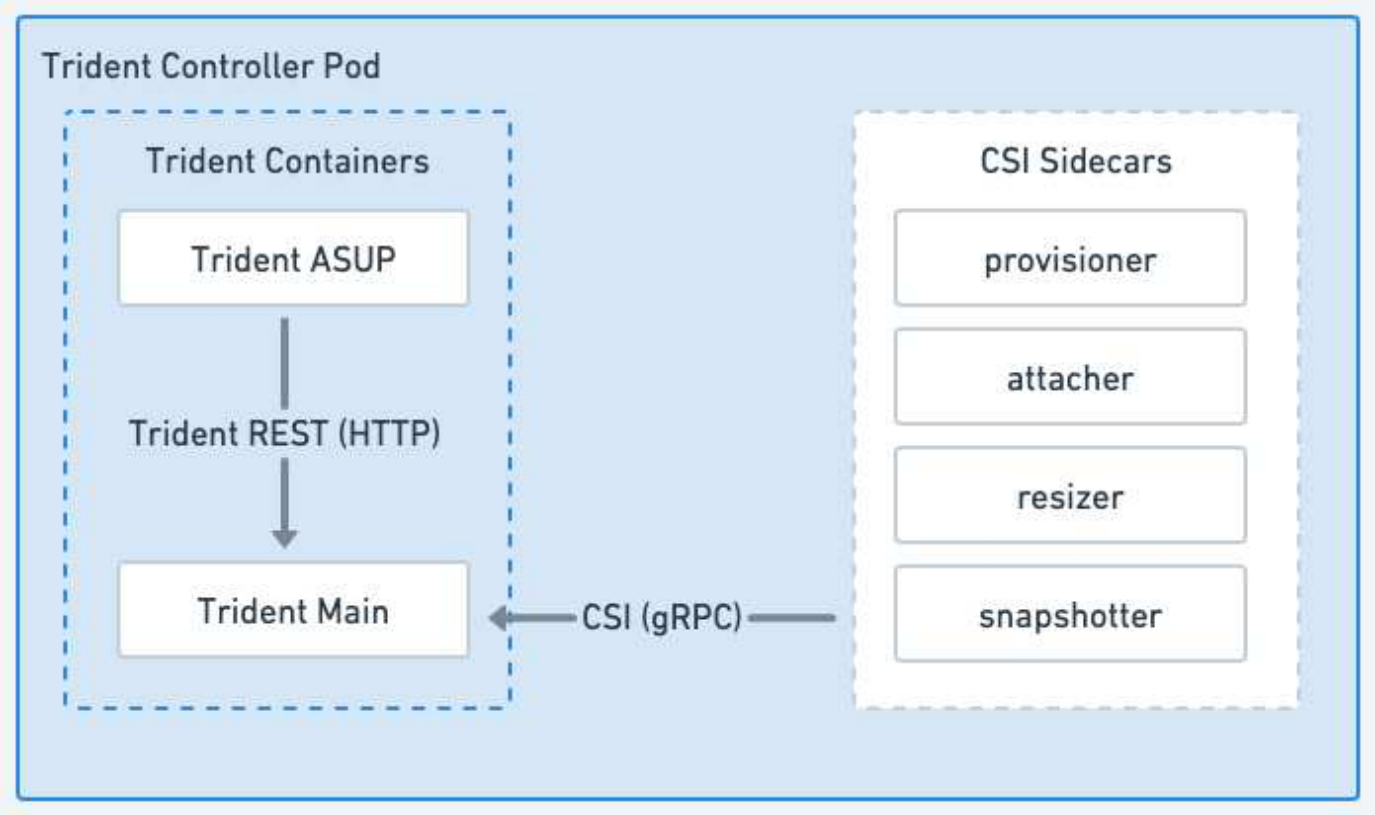


図 **2. Trident**コントローラー ポッドの図

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ポッド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によって管理される
- NetAppストレージをマウントするノードで実行する必要がある

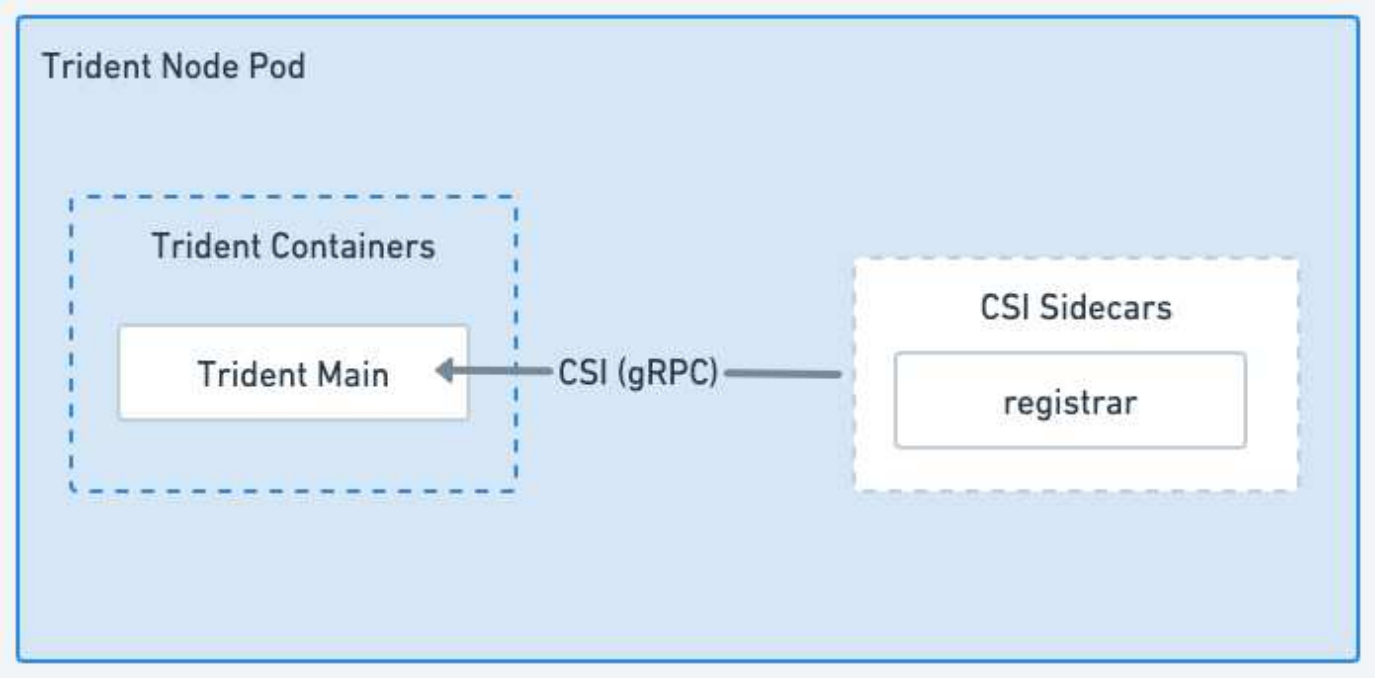


図 **3. Trident Node Pod** の図

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 クラスター 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、インフラストラクチャ、コンピューティング	はい	はい

概念

プロビジョニング

Tridentのプロビジョニングには主に 2 つのフェーズがあります。最初のフェーズでは、ストレージ クラスを適切なバックエンド ストレージ プールのセットに関連付け、プロビジョニング前の必要な準備として実行されます。 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 プールからストレージ プールを選択する必要があります。

ストレージクラスの関連付け

バックエンドストレージプールをストレージクラスに関連付けるには、ストレージクラスの要求された属性と

`storagePools`、 `additionalStoragePools` 、 そして `excludeStoragePools`リスト。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`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、そのセットにリストを追加します。使用すべきは `excludeStoragePools` ストレージ クラスで使用されていないストレージ プールを上書きして削除するためのリスト。 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

次に、 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 は最初に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、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ず、 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どのように処理するかについて詳しく説明します。

ボリュームスナップショットの作成について学ぶ

- のために `ontap-nas`、 `ontap-san` 、 `gcp-cvs` 、 そして `azure-netapp-files`ドライバーごとに、各永続ボリューム (PV) がFlexVol volumeにマップされます。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 テクノロジーは、競合するスナップショット テクノロジーよりも優れた安定性、拡張性、回復性、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。
- のために `ontap-nas-flexgroup`ドライバーでは、各永続ボリューム (PV) がFlexGroupにマップされます。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 テクノロジーは、競合するスナップショット テクノロジーよりも優れた安定性、拡張性、回復性、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。
- のために `ontap-san-economy`ドライバーでは、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 VolumeSnapshot は、関連付けられた LUN の FlexClone を実行することによって実現されます。 ONTAP FlexCloneテクノロジーにより、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ものを除いてストレージを消費しません。
- のために `solidfire-san`ドライバーにより、各 PV はNetApp Elementソフトウェア/ NetApp HCIクラスター上に作成された LUN にマッピングされます。 VolumeSnapshots は、基盤となる LUN の要素スナップショットによって表されます。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にしか消費しません。
- 作業する際は `ontap-nas`そして `ontap-san`ドライバーの場合、 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。

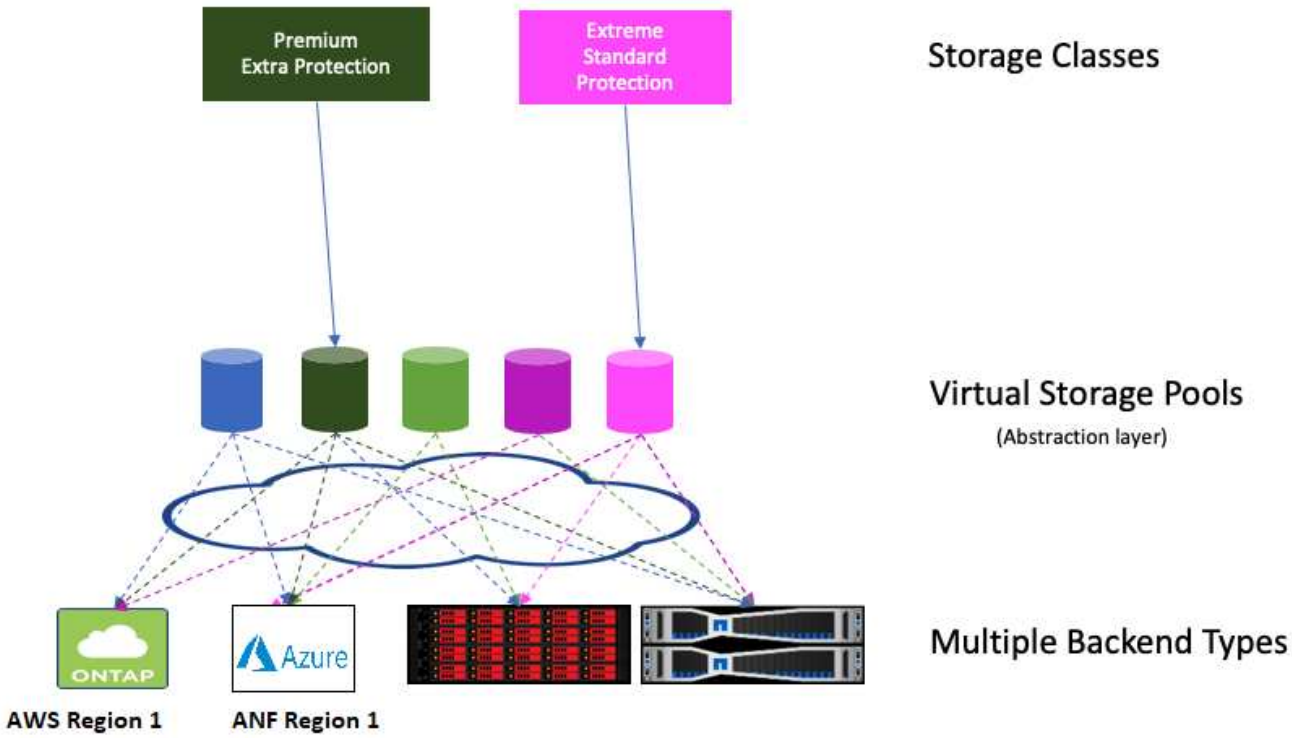
Tridentでは、VolumeSnapshots を使用してそこから新しい PV を作成できます。これらのスナップショットからの PV の作成は、サポートされているONTAPおよび CVS バックエンドのFlexCloneテクノロジーを使用して実行されます。スナップショットから PV を作成する場合、バックアップ ボリュームはスナップショットの親ボリュームのFlexCloneになります。その `solidfire-san`ドライバーは、Element ソフトウェア ボリューム クローンを使用してスナップショットから PV を作成します。ここでは、Element スナップショットからクローンを作成します。

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 の間に抽象化レイヤーを提供します。 `StorageClasses` 。管理者は、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。 `StorageClass`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 タイプを指定します。

仮想プールについて学ぶ

ストレージ管理者は、JSON または YAML 定義ファイルで、任意のTridentバックエンド上の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、マッチングには利用できないことです。 `StorageClasses`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。各ラベルはキーと値のペアであり、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 `-`

あ `StorageClass`セレクタパラメータ内のラベルを参照して、使用する仮想プールを識別します。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	マッチ
`!=`	パフォーマンス！=極端	一致しない
`in`	（東、西）の場所	価値観の集合体である
`notin`	パフォーマンスノッティング（シルバー、ブロンズ）	価値観に合わない
`<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がどのように活用されているかについて詳しくは "ボリュームアクセスグループ"。

CHAP を使用している場合は、管理を簡素化し、以下で説明するスケーリング制限を回避するためにこのセクションを無視することをお勧めします。なお、 Trident をCSI モードで使用している場合は、このセクションは無視できます。 Trident は、拡張 CSI プロビジョナーとしてインストールされる場合、CHAP を使用します。

ボリュームアクセスグループについて学ぶ

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、次のアクセスグループが見つかると予想されます。 `trident`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、これは実行されません。

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したり管理したりしません。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 が含まれている必要があります。ほとんどのインストールでは、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、以下を参照してください。 "ここをクリックしてください。" 。

デフォルトを使用している構成から変更する場合 `trident`アクセスグループを他のグループも使用するグループに変更する場合は、 `trident`リスト内のアクセス グループ。

Tridentのクイックスタート

数ステップでTridentをインストールし、ストレージ リソースの管理を開始できます。始める前に、"Tridentの要件" 。

Dockerについては、"Docker 用のTrident" 。

1 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"

2 **Trident**をインストールする

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をインストールする"

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 システム間の関係を定義します。これは、Trident にそのストレージ システムと通信する方法と、そこからボリュームをプロビジョニングする方法を指示します。

"バックエンドを構成する"ストレージシステム用

4 **Kubernetes** ストレージクラスを作成する

Kubernetes StorageClass オブジェクトは、プロビジョナーとしてTrident を指定し、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 クラスを作成できるようにします。 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。

構成された Kubernetes StorageClass を使用して PV へのアクセスを要求する PersistentVolume (PV) と PersistentVolumeClaim (PVC) を作成します。その後、PV をポッドにマウントできます。

"ボリュームをプロビジョニングする"

次の手順

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しており、推奨値は `find_multipaths: no`multipath.conf ファイル内。

  非マルチパス構成の使用または `find_multipaths: yes`または `find_multipaths: smart`multipath.conf ファイルの値が小さいとマウントが失敗します。`Trident`は、`find_multipaths: no` 21.07 リリース以降。

サポートされているフロントエンド（オーケストレーター）

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ています。

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

> Tridentは、 OpenShiftの古いバージョンを引き続きサポートします。"Red Hat 延長アップデートサポート (EUS) リリースライフサイクル"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、Tridentをインストールするときに、Kubernetes バージョンに関する警告メッセージは無視しても問題ありません。

- Rancher Kubernetes Engine 2 (RKE2) v1.27.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、以下のストレージプロバイダーとして使用できます。"キューブバート" 。

> Tridentがインストールされ ているKubernetesクラスタを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" 。

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、サポートされている以下のバックエンドの 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オール SAN アレイ (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 と 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 用の次のONTAPドライバーをサポートしています。

- オンタップナス
- オンタップナスエコノミー
- ontap-san (iSCSI、FCP、NVMe over TCP)
- ontap-san-economy (iSCSI のみ)

考慮すべき点:

- ストレージクラスを更新して、 `fsType`パラメータ（例： `fsType: "ext4"`) を OpenShift Virtualization 環境で実行します。必要に応じて、ボリュームモードを明示的にブロックに設定します。`volumeMode=Block`パラメータの `dataVolumeTemplates`CDI にブロック データ ボリュームを作成するように通知します。
- ブロック ストレージ ドライバーの *RWX* アクセス モード: ontap-san (iSCSI、NVMe/TCP、FC) および ontap-san-economy (iSCSI) ドライバーは、"volumeMode: Block" (raw デバイス) でのみサポートされます。これらのドライバーにとって、 `fstype`ボリュームが RAW デバイス モードで提供されているため、この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デバイス）

機能の要件

以下の表は、このリリースのTridentで利用できる機能と、それがサポートする Kubernetes のバージョンをまとめたものです。

特徴	**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

特徴	**Kubernetes**バージョン	機能ゲートが必要ですか**?**
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オペレーター	1.27 - 1.34	いいえ
CSIトポロジ	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

Trident は特定のオペレーティング システムを公式にはサポートしていませんが、次のオペレーティング システムは動作することが分かっています。

- AMD64 および ARM64 上の OpenShift Container Platform でサポートされる 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。

その `tridentctl`このユーティリティは、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、バックエンド構成で使用できるようになる前に、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。

"バックエンドを構成する"

Tridentポート

Trident は通信のために特定のポートにアクセスする必要があります。

"Tridentポート"

コンテナイメージと対応するKubernetesバージョン

エアギャップ インストールの場合、次のリストはTrident をインストールするために必要なコンテナー イメージのリファレンスです。使用 `tridentctl images`必要なコンテナ イメージのリストを確認するコマンド。

Trident 25.06.2 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/トライデント:25.06.2 • docker.io/netapp/trident-autosupport:25.06 • registry.k8s.io/sig-storage/csi-provisioner:v5.2.0 • registry.k8s.io/sig-storage/csi-attacher:v4.8.1 • registry.k8s.io/sig-storage/csi-resizer:v1.13.2 • registry.k8s.io/sig-storage/csi-snapshotter:v8.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.13.0 • docker.io/netapp/trident-operator:25.06.2 (オプション)

Trident 25.06 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/トライデント:25.06.0 • docker.io/netapp/trident-autosupport:25.06 • registry.k8s.io/sig-storage/csi-provisioner:v5.2.0 • registry.k8s.io/sig-storage/csi-attacher:v4.8.1 • registry.k8s.io/sig-storage/csi-resizer:v1.13.2 • registry.k8s.io/sig-storage/csi-snapshotter:v8.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.13.0 • docker.io/netapp/trident-operator:25.06.0 (オプション)

Tridentをインストールする

Tridentオペレーターを使用してインストールする

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てインストールする

Tridentを使用する

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` 、 `azure-netapp-files` 、 `gcp-cvs` 。

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"使用している場合 `ontap-san`不揮発性メモリ エクスプレス (NVMe) over TCP (NVMe/TCP) プロトコル用。

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

SCSI over FCツール

参照"FCおよびFC-NVMe SANホストの構成方法"FC および FC-NVMe SAN ホストの構成の詳細については、こちらをご覧ください。

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

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。

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>
```

発見したサービスを確認する

Trident は、 Tridentノード CR 上の各ノードに対して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。起動時に NFS サービスが起動されていることを確認します。

RHEL 8以降

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

Ubuntu

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

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5 分ごとに iSCSI 自己修復を実行して次の処理を実行します。

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. 希望する状態と現在の状態を*比較*して、必要な修復箇所を特定します。 Trident は、修理の優先順位と修理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。

自己修復活動のログは、 `trident-main`それぞれの Daemonset ポッド上のコンテナー。ログを表示するには、設定が必要です `debug`Trident のインストール中に「true」に設定します。

Trident iSCSI の自己修復機能は、次のような事態を防ぐのに役立ちます。

- 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。セッションが古い場合、 Trident はログアウトしてからポータルとの接続を再確立するまで 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。自己修復機能はこれを認識し、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 自己修復により SCSI バス内のすべてのデバイスの SCSI 再スキャンが開始されます。
- 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 自己修復により、SCSI バス内の正確な LUN ID の SCSI 再スキャンが開始されます。
- ノードごとの 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 自己修復により、SCSI バス内の正確な LUN ID の SCSI 再スキャンが開始されます。

iSCSIツールをインストールする

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使用している場合は、 `solidfire-san`ドライバーとElement OS 12.5以前を使用している場合は、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。 `/etc/iscsi/iscsid.conf` 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。

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

- iSCSI PVでRHEL/Red Hat Enterprise Linux CoreOS (RHCOS)を実行するワーカーノードを使用する場合は、 `discard`インライン スペース再利用を実行するには、StorageClass の mountOption を使用します。参照 "Red Hat ドキュメント"。
- 最新バージョンにアップグレードしたことを確認してください。 `multipath-tools` 。

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
```

> 確保する `/etc/multipath.conf`含む `find_multipaths no`下 `defaults`。

5. 確実に `iscsid`そして `multipathd`実行中:

```
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
```

> 確保する `/etc/multipath.conf`含む `find_multipaths no`下 `defaults。

5. 確実に `open-iscsi`そして `multipath-tools`有効になっていて実行中である:

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

> Ubuntu 18.04では、ターゲットポートを次のように検出する必要があります。 `iscsiadm`始める前に `open-iscsi`iSCSI デーモンを起動します。あるいは、 `iscsi`サービスを開始する `iscsid`自動的に。

iSCSI 自己修復を構成または無効にする

古いセッションを修正するには、次のTrident iSCSI 自己修復設定を構成できます。

- **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 自己修復は完全に停止します。 iSCSI 自己修復を無効にすることはお勧めしません。iSCSI 自己修復が意図したとおりに動作していない場合やデバッグ目的の場合など、特定のシナリオでのみ無効にしてください。

- **iSCSI** 自己修復待機時間: 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に iSCSI 自己修復が待機する時間を決定します (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長くなります。

舵

iSCSI自己修復設定を構成または変更するには、 `iscsiSelfHealingInterval`そして `iscsiSelfHealingWaitTime`Helm のインストールまたは Helm の更新中のパラメータ。

次の例では、iSCSI 自己修復間隔を 3 分、自己修復待機時間を 6 分に設定します。

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

トライデント**ctl**

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

次の例では、iSCSI 自己修復間隔を 3 分、自己修復待機時間を 6 分に設定します。

```
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 クラスター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 `ctrl_device_tmo`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の値。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。参照 "network interface show"詳細についてはこちらをご覧ください。
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でRHEL/Red Hat Enterprise Linux CoreOS (RHCOS)を実行するワーカーノードを使用する場合は、 `discard`インライン スペース再利用を実行するには、StorageClass の mountOption を使用します。参照 "Red Hat ドキュメント"。

RHEL 8以降

1. 次のシステム パッケージをインストールします。

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

2. マルチパスを有効にする:

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

> 確保する /etc/multipath.conf`含む `find_multipaths no`下 `defaults。

3. 確実に `multipathd`実行中:

```
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
```

> 確保する /etc/multipath.conf`含む `find_multipaths no`下 `defaults。

3. 確実に `multipath-tools`有効になっていて実行中:

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

バックエンドの構成と管理

バックエンドを構成する

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。これは、Trident にそのストレージ システムと通信する方法と、そこからボリュームをプロビジョニングする方法を指示します。

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構成する方法を学習します。

- "Azure NetApp Filesバックエンドを構成する"
- "Google Cloud NetApp Volumesバックエンドを構成する"
- "Google Cloud Platform バックエンド用のCloud Volumes Serviceを構成する"
- "NetApp HCIまたはSolidFireバックエンドを構成する"
- "ONTAPまたはCloud Volumes ONTAP NAS ドライバーを使用してバックエンドを構成する"
- "ONTAPまたはCloud Volumes ONTAP SAN ドライバーを使用してバックエンドを構成する"
- "Amazon FSx for NetApp ONTAPでTrident を使用する"

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の管理もサポートしています。

Azure NetApp Filesドライバーの詳細

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

ドライバ	プロトコル	ボリューム モード	サポートされているアクセスモード	サポートされ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。より小さいボリュームが要求された場合、 Trident は自動的に 50 GiB のボリュームを作成します。
- Trident は、 Windows ノードで実行されているポッドにマウントされた SMB ボリュームのみをサポート

します。

AKS のマネージド ID

Tridentのサポート"マネージドID"Azure Kubernetes Services クラスター用。マネージド ID によって提供される合理化された資格情報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 クラスター
- AKS Kubernetes クラスターで構成されたマネージド ID
- Tridentがインストールされており、 `cloudProvider`指定する `"Azure"。

Tridentオペレーター

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`設定する `cloudProvider`に `"Azure"。例えば：

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

舵

次の例では、 Tridentセットをインストールします `cloudProvider`環境変数を使用してAzureへ `$CP`:

```
helm install trident trident-operator-100.2506.0.tgz --create
-namespace --namespace <trident-namespace> --set cloudProvider=$CP
```

<code>トライデントctl</code>

次の例では、 Tridentをインストールし、 `cloudProvider`フラグを `Azure`:

```
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 資格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 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 クラスター
- AKS Kubernetes クラスターで構成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`指定する `"Azure"`そして `cloudIdentity`ワークロードIDの指定

Tridentオペレーター

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`設定する `cloudProvider`に `"Azure"`そして設定 `cloudIdentity`に `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

例えば：

```
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
```

舵

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、 `cloudProvider`環境変数を使用してAzureへ `$CP`そして、 `cloudIdentity`環境変数を使用する `$CI`:

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

<code>トライデントctl</code>

次の環境変数を使用して、クラウド プロバイダー および クラウド **ID** 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 `$CP`、 そして `cloud-identity`に `$CI`:

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

Azure NetApp Filesバックエンドを構成する準備をする

Azure NetApp Filesバックエンドを構成する前に、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 を初めて使用するか、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、いくつかの初期構成が必要です。参照 "Azure: Azure NetApp Files をセットアップして NFS ボリュームを作成する"。

設定して使用するには "Azure NetApp Files"バックエンドでは、次のものが必要です。

- `subscriptionID`、 `tenantID` 、 `clientID` 、 `location` 、 そして `clientSecret`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` 、 そして `clientSecret`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。参照"Microsoft: Azure NetApp Filesの容量プールを作成する"。
- Azure NetApp Filesに委任されたサブネット。参照"Microsoft: Azure NetApp Filesにサブネットを委任する"。
- `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` 、 そして `clientSecret`から"アプリ登録"Azure Active Directory で、 Azure NetApp Filesサービスに対する十分な権限を持っていること。アプリ登録で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"。
  - あ"カスタム貢献者ロール"サブスクリプションレベルで(`assignableScopes`) に、 Tridentに必要な権限のみに制限された以下の権限が付与されます。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": []
      }
    ]
  }
}
```

- アズール `location`少なくとも1つを含む "委任されたサブネット"。Trident22.01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。
- 使用するには `Cloud Identity`、取得 `client ID`から "ユーザー割り当てマネージド ID"そしてそのIDを `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

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に接続されています。参照"Microsoft: Azure NetApp Filesの Active Directory 接続の作成と管理"。
- 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 資格情報を含む少なくとも 1 つのTridentシークレット。秘密を生成する `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。設定するには `csi-proxy`、参照"GitHub: CSIプロキシ"または"GitHub: Windows 用 CSI プロキシ"Windows 上で実行されている Kubernetes ノード用。

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、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にAzure NetApp Files ボリュームを作成します。

* NetApp Trident 25.06 リリース以降、手動 QoS 容量プールがテクニカル プレビューとしてサポートされます。*

Azure NetApp Filesバックエンドは、次の構成オプションを提供します。

パラメータ	説明	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	ドライバー名 + "_" + 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。	
`tenantID`	アプリ登録からのテナント 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。	
`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。	
`serviceLevel`	1つ `Standard`、 `Premium` 、 または `Ultra`	「」（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	"[]" (フィルターなし)

パラメータ	説明	デフォルト
`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	"[]" (フィルターなし)
`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`	委任先のサブネットの名前 `Microsoft.Netapp/volumes`	""
`networkFeatures`	ボリュームのVNet機能のセット。 `Basic`または `Standard`。ネットワーク機能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。指定 `networkFeatures`この機能が有効になっていないと、ボリュームのプロビジョニングが失敗します。	""
`nfsMountOptions`	NFS マウント オプションのきめ細かな制御。 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、 `nfsvers=4`カンマ区切りのマウント オプション リストで、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。	ヌル
`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。 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	説明	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"。	
`qosType`	QoSタイプ（自動または手動）を表します。* Trident 25.06のテクニカルプレビュー*	オート
`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容量プールでのみサポートされます。* Trident 25.06のテクニカルプレビュー*	`4 MiB/sec`

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

必要な権限とリソース

PVC の作成時に「容量プールが見つかりません」というエラーが表示される場合は、アプリの登録に必要なアクセス許可とリソース (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。

の値は `resourceGroups`、 `netappAccounts` 、 `capacityPools` 、 `virtualNetwork` 、 そして `subnet`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を持つ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。

その `resourceGroups`、 `netappAccounts` 、 そして `capacityPools`値は、検出されたリソースのセットをこのストレージ バックエンドで使用可能なものに制限するフィルターであり、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	<リソース グループ>
NetAppアカウント	<リソース グループ>/<netapp アカウント>
容量プール	<リソース グループ>/<netapp アカウント>/<容量プール>
仮想ネットワーク	<リソース グループ>/<仮想ネットワーク>
サブネット	<リソース グループ>/<仮想ネットワーク>/<サブネット>

ボリュームプロビジョニング

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。参照 [構成例] 詳細については。

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。 `exportRule` CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをコ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、およびサブネットを検出し、それらのプールとサブネットのいずれかに新しいボリュームをランダムに配置します。なぜなら `nasType`省略された場合、 `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を使い始めて試してみ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のスコープをさらに指定することが必要になります。

```
---
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` 、 そして `clientSecret` 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:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
```

AKS のクラウド ID

このバックエンド構成では、 `tenantID` 、 `clientID` 、 そして `clientSecret`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ーを使用した特定のサービスレベル構成

このバックエンド構成では、Azureの `eastus`場所 `Ultra`容量プール。 Trident は、その場所で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 つに新しいボリュームをランダムに配置します。

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

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、Azureの `eastus`手動 QoS 容量プールのある場所。* NetApp Trident 25.06 のテクニカル プレビュー*。

```
---
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: my-virtual-network
subnet: my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: "true"
  size: 200Gi
  unixPermissions: "0777"
```

仮想プールの構成

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、以下の基準に基づいてプールを区別するために使用されました。 `performance` 。

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

サポートされているトポロジ構成

Trident は、リージョンと可用性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。その `supportedTopologies`このバックエンド構成のブロックは、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。ここで指定するリージョンとゾーンの値は、各 Kubernetes クラスター 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" 。

```
---
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。

定義例 `**parameter.selector**`分野

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

```
---
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

使用 `nasType`、 `node-stage-secret-name` 、 そして `node-stage-secret-namespace`、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秘密を使用する

```
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ー。 `nasType: nfs`または `nasType: null`NFS プールのフィルター。

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、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バックエンドを構成する

Google Cloud NetApp Volumes をTridentのバックエンドとして構成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 ボリュームと 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`クラスターと通信するためのドライバー。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP) です。

ドライバ	プロトコル	ボリューム モード	サポートされているアクセスモード	サポートされ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 クラスター。
- GKE クラスタに構成されたワークロード ID と、ノードプールに構成された GKE メタデータ サーバー。
- 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。

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

Tridentオペレーター

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`設定する `cloudProvider`に `"GCP"`そして設定 `cloudIdentity`に `iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com`。

例えば：

```
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'
```

舵

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、 `cloudProvider`環境変数を使用して`GCP`へ `$CP`そして、 `cloudIdentity`環境変数を使用する `$ANNOTATION`:

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

<code>トライデントctl</code>

次の環境変数を使用して、クラウド プロバイダー および クラウド **ID** 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 `$CP`、 そして `cloud-identity`に `$ANNOTATION`:

```
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バックエンドを構成する前に、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 を初めて使用するか、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の初期設定が必要です。参照"開始する前に"。

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サービスで構成された Google Cloud アカウント。参照"Google Cloud NetApp Volumes"。
- Google Cloud アカウントのプロジェクト番号。参照"プロジェクトの特定"。
- NetApp Volumes 管理者権限を持つ Google Cloud サービス アカウント(`roles/netapp.admin`） 役割。参照"アイデンティティとアクセス管理のロールと権限"。
- GCNV アカウントの API キー ファイル。参照"サービスアカウントキーを作成する"
- ストレージ プール。参照"ストレージプールの概要"。

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、以下を参照してください。"Google Cloud NetApp Volumesへのアクセスを設定する" 。

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

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	説明	デフォルト
`version`		常に1
`storageDriverName`	ストレージ ドライバーの名前	の価値 `storageDriverName`「google-cloud-netapp-volumes」として指定する必要があります。

パラメータ	説明	デフォルト
backendName	(オプション) ストレージバックエンドのカスタム名	ドライバー名 + "_" + 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ー。	
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	Trident がGCNV ボリュームを作成する Google Cloud のロケーション。クロスリージョンKubernetesクラスターを作成する場合、 `location`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	Google CloudサービスアカウントのAPIキーと netapp.admin`役割。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。その `apiKey`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 、 そして 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。	
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。	ヌル
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。 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	説明	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" 。例えば：
	`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、 `defaults`構成ファイルのセクション。

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。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`	アクセス `.snapshot`ディレクトリ	NFSv4の場合は「true」、NFSv3の場合は「false」
`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	"" (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 は構成された場所にある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、それらのプールの 1 つに新しいボリュームをランダムに配置します。なぜなら `nasType`省略された場合、 `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を使い始めて試してみ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のスコープをさらに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-----\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    XsYg6gyxy4zq7OlwWgLwGa==\n
    -----END PRIVATE KEY-----\n

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

SMBボリュームの構成

```
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。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。例えば、以下の例では `performance`ラベルと`serviceLevel`タイプは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`そして `exportRule`すべての仮想プールのデフォルトとして機能します。

詳細については、"仮想プール" 。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    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、リージョンと可用性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。その `supportedTopologies`このバックエンド構成のブロックは、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。ここで指定するリージョンとゾーンの値は、各 Kubernetes クラスター 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" 。

```
---
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、バックエンドの構成に問題があります。バックエンドを記述するには、`kubectl get tridentbackendconfig <backend-name>`次のコマンドを実行して、コマンドを実行するか、ログを表示して原因を特定します。

```
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

以下は基本的な `StorageClass`上記のバックエンドを参照する定義。

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

定義例 `**parameter.selector**`分野：

使用 `parameter.selector`それぞれ指定できます `StorageClass`その"仮想プール"ボリュームをホストするために使用されます。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、以下を参照してください。"ストレージクラスを作成する"。

SMBボリュームの定義例

使用 `nasType`、 `node-stage-secret-name` 、 そして `node-stage-secret-namespace`、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任意の 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秘密を使用する

```
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ー。 `nasType: nfs`または `nasType: null`NFS プールのフィルター。

PVC定義の例

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

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

```
kubectl get pvc gcnv-nfs-pvc

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

Google Cloud バックエンド用のCloud Volumes Serviceを構成する

提供されているサンプル構成を使用して、 Tridentインストールのバックエンドとしてを構成する方法を学習します。NetApp Cloud Volumes Service for Google Cloud

Google Cloud ドライバーの詳細

Tridentは `gcp-cvs`クラスターと通信するためのドライバー。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)です。

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

Google Cloud のCloud Volumes Serviceに対するTrident のサポートについて学ぶ

Tridentは、 2つのうちのいずれかでCloud Volumes Serviceボリュームを作成できます。"サービスの種類" :

- **CVS-Performance**: デフォルトのTridentサービス タイプ。このパフォーマンス最適化されたサービス タイプは、パフォーマンスを重視する運用ワークロードに最適です。 CVS-Performance サービス タイプは、最小 100 GiB サイズのボリュームをサポートするハードウェア オプションです。次のいずれかを選択できます"3つのサービスレベル":
  - `standard`
  - `premium`
  - `extreme`
- **CVS**: CVS サービス タイプは、限定的から中程度のパフォーマンス レベルで、高いゾーン可用性を提供します。 CVS サービス タイプは、ストレージ プールを使用して 1 GiB という小さなボリュームをサポートするソフトウェア オプションです。ストレージ プールには最大 50 個のボリュームを含めることができ、すべてのボリュームがプールの容量とパフォーマンスを共有します。次のいずれかを選択できます"2つのサービスレベル":
  - `standardsw`
  - `zoneredundantstandardsw`

要件

設定して使用するには "Cloud Volumes Service for Google Cloud"バックエンドでは、次のものが必要です。

- NetApp Cloud Volumes Serviceが設定された Google Cloud アカウント
- Google Cloud アカウントのプロジェクト番号
- Google Cloudサービスアカウント `netappcloudvolumes.admin`役割
- Cloud Volumes Serviceアカウントの API キー ファイル

バックエンド構成オプション

各バックエンドは、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	説明	デフォルト
`version`		常に1
`storageDriverName`	ストレージ ドライバーの名前	「gcp-cvs」
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + API キーの一部
`storageClass`	CVS サービス タイプを指定するために使用されるオプション パラメーター。使用 `software`CVS サービス タイプを選択します。それ以外の場合、TridentはCVS-Performanceサービスタイプを想定します。（`hardware`）。	

パラメータ	説明	デフォルト
`storagePools`	CVS サービス タイプのみ。ボリューム作成用のストレージ プールを指定するために使用されるオプションのパラメーター。	
`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。	
`hostProjectNumber`	共有 VPC ネットワークを使用する場合は必須です。このシナリオでは、 `projectNumber`奉仕プロジェクトであり、 `hostProjectNumber`ホストプロジェクトです。	
`apiRegion`	Trident がCloud Volumes Serviceボリュームを作成する Google Cloud リージョン。リージョンをまたがるKubernetesクラスタを作成する場合、 `apiRegion`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`	Google CloudサービスアカウントのAPIキーと`netappcloudvolumes.admin`役割。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。	
`proxyURL`	CVS アカウントに接続するためにプロキシ サーバーが必要な場合のプロキシ URL。プロキシ サーバーは、HTTP プロキシまたは HTTPS プロキシのいずれかになります。 HTTPS プロキシの場合、プロキシ サーバーで自己署名証明書を使用できるようにするために、証明書の検証はスキップされます。認証が有効になっているプロキシ サーバーはサポートされていません。	
`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`	新しいボリュームの CVS-Performance または CVS サービス レベル。 CVSパフォーマンス値は `standard`、 `premium` 、 または `extreme`。 CVS値は `standardsw`または `zoneredundantstandardsw`。	CVS-Performance のデフォルトは「標準」です。CVS のデフォルトは「standardsw」です。
`network`	Cloud Volumes Serviceボリュームに使用される Google Cloud ネットワーク。	"デフォルト"
`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。	ヌル

パラメータ	説明	デフォルト

`allowedTopologies`	リージョン間のアクセスを有効にするには、`allowedTopologies`すべての地域を含める必要があります。例えば：
	`- key: topology.kubernetes.io/region`
	`values:`
	`- us-east1`
	`- europe-west1`	

ボリュームのプロビジョニング オプション

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

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。 CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせを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`	アクセス`.snapshot`ディレクトリ	"間違い"
`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	"" (CVS のデフォルトの 0 を受け入れます)
`size`	新しいボリュームのサイズ。 CVS パフォーマンスの最小値は 100 GiB です。 CVS の最小値は 1 GiB です。	CVS-Performance サービス タイプのデフォルトは「100GiB」です。 CVS サービス タイプではデフォルトは設定されませんが、最小 1 GiB が必要です。

CVS-Performance サービスタイプの例

次の例は、CVS-Performance サービス タイプのサンプル構成を示しています。

例**1:** 最小限の構成

これは、デフォルトの「標準」サービス レベルでデフォルトの CVS-Performance サービス タイプを使用する最小のバックエンド構成です。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: "012345678901"
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: <id_value>
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: "123456789012345678901"
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```

例2: サービスレベル構成

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

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

例3: 仮想プールの構成

このサンプルでは `storage`仮想プールを構成し、 `StorageClasses`それらを参照します。参照[ストレージクラスの定義]ストレージ クラスがどのように定義されているかを確認します。

ここでは、すべての仮想プールに特定のデフォルトが設定され、 `snapshotReserve` 5%で `exportRule`0.0.0.0/0 に変更します。仮想プールは、 `storage`セクション。各仮想プールは独自の `serviceLevel`一部のプールではデフォルト値が上書きされます。仮想プールラベルは、以下の基準に基づいてプールを区別するために使用されました。 `performance`そして `protection`。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
    performance: extreme
    protection: extra
  serviceLevel: extreme
  defaults:
```

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

ストレージクラスの定義

次の StorageClass 定義は、仮想プールの構成例に適用されます。使用 `parameters.selector`、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 StorageClass ごとに指定できます。ボリュームには、選択したプールで定義された側面が含まれます。

ストレージクラスの例

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

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

- 最初のStorageClass(`cvs-extreme-extra-protection`) は最初の仮想プールにマップされます。これは、スナップショット予約が 10% で極めて優れたパフォーマンスを提供する唯一のプールです。
- 最後のStorageClass(`cvs-extra-protection`) は、10% のスナップショット予約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決定し、スナップショット予約要件が満たされていることを確認します。

CVS サービスタイプの例

次の例は、CVS サービス タイプのサンプル構成を示しています。

例1: 最小構成

これは、 `storageClass` CVSサービスタイプとデフォルトを指定する `standardsw`サービスレベル。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
serviceLevel: standardsw
```

例2: ストレージプールの構成

このサンプルのバックエンド構成では、 `storagePools`ストレージ プールを構成します。

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

バックエンドの作成に失敗した場合、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。

NetApp HCIまたはSolidFireバックエンドを構成する

Tridentインストールで Element バックエンドを作成して使用する方法を学習します。

要素ドライバーの詳細

Tridentは `solidfire-san`クラスターと通信するためのストレージ ドライバー。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。

その `solidfire-san`ストレージ ドライバーは、ファイル および ブロック ボリューム モードをサポートします。のために `Filesystem`volumeMode では、 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種類は、StorageClass によって指定されます。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされ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。生のブロックデバイス。
`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されているストレージ システム。
- ボリュームを管理できるNetApp HCI/ SolidFireクラスター管理者またはテナント ユーザーの資格情報。
- 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。参照"ワーカーノードの準備情報"。

バックエンド構成オプション

バックエンド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	説明	デフォルト
`version`		常に1
`storageDriverName`	ストレージ ドライバーの名前	いつも「solidfireさん」
`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

パラメータ	説明	デフォルト
`Endpoint`	テナント資格情報を持つSolidFire クラスタの 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）	
`InitiatorIFace`	iSCSIトラフィックを特定のホストインターフェースに制限する	"デフォルト"
`UseCHAP`	CHAP を使用して iSCSI を認証します。 Trident はCHAP を使用します。	true
`AccessGroups`	使用するアクセスグループIDのリスト	「trident」という名前のアクセスグループのIDを検索します
`Types`	QoS仕様	
`limitVolumeSize`	要求されたボリューム サイズがこの値を超える場合、プロビジョニングに失敗します	"" (デフォルトでは強制されません)
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例: {"api":false, "method":true}	ヌル

使用しないでください `debugTraceFlags`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。

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

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、これらのそれぞれを消費するためのストレージクラスを定義することになるでしょう。 `IOPS`ストレージ クラス パラメータ。

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

この例では、仮想プールと、それらを参照する StorageClasses が構成されたバックエンド定義ファイルを示します。

Trident は、プロビジョニング時にストレージ プールにあ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`シルバーで。仮想プールは、 `storage`セクション。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。使用して `parameters.selector`フィールドでは、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。これはゴールドパフォーマンスを提供する唯一のプールです `Volume Type QoS`ゴールドの。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 ドライバーの概要

ONTAPおよびCloud Volumes ONTAP SAN ドライバーを使用してONTAPバックエンドを構成する方法について学習します。

ONTAP SAN ドライバーの詳細

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

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされ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 はファイルシステム ボリューム モードでは使用できません。	`xfs`、 `ext3` 、 `ext4`
`ontap-san`	NVMe/TCP 参照NVMe/TCPに関する追加の考慮事項。	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし、rawブロックデバイス

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされているファイルシステム
`ontap-san`	NVMe/TCP 参照NVMe/TCPに関する追加の考慮事項。	Filesystem	RWO、RWOP ROX と RWX はファイルシステム ボリューム モードでは使用できません。	`xfs`、 `ext3` 、 `ext4`
`ontap-san-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 はファイルシステム ボリューム モードでは使用できません。	`xfs`、 `ext3` 、 `ext4`

- 使用 `ontap-san-economy`永続ボリュームの使用数が"サポートされているONTAPボリュームの制限"。
- 使用 `ontap-nas-economy`永続ボリュームの使用数が"サポートされているONTAPボリュームの制限"そして `ontap-san-economy`ドライバーは使用できません。
- 使用しないでください `ontap-nas-economy`データ保護、災害復旧、モビリティの必要性が予想される場合。
- NetApp、ontap-san を除くすべてのONTAPドライバーで Flexvol autogrow を使用することは推奨されていません。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、 ONTAPまたはSVM管理者として実行することを想定しています。 `admin`クラスターユーザーまたは `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。 Amazon FSx for NetApp ONTAPの導入では、 Tridentはクラスタを使用して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。 `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。その `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。

を使用する場合 `limitAggregateUsage`パラメータには、クラスター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`そして `fsxadmin`ユーザーアカウント。このパラメータを指定すると、構成操作は失敗します。

ONTAP内でTridentドライバーが使用できる、より制限の厳しいロールを作成することは可能ですが、お勧めしません。 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、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ています。 `ontap-san`ドライバーには以下が含まれます:

- IPv6
- NVMeボリュームのスナップショットとクローン
- 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 は以下をサポートしていません:

- NVMeでネイティブにサポートされている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) ではサポートされません。

ONTAP SAN ドライバーを使用してバックエンドを構成する準備をする

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

要件

すべてのONTAPバックエンドでは、 Trident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"ストレージ層の実装は他のONTAPシステム (ASA、 AFF、 FAS) とは異なります。 ASA r2 システムでは、集約の代わりにストレージ可用性ゾーンが使用されます。参照"事項を"ASA r2 システムで SVM にアグリゲートを割り当てる方法に関するナレッジベースの記事。

複数のドライバーを実行し、いずれかを指すストレージ クラスを作成することもできることに注意してください。例えば、 `san-dev`を使用するクラス `ontap-san`運転手と `san-default`を使用するクラス `ontap-san-economy`1つ。

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。参照"ワーカーノードを準備する" 詳細については。

ONTAPバックエンドを認証する

Trident は、 ONTAPバックエンドを認証する 2 つのモードを提供します。

- 資格情報ベース: 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。次のような事前定義され

たセキュリティログインロールを使用することをお勧めします。 `admin`または `vsadmin`ONTAPバージョンとの最大限の互換性を確保するためです。

- 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クラスター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、推奨）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。

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間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。

資格情報ベースの認証を有効にする

Trident、 ONTAPバックエンドと通信するために、SVM スコープ/クラスタ スコープの管理者の認証情報が必要です。次のような標準の事前定義されたロールを利用することをお勧めします。 `admin`または `vsadmin`。これにより、将来のTridentリリースで使用される機能 API を公開する可能性のある将来のONTAPリリースとの前方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で使用することは可能ですが、お勧めしません。

サンプルのバックエンド定義は次のようになります。

ヤムル

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

JSON

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

バックエンド定義は、資格情報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、ユーザー名とパスワードは 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、資格情報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、認証するONTAPユーザーに共通名 (CN) を設定します。

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

2. 信頼できる 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サポートしていることを確認する `cert`認証方法。

```
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、資格情報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。また、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実行します。 `tridentctl backend update` 。

```
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上の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。その後、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、古い証明書を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。バックエンドの更新が成功すると、 Trident がONTAPバックエンドと通信し、将来のボリューム操作を処理できることを示します。

Trident用のカスタムONTAPロールを作成する

最小限の権限を持つONTAPクラスタ ロールを作成すると、 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。

参照"Tridentカスタムロールジェネレーター"Tridentカスタム ロールの作成の詳細については、こちらをご覧ください。

ONTAP CLI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. クラスター レベルでカスタム ロールを作成するには、クラスター **>** 設定 を選択します。

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

   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. 役割*の下の+追加*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. 役割を**Trident**ユーザーにマップします: + ユーザーと役割 ページで次の手順を実行します。
   a. ユーザー*の下の追加アイコン+*を選択します。
   b. 必要なユーザー名を選択し、*役割*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"
- "役割とユーザーを操作する"

双方向**CHAP**による接続の認証

Tridentは、双方向CHAPを使用してiSCSIセッションを認証できます。 `ontap-san`そして `ontap-san-economy`ドライバー。これには、 `useCHAP`バックエンド定義のオプション。に設定すると `true`Trident は、SVM のデフォルトのイニシエータ セキュリティを双方向 CHAP に設定し、バックエンド ファイルから

ユーザー名とシークレットを設定します。 NetApp、接続の認証に双方向 CHAP を使用することを推奨しています。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
```

その `useCHAP`パラメータは一度だけ設定できるブールオプションです。デフォルトでは false に設定されています。 true に設定した後は、false に設定することはできません。

に加えて `useCHAP=true`、 `chapInitiatorSecret` 、 `chapTargetInitiatorSecret` 、 `chapTargetUsername` 、 そして `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、次のコマンドを実行することでシークレットを変更できます。 `tridentctl update` 。

仕組み

設定により `useCHAP`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 を構成するように指示します。これには次のものが含まれます。

- SVM で CHAP を設定する:
  - SVMのデフォルトのイニシエータセキュリティタイプがnone（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、 Tridentはデフォルトのセキュリティタイプを次のように設定します。 `CHAP` CHAP イニシエーター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 上で 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と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 は、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によって作成されるすべての PV は、CHAP 経由でマウントおよび接続されます。

認証情報をローテーションしてバックエンドを更新する

CHAP認証情報を更新するには、CHAPパラメータを更新します。 `backend.json`ファイル。これにはCHAP

シークレットを更新し、 `tridentctl update`これらの変更を反映するコマンド。

> バックエンドのCHAPシークレットを更新する場合は、 `tridentctl`バックエンドを更新します。Trident はこれらの変更を取得できないため、 ONTAP CLI またはONTAP System Manager を使用してストレージ クラスターの資格情報を更新しないでください。

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

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

既存の接続は影響を受けません。SVM 上のTridentによって資格情報が更新されると、既存の接続は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 ドライバーを作成して使用する方法を学習します。このセクションでは、バックエンドを StorageClasses にマッピングするためのバックエンド構成の例と詳細について説明します。

"ASA r2 システム"ストレージ層の実装は他のONTAPシステム (ASA、 AFF、 FAS) とは異なります。これらの

バリエーションは、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。"。

のみ `ontap-san`ドライバー (iSCSI および NVMe/TCP プロトコル付き) は、 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。選択すると `ontap-san`として `storageDriverName`Trident は、 ASA r2 または従来のONTAP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	説明	デフォルト
`version`		常に1
`storageDriverName`	ストレージ ドライバーの名前	`ontap-san`または `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバー名 + "_" + dataLIF
`managementLIF`	クラスタまたは SVM 管理 LIF の IP アドレス。 完全修飾ドメイン名 (FQDN) を指定できます。 Trident がIPv6 フラグを使用してインストールされている場合は、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 。 「vsadmin」の資格情報を使用している場合は、 `managementLIF` SVMの認証情報である必要があります。「admin」認証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	説明	デフォルト
`dataLIF`	プロトコル LIF の IP アドレス。 Trident がIPv6 フラグを使用してインストールされている場合は、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囲ん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選択的 LUN マップ"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`明示的に定義されています。 *Metrocluster の場合は省略します。*参照MetroClusterの例。	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。*参照MetroClusterの例。	SVMの場合導出される `managementLIF`指定されている
`useCHAP`	CHAP を使用してONTAP SAN ドライバーの iSCSI を認証します [ブール値]。設定 `true`Trident がバックエンドで指定された SVM のデフォルト認証として双方向 CHAP を設定して使用できるようにします。参照 "ONTAP SAN ドライバーを使用してバックエンドを構成する準備をする" 詳細については。 **FCP** または **NVMe/TCP** ではサポートされません。	`false`
`chapInitiatorSecret`	CHAP イニシエーター シークレット。必須の場合 `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 イニシエーター シークレット。必須の場合 `useCHAP=true`	""
`chapUsername`	受信ユーザー名。必須の場合 `useCHAP=true`	""
`chapTargetUsername`	ターゲットユーザー名。必須の場合 `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クラスタと通信するために必要なユーザー名。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 に対してTrident を認証する"。	""

パラメータ	説明	デフォルト
`password`	ONTAPクラスタと通信するために必要なパスワード。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 に対してTrident を認証する"。	""
`svm`	使用するストレージ仮想マシン	SVMの場合導出される `managementLIF`指定されている
`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)。のために `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、使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コントローラーを再起動せずに、SVM をポーリングすることによってTridentでも自動的に更新されます。Tridentで特定のアグリゲートをボリュームのプロビジョニング用に構成した場合、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 アグリゲートをポーリングしているときに、 Tridentでバックエンドが障害状態になります。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 システムで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、指定しないでください。 `limitAggregateUsage` 。提供された `fsxadmin`そして `vsadmin`集計使用量を取得し、 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 システムで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	説明	デフォルト
`lunsPerFlexvol`	Flexvolあたりの最大LUN数は[50, 200]の範囲でなければなりません	`100`
`debugTraceFlags`	トラブルシューティング時に使用するデバッグ フラグ。例: {"api":false, "method":true} トラブルシューティングを行っており、詳細なログ ダンプが必要な場合を除き、使用しないでください。	`null`
`useREST`	ONTAP REST API を使用するためのブール パラメーター。 `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` Tridentは、バックエンドとの通信に `ONTAPI (ZAPI)`呼び出しを使用します。この機能にはONTAP 9.11.1以降が必要です。さらに、使用するONTAPログインロールには、`ontapi`応用。これは、事前に定義された`vsadmin`そして `cluster-admin`役割。 Trident 24.06リリースおよびONTAP 9.15.1以降では、`useREST`設定されている`true`デフォルト; 変更 `useREST`に`false`ONTAPI (ZAPI)呼び出しを使用します。 `useREST`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) ではサポートされません。 指定されている場合、常に **`true`ASA r2** システムの場合。	`true`ONTAP 9.15.1以降の場合、それ以外の場合 `false`。
`sanType`	選択するには使用 `iscsi`iSCSIの場合、 `nvme` NVMe/TCPの場合または `fcp`SCSI over Fibre Channel (FC) 用。	`iscsi`空白の場合

パラメータ	説明	デフォルト
`formatOptions`	使用 `formatOptions`コマンドライン引数を指定するには `mkfs`ボリュームがフォーマットされるたびに適用されます。これにより、好みに応じてボリュームをフォーマットできます。デバイス パスを除いて、mkfs コマンド オプションと同様の formatOptions を指定してください。例: "-E nodiscard" 対応機種 **`ontap-san`**そして **`ontap-san-economy`iSCSI** プロトコルを使用したドライバー。さらに、**iSCSI** および **NVMe/TCP** プロトコルを使用する場合、 **ASA r2** システムでもサポートされます。	
`limitVolumePoolSize`	ontap-san-economy バックエンドで LUN を使用する場合の、要求可能なFlexVol の最大サイズ。	"" (デフォルトでは強制されません)
`denyNewVolumePools`	制限 `ontap-san-economy`バックエンドが LUN を格納するための新しいFlexVolボリュームを作成できないようにします。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 破棄なし:

- 保持し、mkfs 時にブロックを破棄しないでください (最初にブロックを破棄することは、ソリッド ステート デバイスおよびスパース / シン プロビジョニング ストレージで役立ちます)。これは非推奨のオプション「-K」に代わるもので、すべてのファイルシステム (xfs、ext3、ext4) 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** に対して**Trident** を認証する

Active Directory (AD) 認証情報を使用してバックエンド SVM に対して認証するようにTrident を設定できます。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。参照 "ONTAPでActive Directoryドメインコントローラのアクセスを構成する" 詳細については。

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ユーザー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` そして `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渡します。

ボリュームのプロビジョニングのためのバックエンド構成オプション

デフォルトのプロビジョニングは、以下のオプションを使用して制御できます。 `defaults`構成のセクション。例については、以下の構成例を参照してください。

パラメータ	説明	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、**`true` ASA r2** システムの場合。
`spaceReserve`	スペース予約モード。「なし」(薄い) または「ボリューム」(厚い)。 設定 **`none`ASA r2** システムの場合。	"なし"
`snapshotPolicy`	使用するスナップショット ポリシー。 設定 **`none`ASA r2** システムの場合。	"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ます。 Tridentで QoS ポリシー グループを使用するには、 ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各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ます	""
`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 システムでは指定しないでください*。	「0」の場合 `snapshotPolicy`は「なし」、それ以外の場合は「」
`splitOnClone`	クローン作成時に親からクローンを分割する	"間違い"
`encryption`	新しいボリュームでNetAppボリューム暗号化（NVE）を有効にします。デフォルトは `false`。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。バックエンドで NAE が有効になっている場合、 Tridentでプロビジョニングされたすべてのボリュームで NAE が有効になります。詳細については、以下を参照してください。"Trident がNVE および NAE と連携する仕組み" 。	"false" 指定されている場合は、**`true` ASA r2** システムの場合。

パラメータ	説明	デフォルト
`luksEncryption`	LUKS 暗号化を有効にします。参照"Linux Unified Key Setup (LUKS) を使用する"。	"" 設定 `**false**`ASA r2 システムの場合。
`tieringPolicy`	階層化ポリシーは「なし」を使用します。* ASA r2 システムで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。

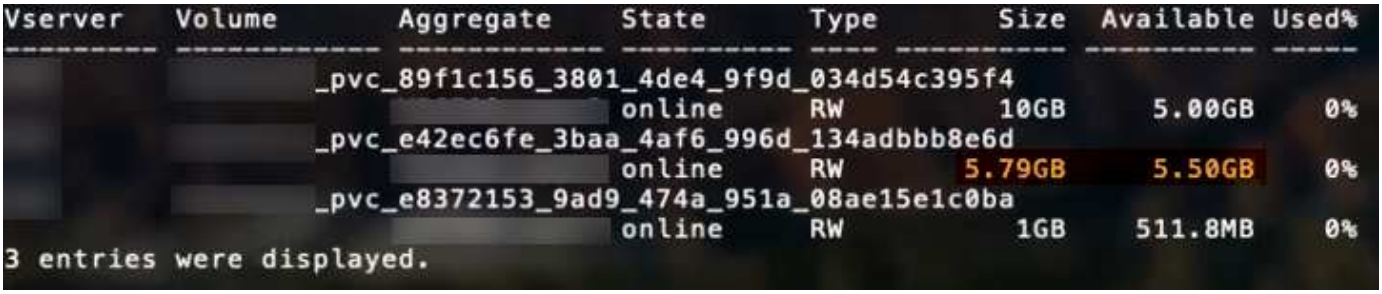
```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: <password>
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: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

作成されたすべてのボリュームについて `ontap-san`ドライバーにより、 Trident はLUN メタデータに対応するためにFlexVolに 10 パーセント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 パーセントを追加します (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す。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。

定義するバックエンドの場合 `snapshotReserve`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
```

にTridentがFlexVolに追加する10%の容量です。のために `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。その `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。

Tridentを搭載したNetApp ONTAPでAmazon FSx を使用している場合、 NetApp、 LIF に IP アドレスではなく 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。

```
---
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

バックエンドを設定することで、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くなります。"SVMのレプリケーションとリカバリ" 。

シームレスなスイッチオーバーとスイッチバックを行うには、SVMを次のように指定します。 `managementLIF`そして省略する `svm`パラメータ。例えば：

```
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` 、 そして `trustedCACertificate`（信頼できるCAを使用する場合はオプション）が入力されます `backend.json`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`に設定 `true`。

ONTAP SAN CHAPの例

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

ONTAP SANエコノミーCHAPの例

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

NVMe/TCPの例

ONTAPバックエンドに NVMe が設定された 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バックエンドに FC が設定された 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}}"
```

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

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

仮想プールを備え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。 `spaceReserve`どれも、 `spaceAllocation`偽で、そして `encryption`偽です。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、 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` 、 そして `encryption`値があり、一部のプールはデフォルト値を上書きします。

ONTAP SANの例

```yaml
---
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

以下のStorageClass定義は、[仮想プールを備えたバックエンドの例] 。使用して `parameters.selector`フィールドでは、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。

- その `protection-gold`ストレージクラスは、 `ontap-san`バックエンド。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"
```

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

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

- その `app-mysqldb`StorageClassは3番目の仮想プールにマッピングされます `ontap-san-economy`バックエンド。これは、mysqldb タイプ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"
```

- その `protection-silver-creditpoints-20k`ストレージクラスは2番目の仮想プールにマッピングされます `ontap-san`バックエンド。これは、シルバー レベルの保護と 20,000 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"
```

- その `creditpoints-5k`StorageClassは3番目の仮想プールにマッピングされます `ontap-san`バックエンドと4番目の仮想プール `ontap-san-economy`バックエンド。これらは 5000 クレジットポイント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"
```

- その `my-test-app-sc`StorageClassは `testAPP`仮想プール `ontap-san`ドライバー付き `sanType: nvme`。これは唯一のプールの提供です `testApp`。

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

Trident はどの仮想プールが選択されるか決定し、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 ドライバーの概要

ONTAPおよびCloud Volumes ONTAP NAS ドライバーを使用してONTAPバックエンドを構成する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 ドライバーを提供します。サポートされているアクセス モードは、*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP) です。

ドライバ	プロトコル	ボリューム モード	サポートされているアクセスモード	サポートされ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`

- 使用 `ontap-san-economy`永続ボリュームの使用数が"サポートされているONTAPボリュームの制限"。
- 使用 `ontap-nas-economy`永続ボリュームの使用数が"サポートされているONTAPボリュームの制限"そして `ontap-san-economy`ドライバーは使用できません。
- 使用しないでください `ontap-nas-economy`データ保護、災害復旧、モビリティの必要性が予想される場合。
- NetApp、ontap-san を除くすべてのONTAPドライバーで Flexvol autogrow を使用することは推奨されていません。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、 ONTAPまたはSVM管理者として実行することを想定しています。 `admin`クラスターユーザーまたは `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。

Amazon FSx for NetApp ONTAPの導入では、 Tridentはクラスタを使用して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。 `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。その `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。

を使用する場合 `limitAggregateUsage`パラメータには、クラスター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`そして `fsxadmin`ユーザーアカウント。このパラメータを指定すると、構成操作は失敗します。

ONTAP内でTridentドライバーが使用できる、より制限の厳しいロールを作成することは可能ですが、お勧めしません。 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てバックエンドを構成する準備をする

ONTAP NAS ドライバーを使用してONTAPバックエンドを構成するための要件、認証オ

プション、およびエクスポート ポリシーを理解します。

要件

- すべてのONTAPバックエンドでは、 Trident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` ドライバーとブロンズクラスは `ontap-nas-economy`1つ。
- すべての Kubernetes ワーカーノードに適切な NFS ツールがインストールされている必要があります。参照"ここをクリックしてください。"詳細についてはこちらをご覧ください。
- Trident は、 Windows ノードで実行されているポッドにマウントされた SMB ボリュームのみをサポートします。参照 SMBボリュームのプロビジョニングの準備 詳細については。

ONTAPバックエンドを認証する

Trident は、 ONTAPバックエンドを認証する 2 つのモードを提供します。

- 認証情報ベース: 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。次のような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。 `admin`または `vsadmin`ONTAPバージョンとの最大限の互換性を確保するためです。
- 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がインストールされている必要があります。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間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。

資格情報ベースの認証を有効にする

Trident、 ONTAPバックエンドと通信するために、SVM スコープ/クラスタ スコープの管理者の認証情報が必要です。次のような標準の事前定義されたロールを利用することをお勧めします。 `admin`または `vsadmin`。これにより、将来のTridentリリースで使用される機能 API を公開する可能性のある将来のONTAPリリースとの前方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で使用することは可能ですが、お勧めしません。

サンプルのバックエンド定義は次のようになります。

ヤムル

```
---
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、資格情報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、ユーザー名とパスワードは 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、資格情報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、認証するONTAPユーザーに共通名 (CN) を設定します。

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

2. 信頼できる 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サポートしていることを確認する `cert`認証方法。

```
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次のように設定されていることを確認する必要があります。 `default-data-management` 。

```
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、資格情報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。また、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実行します。 `tridentctl update backend` 。

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

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

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

パスワードをローテーションする場合、ストレージ管理者はまずONTAP上の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。その後、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、古い証明書を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。バックエンドの更新が成功すると、 Trident がONTAPバックエンドと通信し、将来のボリューム操作を処理できることを示します。

Trident用のカスタムONTAPロールを作成する

最小限の権限を持つONTAPクラスタ ロールを作成すると、 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。

参照"Tridentカスタムロールジェネレーター"Tridentカスタム ロールの作成の詳細については、こちらをご覧ください。

ONTAP CLI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. クラスター レベルでカスタム ロールを作成するには、クラスター **>** 設定 を選択します。

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

   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. 役割*の下の+追加*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. 役割を**Trident**ユーザーにマップします: + ユーザーと役割 ページで次の手順を実行します。
   a. ユーザー*の下の追加アイコン+*を選択します。
   b. 必要なユーザー名を選択し、*役割*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"
- "役割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、エクスポート ポリシーを操作するときに 2 つのオプションが提供されます。

- Trident はエクスポート ポリシー自体を動的に管理できます。この動作モードでは、ストレージ管理者は許容 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 ポリシー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 ポリシーに追加されます。
- ストレージ管理者は、エクスポート ポリシーを作成し、ルールを手動で追加できます。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、 Trident は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 クラスター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 クラスター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は、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 ルールを含むエクスポート ポリシーが以前に作成されていることを確認する必要があります (デフォルトのエクスポート ポリシーなど)。 SVM をTrident専用にする場合は、常にNetApp が推奨するベスト プラクティスに従ってください。

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`設定されている `true`。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ボリュームごとにエクスポートポリシーを作成することを示します。 `svm1` SVMを使用してルールの追加と削除を処理します `autoexportCIDRs`アドレス ブロック。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の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で使用されるエクスポートポリシーにも追加されます。
  - 例えば：
    - 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という名前のエクスポートポリシーを作成します。 `trident-403b5326-8482-40db96d0-d83fb3f4daec` 、qtreeのエクスポートポリシー `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`、および空のエクスポートポリシー `trident_empty`SVM 上。 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`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 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、公開されているこのアドレス範囲内にある Kubernetes ノード IP が、 Trident が作成するエクスポート ポリシーに追加されることを示します。Tridentは、そのノードを登録する際に、そのノードのIPアドレスを取得し、それを以下のアドレスブロックと照合します。 `autoExportCIDRs`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。

更新できます `autoExportPolicy`そして `autoExportCIDRs`バックエンドを作成した後。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。無効にすることもできます `autoExportPolicy`バックエンドにエクスポート ポリシーを手動で作成し、フォールバックします。これには、 `exportPolicy`バックエンド構成のパラメータ。

Tridentがバックエンドを作成または更新した後、以下のコマンドでバックエンドを確認できます。 `tridentctl`または対応する `tridentbackend`CRD:

```
./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、その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 を削除することにより、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、 Trident は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、バックエンドを次のように更新します。 `tridentctl update backend` Trident がエクスポート ポリシーを自動的に管理することを保証します。これにより、必要に応じて、バックエンドの UUID と qtree 名に基づいて名前が付けられた 2 つの新しいエクスポート ポリシーが作成されます。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。その後、 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、SMBボリュームをプロビジョニングできます。 `ontap-nas`ドライバー。

> 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。 `ontap-nas-economy` ONTAPオンプレミス クラスターの SMB ボリューム。これらのプロトコルのいずれかを構成しないと、SMB ボリュームの作成が失敗します。

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

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 資格情報を含む少なくとも 1 つのTridentシークレット。秘密を生成する `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。設定するには `csi-proxy`、参照"GitHub: CSIプロキシ"または"GitHub: Windows 用 CSI プロキシ"Windows 上で実行されている Kubernetes ノード用。

手順

1. オンプレミスのONTAPの場合、オプションで SMB 共有を作成するか、 Trident で作成することもできます。

   Amazon FSx for ONTAPには SMB 共有が必要です。

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

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

      その `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
```

   参照"SMB共有を作成する"詳細についてはこちらをご覧ください。

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。 FSx for ONTAPのバックエンド構成オプションについては、以下を参照してください。"FSx for ONTAP の

構成オプションと例" 。

パラメータ	説明	例
smbShare	次のいずれかを指定できます: Microsoft 管理コンソールまたはONTAP CLI を使用して作成された SMB 共有の名前、 Trident がSMB 共有を作成できるようにする名前、またはボリュームへの共通共有アクセスを防止するためにパラメータを空白のままにしておくことができます。このパラメータは、オンプレミスのONTAPではオプションです。このパラメータはAmazon FSx for ONTAPバックエンドに必須であり、空白にすることはできません。	smb-share
nasType	設定する必要があります **smb.** nullの場合、デフォルトは nfs。	smb
securityStyle	新しいボリュームのセキュリティ スタイル。 設定する必要があります **`ntfs`**または **`mixed`SMB** ボリュームの場合。	`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。	""

安全な**SMB**を有効にする

25.06リリース以降、 NetApp Tridentは、以下の方法で作成された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。 `ontap-nas`そして `ontap-nas-economy`バックエンド。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

- インポート `ontap-nas-economy`ボリュームはサポートされていません。
- 読み取り専用クローンのみがサポートされています `ontap-nas-economy`ボリューム。
- Secure SMB が有効になっている場合、 Trident はバックエンドに記載されている 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 の注釈で指定された SMB 共有 ACL は、ソース PVC の ACL よりも優先されます。
- 安全な SMB を有効にする際には、有効な AD ユーザーを提供するようにしてください。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は以下でサポートされています `ontap-nas`管理対象ボリュームのインポートには適用され、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. 次の例に示すように、TridentBackendConfig で adAdminUser を指定します。

```
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。

   追加する `trident.netapp.io/smbShareAdUser` ストレージ クラスにアノテーションを追加して、安全な SMB を確実に有効にします。注釈に指定されたユーザー値 `trident.netapp.io/smbShareAdUser` 指定されたユーザー名と同じである必要があります `smbcreds` 秘密。次のいずれかを選択できます `smbShareAdUserPermission`: `full_control` 、 `change` 、 または `read`。デフォルトの権限は `full_control`。

```
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 の構成オプションと例

TridentインストールでONTAP NAS ドライバーを作成して使用する方法を学習します。このセクションでは、バックエンドを StorageClasses にマッピングするためのバックエンド構成の例と詳細について説明します。

バックエンド構成オプション

バックエンド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	説明	デフォルト
`version`		常に1
`storageDriverName`	ストレージ ドライバーの名前	`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 。	"10.0.0.1"、"[2001:1234:abcd::fefe]"

パラメータ	説明	デフォルト
`dataLIF`	プロトコル LIF の IP アドレス。 NetAppは以下を指定することを推奨しています `dataLIF`。指定されない場合、 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) を指定できるため、複数のデータ LIF 間で負荷分散を行うラウンドロビン DNS を作成できます。初期設定後も変更可能です。参照。 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]`。 *Metrocluster の場合は省略します。*参照MetroClusterの例。	指定されたアドレス、または指定されていない場合は SVM から派生したアドレス（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。*参照MetroClusterの例。	SVMの場合導出される `managementLIF`指定されている
`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]。使用して `autoExportPolicy`そして `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`	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`が有効になります。使用して `autoExportPolicy`そして `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	""
`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 に対してTrident を認証する"。	
`password`	クラスター/SVM に接続するためのパスワード。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 に対してTrident を認証する"。	

パラメータ	説明	デフォルト
`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 プレフィックスが埋め込まれますが、qtree には埋め込まれません。	"トライデント"
`aggregate`	プロビジョニング用のアグリゲート (オプション。設定する場合は、SVM に割り当てる必要があります)。のために `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、使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コントローラーを再起動せずに、SVM をポーリングすることによってTridentでも自動的に更新されます。Tridentで特定のアグリゲートをボリュームのプロビジョニング用に構成した場合、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 アグリゲートをポーリングしているときに、 Tridentでバックエンドが障害状態になります。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。	""
`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*。	"" (デフォルトでは強制されません)

パラメータ	説明	デフォルト
flexgroupAggregateList	プロビジョニング用のアグリゲートのリスト (オプション。設定する場合は、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 で集計リストが更新されると、Tridentコントローラーを再起動せずに、SVM をポーリングすることによってTridentのリストが自動的に更新されます。ボリュームをプロビジョニングするためにTridentで特定の集約リストを設定した場合、集約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 集約をポーリングしているときに、 Tridentでバックエンドが障害状態になります。バックエンドをオンラインに戻すには、集約リストをSVM 上に存在するリストに変更するか、集約リストを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。また、qtreeの管理対象とな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)
`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} 使用しないでください `debugTraceFlags`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。	ヌル
`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。 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`
`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 オプションを使用します。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 は関連付けられた永続ボリュームにマウント オプションを設定しません。	""
`qtreesPerFlexvol`	FlexVolあたりの最大Qtree数は、[50, 300]の範囲でなければなりません	「200」

パラメータ	説明	デフォルト
smbShare	次のいずれかを指定できます: Microsoft 管理コンソールまたはONTAP CLI を使用して作成された SMB 共有の名前、 Trident がSMB 共有を作成できるようにする名前、またはボリュームへの共通共有アクセスを防止するためにパラメータを空白のままにしておくことができます。このパラメータは、オンプレミスのONTAPではオプションです。このパラメータはAmazon FSx for ONTAPバックエンドに必須であり、空白にすることはできません。	smb-share
useREST	ONTAP REST API を使用するためのブール パラメーター。 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 Trident は、バックエンドとの通信にONTAPI (ZAPI) 呼び出しを使用します。この機能にはONTAP 9.11.1 以降が必要です。さらに、使用するONTAPログインロールには、 `ontapi`応用。これは、事前に定義された `vsadmin`そして `cluster-admin`役割。 Trident 24.06リリースおよびONTAP 9.15.1以降では、 `useREST`設定されている `true`デフォルト; 変更 `useREST`に `false`ONTAPI (ZAPI) 呼び出しを使用します。	true`ONTAP 9.15.1以降の場合、それ以外の場合 `false。
limitVolumePoolSize	ontap-nas-economy バックエンドで Qtree を使用する場合の、要求可能なFlexVol の最大サイズ。	"" (デフォルトでは強制されません)
denyNewVolumePools	制限 `ontap-nas-economy`バックエンドが Qtree を格納するための新しいFlexVolボリュームを作成できないようにします。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 共有への完全な制御権限を持つ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のためのバックエンド構成オプション

デフォルトのプロビジョニングは、以下のオプションを使用して制御できます。 `defaults`構成のセクション。例については、以下の構成例を参照してください。

パラメータ	説明	デフォルト
spaceAllocation	Qtreeのスペース割り当て	"真実"
spaceReserve	スペース予約モード。「なし」（薄い）または「ボリューム」（厚い）	"なし"
snapshotPolicy	使用するスナップショットポリシー	"なし"

パラメータ	説明	デフォルト
qosPolicy	作成されたボリュームに割り当てる QoS ポリシー グループ。ストレージプール/バックエンドごとに qosPolicy または adaptiveQosPolicy のいずれかを選択します	""
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ます。 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」の場合 `snapshotPolicy`は「なし」、それ以外の場合は「」
splitOnClone	クローン作成時に親からクローンを分割する	"間違い"
encryption	新しいボリュームでNetAppボリューム暗号化（NVE）を有効にします。デフォルトは false。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。バックエンドで NAE が有効になっている場合、 Tridentでプロビジョニングされたすべてのボリュームで NAE が有効になります。詳細については、以下を参照してください。"Trident がNVE および NAE と連携する仕組み" 。	"間違い"
tieringPolicy	「なし」を使用する階層化ポリシー	
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	アクセスを制御します `.snapshot`ディレクトリ	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	"デフォルト"
securityStyle	新しいボリュームのセキュリティ スタイル。 NFSサポート `mixed`そして `unix`セキュリティ スタイル。SMBサポート `mixed`そして `ntfs`セキュリティ スタイル。	NFSのデフォルトは unix。 SMBのデフォルトは ntfs。
nameTemplate	カスタムボリューム名を作成するためのテンプレート。	""

Tridentで QoS ポリシー グループを使用するには、 ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各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"
```

のために ontap-nas`そして `ontap-nas-flexgroups`Trident、新しい計算を使用して、SnapshotReserve のパーセンテージと PVC に合わせてFlexVol のサイズが適切に設定されるようになりました。ユーザーがPVCを要求すると、 Tridentは新しい計算方法を用いて、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り、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より前のバージョンでは、ユーザーがSnapshotReserveを50%に設定してPVC（例えば5GiB）を要求した場合、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全巻であり、`snapshotReserve`それはそのパーセンテージです。 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はそれを定義します。 `snapshotReserve`全体の量の割合として数値を表示します。これは適用されません `ontap-nas-economy。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

計算は次のようになります。

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

snapshotReserve = 50%、PVCリクエスト = 5 GiBの場合、ボリュームの合計サイズは5/0.5 = 10 GiBとなり、使用可能なサイズはユーザーがPVCリクエストで要求した5 GiBになりますその `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をアップグレードする際に、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については、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。例えば、2GiBのPVCで `snapshotReserve=50`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。例えば、ボリュームを3GiBにサイズ変更すると、6GiBのボリュームで3GiBの書き込み可能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。

> TridentをNetApp ONTAPでAmazon FSx を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お勧めします。

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

バックエンドを設定することで、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くなります。"SVMのレプリケーションとリカバリ"。

シームレスなスイッチオーバーとスイッチバックを行うには、SVMを次のように指定します。`managementLIF`そして省略する `dataLIF`そして `svm`パラメータ。例えば：

```
---
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` 、 そして `trustedCACertificate`（信頼できるCAを使用する場合はオプション）が入力されます `backend.json`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、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Tridentに指示する方法を示します。これは、 `ontap-nas-economy`そして `ontap-nas-flexgroup` ドライバー。

```
---
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`IPv6 アドレスを使用します。

```
---
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
```

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

その `smbShare`SMB ボリュームを使用する FSx for ONTAPにはこのパラメータが必要です。

```
---
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}}"
```

仮想プールを備え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。 `spaceReserve`どれも、 `spaceAllocation`偽で、そして `encryption`偽です。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。 FlexVolにコメントが設定されている `ontap-nas`または`FlexGroup `ontap-nas-flexgroup`。 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` 、 そして `encryption`値があり、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

以下のStorageClass定義は、[仮想プールを備えたバックエンドの例] 。使用して `parameters.selector`フィールドでは、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。

- その `protection-gold`ストレージクラスは、 `ontap-nas-flexgroup`バックエンド。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"
```

- その `protection-not-gold`ストレージクラスは、 `ontap-nas-flexgroup`バックエンド。これらは、ゴールド以外の保護レベルを提供する唯一のプールです。

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

- その `app-mysqldb`ストレージクラスは、 `ontap-nas`バックエンド。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`ストレージクラスは、 `ontap-nas-flexgroup`バックエンド。これは、シルバー レベルの保護と 20,000 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"
```

- その `creditpoints-5k`ストレージクラスは、 `ontap-nas`バックエンドと2番目の仮想プール `ontap-nas-economy`バックエンド。これらは 5000 クレジットポイント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。

アップデート `dataLIF`初期設定後

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。

セキュアな中小企業の例

ontap-nas ドライバー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 ドライバー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 ドライバー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
```

必ず追加してください `annotations`安全な SMB を有効にします。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 SMB は機能しません。

ontap-nas-economy ドライバー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** ユーザーによる **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** ユーザーによる **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 を使用する

"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 システムを起動および実行できるようにする、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、スケーラビリティを活用できます。 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と統合して、Amazon Elastic Kubernetes Service (EKS) で実行されている Kubernetes クラスターがONTAPによって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することができます。

ファイルシステムは、オンプレミスのONTAPクラスターに類似した、 Amazon FSxの主要なリソースです。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 つまたは複数のボリュームを作成できます。 Amazon FSx for NetApp ONTAP 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 をAmazon FSx for NetApp ONTAPと併用することで、Amazon Elastic Kubernetes Service (EKS) で実行されている Kubernetes クラスターがONTAPによって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。

要件

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

- 既存のAmazon EKSクラスターまたはセルフマネージドKubernetesクラスターで `kubectl`インストールされました。
- クラスターのワーカーノードからアクセスできる既存のAmazon FSx for NetApp ONTAPファイルシステムとストレージ仮想マシン (SVM)。
- 準備されたワーカーノード"NFS または iSCSI"。

> Amazon LinuxおよびUbuntuに必要なノード準備手順に従ってください。 "Amazon マシンイメージ" (AMI) は EKS AMI タイプによって異なります。

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。
  - 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。参照 "SMBボリュームのプロビジョニングの準備" 詳細については。
- 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では削除できませんでした。 Trident 24.02以降でこの問題を回避するには、 `fsxFilesystemID` 、AWS `apiRegion` 、AWS `apikey` 、AWS `secretKey` AWS FSx for ONTAPのバックエンド構成ファイル内。

> TridentにIAMロールを指定する場合は、 `apiRegion` 、 `apiKey` 、 そして `secretKey`フィールドをTridentに明示的に渡します。詳細については、"FSx for ONTAP の構成オプションと例" 。

Trident SAN/iSCSI と EBS-CSI ドライバーの同時使用

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、ノードに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 ディスクに干渉せずにマルチパスが機能することを保証するには、マルチパス設定で EBS を除外する必要があります。この例では、 `multipath.conf` EBS ディスクをマルチパスから除外しながら必要なTrident設定を含むファイル:

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

認証

Trident は2 つの認証モードを提供します。

- 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。使用することができます `fsxadmin`ファイルシステムのユーザーまたは `vsadmin`SVM 用に構成されたユーザー。

> 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。 Amazon FSx for NetApp ONTAPには `fsxadmin`ONTAPの限定的な代替品であるユーザー `admin`クラスター ユーザー。強くお勧めします `vsadmin`Trident付き。

- 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。次の AMI はNetApp Trident 25.02 でテストされています。

アミ	NAS	NAS-エコノミー	iSCSI	iSCSIエコノミー
AL2023_x86_64_標準	はい	はい	はい	はい
AL2_x86_64	はい	はい	○*	○*
ボトルロケット_x86_64	はい**	はい	該当なし	該当なし
AL2023_ARM_64_標準	はい	はい	はい	はい
AL2_ARM_64	はい	はい	○*	○*

ボトルロケットアーム64	はい**	はい	該当なし	該当なし

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

> 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: ストレージクラス、PVC、ポッド
- 削除: ポッド、PVC (通常、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 クラスターをデプロイ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 認証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 に必要な権限を与え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"
}
```

サービス アカウントの関連付け (IRSA) 用の Pod Identity または 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。

ポッドのアイデンティティ

Amazon EKS ポッドアイデンティティの関連付けは、Amazon EC2 インスタンスプロファイルが Amazon EC2 インスタンスに認証情報を提供するのと同様に、アプリケーションの認証情報を管理する機能を提供します。

EKS クラスターに **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 ID 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 ポッドアイデンティティエージェントを設定する" 。

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、次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 の関連付けを作成します。

```
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"
        }
      }
    }
  ]
}
```

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

- **<account_id>** - AWS アカウントID
- **<oidc_provider>** - EKS クラスターの 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、(上記の手順で作成した) 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 をクラスターに関連付けます。

```
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をインストールする

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

次のいずれかの方法でTrident をインストールできます。

- 舵
- 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。参照"CSIボリュームのスナップショット機能を有効にする"詳細についてはこちらをご覧ください。

Helm経由で**Trident**をインストールする

ポッドのアイデンティティ

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
```

使用することができます `helm list`名前、名前空間、チャート、ステータス、アプリのバージョン、リビジョン番号などのインストールの詳細を確認するコマンド。

```
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. クラウド プロバイダー と クラウド **ID** 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
```

使用することができます `helm list`名前、名前空間、チャート、ステータス、アプリのバージョン、リビジョン番号などのインストールの詳細を確認するコマンド。

```
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-
100.2506.0    25.06.0
```

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 ワーカー ノード 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**をインストールする

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、以下があることを確認してください。

- アドオンサブスクリプション付きの Amazon EKS クラスターアカウント
- 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、[クラスター] を選択します。
3. NetApp Trident CSI アドオンを構成するクラスター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 セクションまで下にスクロールし、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 ボックスを選択します。
   c. *次へ*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. 使用したい*バージョン*を選択してください。
   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。
      - 使用したい*バージョン*を選択してください。
      - アドオン構成スキーマ に従い、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 (値は次の形式である必要があります)。

```
{

  "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. *次へ*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.作成する `**add-on.json**`ファイル：

Pod Identity 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",
}
```

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"
  }
}
```

交換する `<role ARN>`前の手順で作成されたロールの ARN を使用します。

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コンソールを開く https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーション ペインで、[クラスター] 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. * Trident by NetAppの構成 *ページで、次の操作を行います。
   a. 使用したい*バージョン*を選択してください。
   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 アドオンを更新します。

```
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ます。交換する `my-cluster`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を更新します。

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

削除すると `--force`オプションと 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が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、"アドオン" 。 Amazon EKS Kubernetesフィールド管理の詳細については、以下を参照してください。"Kubernetes フィールド管理" 。

Trident EKSアドオンをアンインストール/削除する

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、アドオン ソフトウェアはクラスター上に保持されます。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。保持する `--preserve`アドオンを保持するには、コマンドにオプションを追加します。
- クラスターからアドオン ソフトウェアを完全に削除する – NetApp、クラスター上に Amazon EKS アドオンに依存するリソースがない場合にのみ、クラスターから Amazon EKS アドオンを削除することをお勧めします。削除する `--preserve`オプションから `delete`アドオンを削除するコマンド。

アドオンに IAM アカウントが関連付けられている場合、その 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、[クラスター] 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、 Trident by NetApp*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、[クラスターで保持] を選択します。アドオン ソフトウェアをクラスター上に保持して、アドオンのすべての設定を自分で管理できるようにする場合は、これを実行してください。
   b. **netapp_trident-operator** と入力します。
   c. *削除*を選択します。

AWS CLI

交換する `my-cluster`クラスターの名前に変更し、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構成する

ONTAP SANおよびNASドライバーの統合

ストレージ バックエンドを作成するには、JSON または YAML 形式で構成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)、ストレージの取得元となるファイル システム、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 への認証情報を保存する方法を示しています。

ヤムル

```
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) を作成し、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

次のドライバーを使用して、 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、独自のAmazon FSx for NetApp ONTAPボリューム内の LUN です。ブロックストレージに推奨されます。
- `ontap-nas`: プロビジョニングされる各 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る各 PV は、 Amazon FSx for NetApp ONTAPボリュームごとに設定可能な数の LUN を持つ LUN です。
- `ontap-nas-economy`: プロビジョニングされる各 PV は qtree であり、 Amazon FSx for NetApp ONTAP ボリュームごとに設定可能な数の qtree があります。
- `ontap-nas-flexgroup`: プロビジョニングされる各 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"そして"SANドライバー"。

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	説明	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) を指定できます。 TridentがIPv6 フラグを使用してインストールされている場合は、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。あなたが提供した `fsxFilesystemID`の下で `aws`フィールドに入力する必要はありません `managementLIF`TridentはSVMを取得するため `managementLIF`AWS からの情報。そのため、SVM下のユーザーの資格情報（例：vsadmin）を提供する必要があり、ユーザーは `vsadmin`役割。	"10.0.0.1"、"[2001:1234:abcd::fefe]"

パラメータ	説明	例
`dataLIF`	プロトコル LIF の IP アドレス。 * ONTAP NAS ドライバー*: NetApp、 dataLIF を指定することを推奨しています。指定されない場合、 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) を指定できるため、複数のデータ LIF 間で負荷分散を行うラウンドロビン DNS を作成できます。初期設定後も変更可能です。参照。 * ONTAP SAN ドライバー*: iSCSI の場合は指定しないでください。 Trident はONTAP選択的 LUN マップを使用して、マルチパス セッションを確立するために必要な iSCI LIF を検出します。 dataLIF が明示的に定義されている場合、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]。使用して `autoExportPolicy`そして `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`	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`が有効になります。使用して `autoExportPolicy`そして `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	説明	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。資格情報ベースの認証に使用されます。	
`svm`	使用するストレージ仮想マシン	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`そして `vsadmin` 集計使用量を取得し、 Trident を使用して制限するために必要な権限が含まれていません。	使用しないでください。
`limitVolumeSize`	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ボリュームあたりの最大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} 使用しないでください `debugTraceFlags` 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。	ヌル

パラメータ	説明	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 オプションを使用します。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 は関連付けられた永続ボリュームにマウント オプションを設定しません。	""
`nasType`	NFS 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。 設定する必要があります **`smb`SMB** ボリュームの場合。 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]の範囲でなければなりません	`"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 ONTAPバックエンドに必須です。	`smb-share`
`useREST`	ONTAP REST API を使用するためのブール パラメーター。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。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`応用。これは、事前に定義された `vsadmin`そして `cluster-admin`役割。	`false`

パラメータ	説明	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 リソーススネーム (ARN)。 - type : に設定 awsarn。参照"AWS Secrets Managerシークレットを作成する" 詳細についてはこちらをご覧ください。	

ボリュームのプロビジョニングのためのバックエンド構成オプション

デフォルトのプロビジョニングは、以下のオプションを使用して制御できます。 `defaults`構成のセクション。例については、以下の構成例を参照してください。

パラメータ	説明	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。「なし」（薄い）または「ボリューム」（厚い）	none
snapshotPolicy	使用するスナップショットポリシー	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ます。 Tridentで QoS ポリシー グループを使用するには、ONTAP 9.8 以降が必要です。共有されていない QoS ポリシー グループを使用し、ポリシー グループが各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	説明	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ます。 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`は `none、 else 「」
splitOnClone	クローン作成時に親からクローンを分割する	false
encryption	新しいボリュームでNetAppボリューム暗号化（NVE）を有効にします。デフォルトは false。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。バックエンドで NAE が有効になっている場合、 Tridentでプロビジョニングされたすべてのボリュームで NAE が有効になります。詳細については、以下を参照してください。"Trident がNVE および NAE と連携する仕組み" 。	false
luksEncryption	LUKS 暗号化を有効にします。参照"Linux Unified Key Setup (LUKS) を使用する"。 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サポート `mixed`そして `unix`セキュリティ スタイル。 SMBサポート `mixed`そして `ntfs`セキュリティ スタイル。	NFSのデフォルトは unix。 SMBのデフォルトは ntfs。

SMBボリュームのプロビジョニングの準備

SMBボリュームをプロビジョニングするには、 `ontap-nas`ドライバ。完了する前にONTAP SANおよびNASドライバーの統合次の手順を実行します。

開始する前に

SMBボリュームをプロビジョニングする前に、 `ontap-nas`ドライバーには次のものが必要です。

- Linux コントローラー ノードと、Windows Server 2019 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 資格情報を含む少なくとも 1 つのTridentシークレット。秘密を生成する `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。設定するには `csi-proxy`、参照"GitHub: CSIプロキシ"または"GitHub: Windows 用 CSI プロキシ"Windows 上で実行されている Kubernetes ノード用。

手順

1. SMB 共有を作成します。 SMB管理共有は、次の2つの方法のいずれかで作成できます。"Microsoft管理コンソール"共有フォルダ スナップインまたはONTAP CLI を使用します。 ONTAP CLI を使用して SMB 共有を作成するには、次の手順を実行します。

   a. 必要に応じて、共有のディレクトリ パス構造を作成します。

      その `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
```

   参照"SMB共有を作成する"詳細についてはこちらをご覧ください。

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。 FSx for ONTAPのバックエンド構成オプションについては、以下を参照してください。"FSx for ONTAP の構成オプションと例" 。

パラメータ	説明	例
`smbShare`	次のいずれかを指定できます: Microsoft 管理コンソールまたはONTAP CLI を使用して作成された SMB 共有の名前、またはTrident 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 ONTAPバックエンドに必須です。	`smb-share`

パラメータ	説明	例
nasType	設定する必要があります **smb.** null の場合、デフォルトは nfs。	smb
securityStyle	新しいボリュームのセキュリティスタイル。 設定する必要があります **`ntfs`**または **`mixed`SMB** ボリュームの場合。	`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。	""

ストレージクラスと**PVC**を構成する

Kubernetes StorageClass オブジェクトを構成し、ボリュームのプロビジョニング方法をTrident に指示するストレージ クラスを作成します。構成された 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。

ストレージクラスを作成する

Kubernetes StorageClassオブジェクトを構成する

その "Kubernetes StorageClassオブジェクト"オブジェクトは、そのクラスに使用されるプロビジョナーとしてTrident を識別し、ボリュームをプロビジョニングする方法をTridentに指示します。NFS を使用するボリュームの Storageclass を設定するには、この例を使用します (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`ストレージクラスに:

```
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
```

参照"KubernetesとTridentオブジェクト"ストレージクラスがどのように相互作用するかの詳細については、`PersistentVolumeClaim` 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ためのパラメータ。

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` Kubernetes で作成します。

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

2. これで、Kubernetes とTridentの両方に **basic-csi** ストレージ クラスが表示され、 Trident は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

あ "永続ボリュームクレーム"(PVC) は、クラスター上の 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。関連付けられた StorageClass を使用すると、クラスター管理者は PersistentVolume のサイズやアクセス モードだけでなく、パフォーマンスやサービス レベルなどの制御も行えます。

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に関連付けられています。 `basic-csi` 。

```
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に関連付けられています。 `protection-gold` 。

```
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

```
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
```

参照"KubernetesとTridentオブジェクト"ストレージクラスがどのように相互作用するかの詳細については、`PersistentVolumeClaim` 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ためのパラメータ。

Tridentの属性

これらのパラメータは、特定のタイプのボリューム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	支援
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
プロビジョニングタイプ	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	指定されたプロビジョニング方法	厚い：すべてオンタップ、薄い：すべてオンタップとsolidfireさん
バックエンドタイプ	string	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san、gcp-cvs、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	真、偽	プールはスナップショット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、gcp-cvs
クローン	ブール	真、偽	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、gcp-cvs

属性	タイプ	値	オファー	要求	支援
暗号化	ブール	真、偽	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	整数	正の整数	プールはこの範囲のIOPSを保証できる	ボリュームはこれらのIOPSを保証	solidfireさん

[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
```

進捗状況は以下で確認できます。 `kubectl get pod --watch` 。

2. ボリュームがマウントされていることを確認する `/my/mount/path`。

```
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
```

これでポッドを削除できます。 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 クラスターで**Trident EKS** アドオンを構成する

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

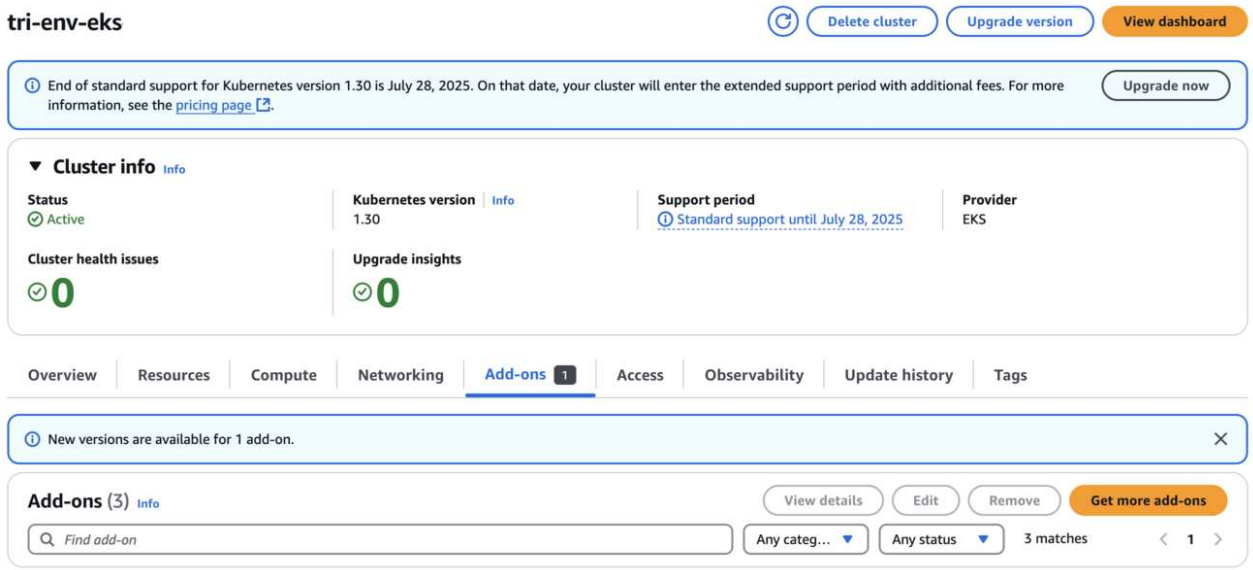
前提条件

AWS EKS のTridentアドオンを設定する前に、以下があることを確認してください。

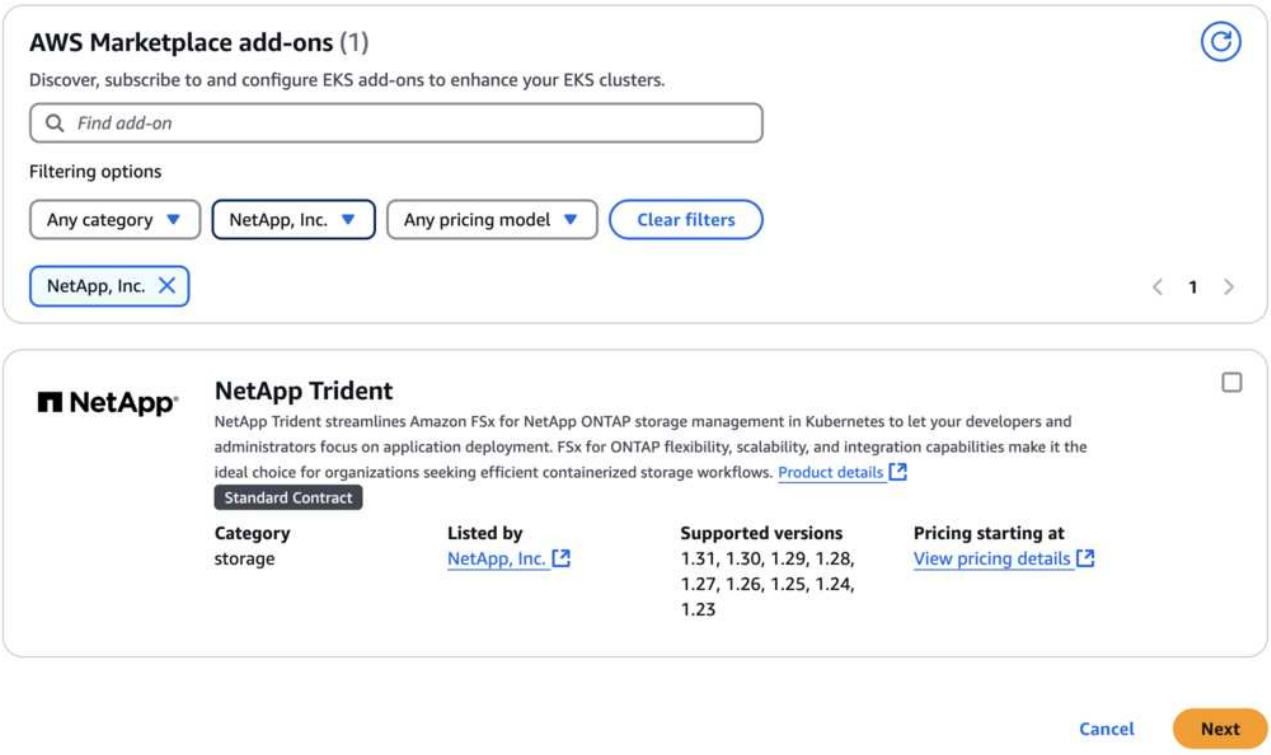
- アドオンを操作する権限を持つ Amazon EKS クラスターアカウント。参照"Amazon EKS アドオン"。
- AWS マーケットプレイスへの AWS 権限:
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI タイプ: Amazon Linux 2 (AL2_x86_64) または Amazon Linux 2 Arm(AL2_ARM_64)
- ノードタイプ: AMD またはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

手順

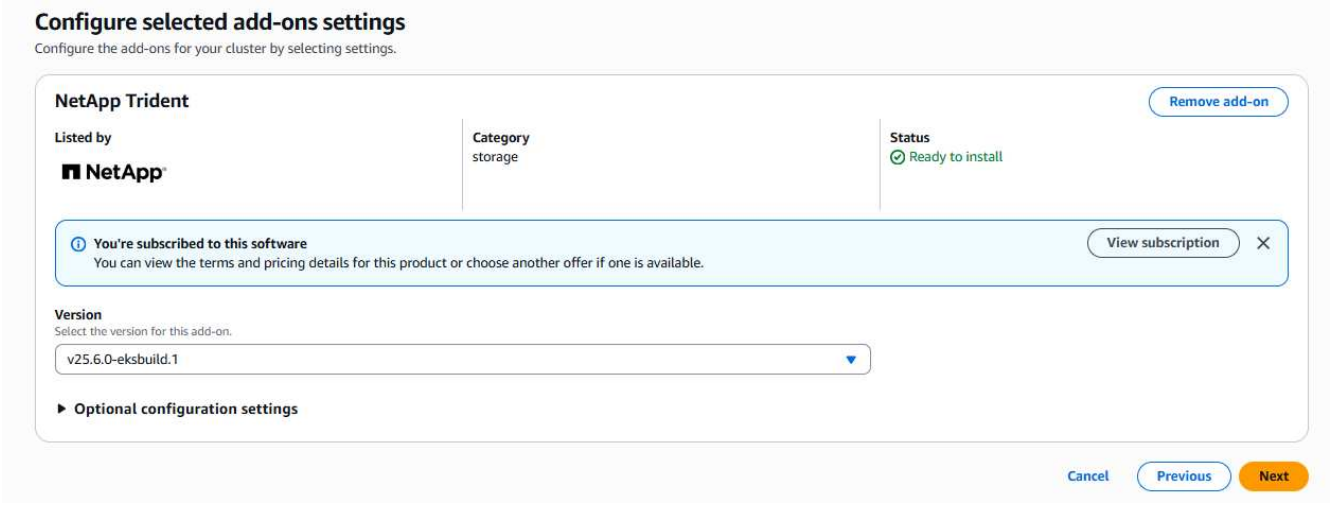
1. EKS ポッドが AWS リソースにアクセスできるようにするには、IAM ロールと AWS シークレットを作成してください。手順については、"IAMロールとAWSシークレットを作成する" 。
2. EKS Kubernetes クラスターで、[アドオン] タブに移動します。



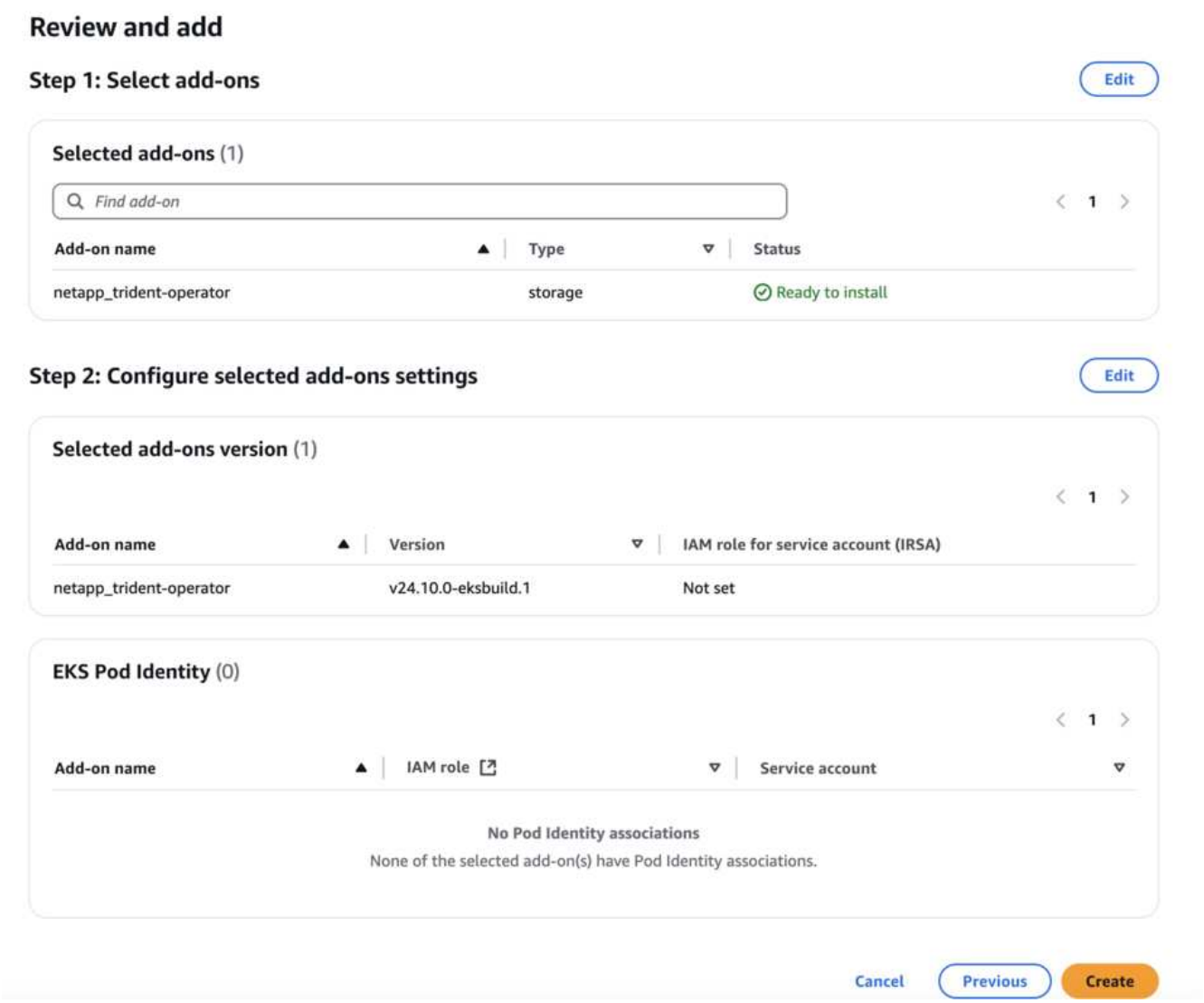
3. **AWS Marketplace** アドオン に移動し、*storage* カテゴリを選択します。



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. アドオンの希望するバージョンを選択します。

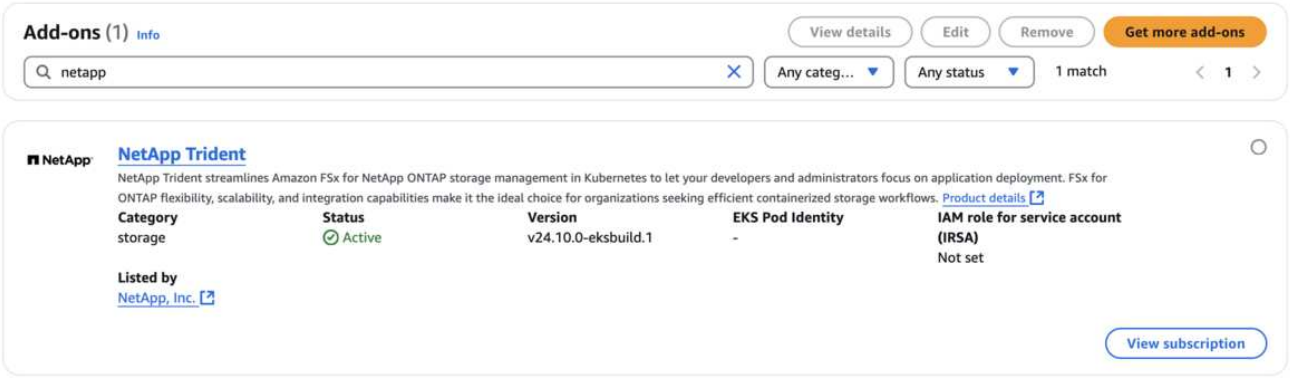


6. 必要なアドオン設定を構成します。



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. *作成*を選択します。

9. アドオンのステータスが アクティブ であることを確認します。



10. 次のコマンドを実行して、 Tridentがクラスター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構成する" 。

CLI を使用して**Trident EKS** アドオンをインストール/アンインストールする

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` （専用版あり）

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。これは、Trident にそのストレージ システムと通信する方法と、そこからボリュームをプロビジョニングする方法を指示します。 Tridentをインストールしたら、次のステップはバックエンドを作成することです。その `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Tridentバックエンドを直接作成および管理できます。これは次のように行うことができます `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。

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`、 `tbconfig` 、 `tbackendconfig`) は、 Tridentバックエンドを管理できるフロントエンドの名前空間 CRD です。 `kubectl` 。 Kubernetes 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。(`tridentctl` ）。

の作成時に `TridentBackendConfig`オブジェクトの場合、次のようになります。

- バックエンドは、指定した設定に基づいてTridentによって自動的に作成されます。これは内部的には `TridentBackend` (`tbe`、 `tridentbackend`) CR。
- その `TridentBackendConfig`一意に結びついている `TridentBackend`Tridentによって作成されました。

それぞれ `TridentBackendConfig`1対1のマッピングを維持する `TridentBackend`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現する方法です。

> ⚠ `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したい場合は、 `TridentBackendConfig`物体。

フォーマットについては次の例を参照してください。 `TridentBackendConfig` CR:

```
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、以下の例もご覧ください。 "トライデントインストーラー"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のディレクトリ。

その `spec`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 ドライバーであり、ここに表されている構成パラメータを使用します。ご希望のストレージドライバの設定オプションのリストについては、"ストレージドライバーのバックエンド構成情報" 。

その `spec`セクションには以下も含まれています `credentials`そして `deletionPolicy`新たに導入されたフィールド `TridentBackendConfig`CR:

- `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`削除されます。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、両方の `TridentBackendConfig`CR および関連するバックエンド。これがデフォルト値です。
- `retain`: とき `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`。削除ポリシーを設定する `retain`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`が作成されます。

バックエンドの名前は次のように設定されます。 `spec.backendName`。指定しない場合は、バックエンドの名前は `TridentBackendConfig`オブジェクト `(metadata.name)`。バックエンド名を明示的に設定することをお勧めします。 `spec.backendName`。

作成されたバックエンド `tridentctl`関連する `TridentBackendConfig`物体。このようなバックエンドを管理するには、 `kubectl`作成することで `TridentBackendConfig`CR。同一の設定パラメータ（例えば `spec.backendName`、`spec.storagePrefix`、 `spec.storageDriverName`、 等々）。Tridentは新しく作成された `TridentBackendConfig`既存のバックエンドを使用します。

手順の概要

新しいバックエンドを作成するには `kubectl`、次の操作を行う必要があります。

1. 作成する "Kubernetes シークレット"シークレットには、 Trident がストレージ クラスター/サービスと通信するために必要な資格情報が含まれています。
2. 作成する `TridentBackendConfig`物体。これには、ストレージ クラスター/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、次のようにしてそのステータスを確認できます。 `kubectl get tbc <tbc-name> -n <trident-namespace>`追加の詳細情報を収集します。

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秘密フィールドの説明	秘密	フィールドの説明
Azure NetApp Files	クライアントID	アプリ登録からのクライアントID
Cloud Volumes Service for GCP	秘密鍵ID	秘密鍵のID。 CVS 管理者ロールを持つ GCP サービス アカウントの API キーの一部
Cloud Volumes Service for GCP	秘密鍵	秘密鍵。 CVS 管理者ロールを持つ GCP サービス アカウントの API キーの一部
エレメント (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。資格情報ベースの認証に使用される
ONTAP	パスワード	クラスター/SVM に接続するためのパスワード。資格情報ベースの認証に使用される
ONTAP	クライアント秘密鍵	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る
ONTAP	chapユーザー名	受信ユーザー名。 useCHAP=true の場合は必須です。のために `ontap-san`そして `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP イニシエーター シークレット。 useCHAP=true の場合は必須です。のために `ontap-san`そして `ontap-san-economy`
ONTAP	chapターゲットユーザー名	ターゲットユーザー名。 useCHAP=true の場合は必須です。のために `ontap-san`そして `ontap-san-economy`
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。のために `ontap-san`そして `ontap-san-economy`

このステップで作成されたシークレットは、 `spec.credentials`の分野 `TridentBackendConfig`次のステップで作成されるオブジェクト。

ステップ2: 作成する `TridentBackendConfig`CR

これで作成準備ができました `TridentBackendConfig`CR。この例では、 `ontap-san`ドライバーは、 `TridentBackendConfig`以下に示すオブジェクト:

```
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: ステータスを確認する `TridentBackendConfig`CR

これで、 `TridentBackendConfig` CR、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。

フェーズは次のいずれかの値を取ることができます。

- `Bound`：その `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`に設定 `TridentBackendConfig`CR の uid。
- `Unbound`: 表現方法 `""`。その `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。すべて新しく作成されました `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、再びアンバウンドに戻ることはできません。
- `Deleting`：その `TridentBackendConfig`CRの `deletionPolicy`削除するように設定されました。いつ

`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` Tridentはバックエンドと `TridentBackendConfig`CR。
- 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。その `TridentBackendConfig` その後、CR も削除フェーズに入ります。バックエンドと `TridentBackendConfig`すべての PVC が削除された後にのみ削除されます。

- `Lost`: に関連付けられたバックエンド `TridentBackendConfig`CR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。その `TridentBackendConfig`CRは、 `deletionPolicy`価値。
- `Unknown`: Tridentは、関連付けられたバックエンドの状態または存在を判別できません。 `TridentBackendConfig` CR。たとえば、APIサーバーが応答しない場合や、 `tridentbackends.trident.netapp.io` CRD がありません。これには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`含まれている `backendName`そして `backendUUID`バックエンドは、`TridentBackendConfig CR。その lastOperationStatus`フィールドは、最後の操作のステータスを表します。 `TridentBackendConfig CRはユーザーによってトリガーされる（例えば、ユーザーが spec)またはTridentによってトリガーされます (たとえば、 Trident の再起動時)。成功または失敗のいずれかになります。 phase`の関係の状態を表します `TridentBackendConfig`CR とバックエンド。上記の例では、`phase`値はBoundです。つまり、 `TridentBackendConfig CR はバックエンドに関連付けられています。

実行することができます `kubectl -n trident describe tbc <tbc-cr-name>`イベント ログの詳細を取得するコマンド。

> 関連付けられたバックエンドを更新または削除することはできません。TridentBackendConfig`オブジェクト使用 `tridentctl。切り替え手順を理解するにはtridentctl`そして `TridentBackendConfig、"こちらをご覧ください"。

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

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。 kubectl 。

バックエンドを削除する

削除することで TridentBackendConfig、 Tridentにバックエンドを削除/保持するように指示します（ deletionPolicy ）。バックエンドを削除するには、 deletionPolicy`削除するように設定されています。削除するには `TridentBackendConfig、必ず deletionPolicy`保持するように設定されています。これにより、バックエンドがまだ存在し、以下を使用して管理できることが保証されます。 `tridentctl 。

次のコマンドを実行します。

```
kubectl delete tbc <tbc-name> -n trident
```

Tridentは、使用されていたKubernetes Secretsを削除しません。 TridentBackendConfig 。 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
```

実行することもできます tridentctl get backend -n trident`または `tridentctl get backend -o yaml -n trident`存在するすべてのバックエンドのリストを取得します。このリストには、 `tridentctl 。

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 システムへの資格情報が変更されました。資格情報を更新するには、 `TridentBackendConfig` オブジェクトを更新する必要があります。 Trident は、提供された最新の資格情報を使用してバックエンドを自動的に更新します。 Kubernetes シークレットを更新するには、次のコマンドを実行します。

  ```
  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。
  - 更新できます `TridentBackendConfig`次のコマンドを使用して、Kubernetes 経由でオブジェクトを直接実行します。

```
kubectl apply -f <updated-backend-file.yaml>
```

- あるいは、既存の `TridentBackendConfig`次のコマンドを使用して CR を実行します。

```
kubectl edit tbc <tbc-name> -n trident
```

- バックエンドの更新が失敗した場合、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、次のコマンドを実行します。 `kubectl get tbc <tbc-name> -o yaml -n trident`または `kubectl describe tbc <tbc-name> -n trident`。
- 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

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。 `tridentctl`。

バックエンドを作成する

作成後"バックエンド設定ファイル"、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`再度コマンドを実行します。

バックエンドを削除する

Tridentからバックエンドを削除するには、次の手順を実行します。

1. バックエンド名を取得します。

```
tridentctl get backend -n trident
```

2. バックエンドを削除します。

```
tridentctl delete backend <backend-name> -n trident
```

Trident 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、新しいボリュームがプロビジョニングされなくなります。バックエンドは「削除中」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、次の手順を実行します。

- 概要を取得するには、次のコマンドを実行します。

```
tridentctl get backend -n trident
```

- すべての詳細を取得するには、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`再度コマンドを実行します。

バックエンドを使用するストレージクラスを識別する

これはJSONで答えられる質問の例です。 `tridentctl`バックエンド オブジェクトの出力。これは、 `jq`インストールする必要があるユーティリティ。

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

これは、以下を使用して作成されたバックエンドにも適用されます。 `TridentBackendConfig` 。

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学びます。

の導入により `TridentBackendConfig`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。

- バックエンドは以下を使用して作成できます `tridentctl`管理される `TridentBackendConfig`?
- バックエンドは以下を使用して作成できます `TridentBackendConfig`管理するには `tridentctl`?

管理 `tridentctl`バックエンドを使用する `TridentBackendConfig`

このセクションでは、以下の手順で作成されたバックエンドを管理するために必要な手順について説明します。 `tridentctl` Kubernetesインターフェースから直接作成することで `TridentBackendConfig`オブジェクト。

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`なぜなら、それらは `tridentctl`。
- 作成された新しいバックエンド `tridentctl`、他の `TridentBackendConfig`オブジェクトが存在します。

どちらのシナリオでも、バックエンドは引き続き存在し、 Tridentがボリュームをスケジュールして操作します。管理者はここで 2 つの選択肢のいずれかを選択できます。

- 引き続き使用 `tridentctl`これを使用して作成されたバックエンドを管理します。
- 作成したバックエンドをバインドする `tridentctl`新しい `TridentBackendConfig`物体。そうすることで、バックエンドは次のように管理されることになります。 `kubectl`そしてそうではない `tridentctl`。

既存のバックエンドを管理するには `kubectl`を作成する必要があります `TridentBackendConfig`既存のバックエンドにバインドします。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。シークレットには、 Trident がストレージ クラスター/サービスと通信するために必要な資格情報が含まれています。
2. 作成する `TridentBackendConfig`物体。これには、ストレージ クラスター/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（例えば `spec.backendName`、 `spec.storagePrefix` 、 `spec.storageDriverName` 、 等々）。 `spec.backendName`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

を作成するには `TridentBackendConfig`既存のバックエンドにバインドする場合は、バックエンド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:** 作成する **`TridentBackendConfig`CR**

次のステップは、 `TridentBackendConfig`既存のCRに自動的にバインドされる `ontap-nas-backend`(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定義されている `spec.backendName`。
- 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。
- 資格情報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`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: ステータスを確認する `TridentBackendConfig`CR

その後 `TridentBackendConfig`作成された場合、そのフェーズは `Bound。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`物体。

管理 `TridentBackendConfig`バックエンドを使用する `tridentctl`

`tridentctl`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 `TridentBackendConfig`。さらに、管理者は、次のようなバックエンドを完全に管理することもできます。 `tridentctl`削除することで `TridentBackendConfig`そして確認する `spec.deletionPolicy`設定されている `retain`。

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次のように作成されたとします。 `TridentBackendConfig` :

```
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`正常に作成され、バックエンドにバインドされています [バックエンドの UUID を確認してください]。

ステップ**1:** 確認 `deletionPolicy`設定されている `retain`

の価値を見てみましょう `deletionPolicy`。これを設定する必要があります `retain`。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` 。

```
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
```

以下の場合を除き、次のステップに進まないでください。 `deletionPolicy`設定されている `retain`。

ステップ**2:** 削除する **`TridentBackendConfig`CR**

最後のステップは、 `TridentBackendConfig` CR。確認後、 `deletionPolicy`設定されている `retain`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 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

削除すると、 `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、単にオブジェクトを削除します。

ストレージクラスの作成と管理

ストレージクラスを作成する

Kubernetes StorageClass オブジェクトを構成し、ボリュームのプロビジョニング方法をTrident に指示するストレージ クラスを作成します。

Kubernetes StorageClassオブジェクトを構成する

その "Kubernetes StorageClassオブジェクト"そのクラスに使用されるプロビジョナーとしてTrident を識別し、ボリュームをプロビジョニングする方法をTrident に指示します。例えば：

```
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
```

参照"KubernetesとTridentオブジェクト"ストレージクラスがどのように相互作用するかの詳細については、`PersistentVolumeClaim` 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ためのパラメータ。

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]使用したり変更したりできるいくつかの基本的なサンプルを提供します。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` Kubernetes で作成します。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

2. これで、Kubernetes とTridentの両方に **basic-csi** ストレージ クラスが表示され、 Trident は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"。

あるいは、編集することもできます `sample-input/storage-class-csi.yaml.templ`インストーラーに付属のファイルと置き換えます `*BACKEND_TYPE*`ストレージ ドライバー名を使用します。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

ストレージクラスの管理

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、ユーザーが永続ボリューム要求 (PVC) でストレージ クラスを指定しない場合に、永続ボリューム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。

- アノテーションを設定してデフォルトのストレージクラスを定義します `storageclass.kubernetes.io/is-default-class`ストレージ クラス定義で true に設定します。仕様によれば、その他の値または注釈の欠如は 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"}}}'
```

Tridentインストーラー バンドルにも、この注釈を含む例があります。

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JSONで答えられる質問の例です。 `tridentctl` Tridentバックエンド オブジェクトの出力。これは、`jq`ユーティリティを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>`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 クラスを通じて作成された永続ボリュームはそのまま残り、 Tridentによって引き続き管理されます。

> Tridentは空白を強制する `fsType`作成されるボリュームに対して。 iSCSIバックエンドの場合、強制することが推奨されます `parameters.fsType`StorageClass 内。既存のStorageClassesを削除して、 `parameters.fsType`指定された。

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

構成された 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

あ "永続ボリュームクレーム"(PVC) は、クラスター上の 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。関連付けられた StorageClass を使用すると、クラスター管理者は PersistentVolume のサイズやアクセス モードだけでなく、パフォーマンスやサービス レベルなどの制御も行えます。

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

```
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
```

進捗状況は以下で確認できます。 `kubectl get pod --watch` 。

2. ボリュームがマウントされていることを確認する `/my/mount/path`。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. これでポッドを削除できます。 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、StorageClassに関連付けられています。 `basic-csi` 。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe/TCP 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を示しており、これはStorageClassに関連付けられています。 `protection-gold` 。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

ポッド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構成を示しています。

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
```

参照"KubernetesとTridentオブジェクト"ストレージクラスがどのように相互作用するかの詳細については、`PersistentVolumeClaim` 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ためのパラメー

タ。

ボリュームを拡張する

Trident は、Kubernetes ユーザーにボリュームを作成後に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`ドライバーと 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を構成する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドへ `true`。

```
cat storageclass-ontapsan.yaml
```

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

既存のStorageClassの場合は、 `allowVolumeExpansion`パラメータ。

ステップ2: 作成したStorageClassでPVCを作成する

PVC定義を編集して更新します `spec.resources.requests.storage`新しく希望するサイズを反映するため、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
```

ステップ**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 バックエンド上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` 。

```
 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` 2Giまで。

```
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かどうかを確認できます。

```
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`ドライバーであり、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**を構成する

StorageClass定義を編集して、 `allowVolumeExpansion`フィールドへ `true`。

```
cat storageclass-ontapsan.yaml
```

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

既存のStorageClassの場合は、 `allowVolumeExpansion`パラメータ。

ステップ2: 作成したStorageClassでPVCを作成する

PVC定義を編集して更新します `spec.resources.requests.storage`新しく希望するサイズを反映するため、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
```

ステップ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 バックエンド上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` 。

```
 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` 2Giまで。

```
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かどうかを確認できます。

```
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、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ます。 `ontap-nas` 、 `ontap-nas-economy` 、 `ontap-nas-flexgroup` 、 `gcp-cvs` 、 そして `azure-netapp-files`バックエンド。

ステップ1: ボリューム拡張をサポートするようにStorageClassを構成する

NFS PVのサイズを変更するには、管理者はまず、ボリューム拡張を可能にするためにストレージクラスを設定する必要があります。 `allowVolumeExpansion`フィールドへ `true`:

```
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、既存のストレージクラスを編集するだけで済みます。 `kubectl edit storageclass`ボリュームの拡張を可能にします。

ステップ**2:** 作成した**StorageClass**で**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を編集して設定します。
`spec.resources.requests.storage` 1 GiBまで:

```
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行われたかどうかを確認できます。

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

輸入量

`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

ボリュームをTridentにインポートすると、次のような効果が得られます。

- 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クラスターを再構築する
- 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 はRW (読み取り/書き込み) タイプのONTAPボリュームのみをインポートできます。 DP (データ保護) タイプのボリュームは、 SnapMirror の宛先ボリュームです。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。これによりデータが破損する可能性があります。

- けれど `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利用可能なプールから選択するために使用されます。ボリュームは既に存在するため、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 ドライバーによってインポートされた後、PVC への ClaimRef を使用して PV が作成されます。
  - 回収ポリシーは初期設定では `retain`PVでは。 Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーはストレージクラスの再利用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`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 はPVC を管理し、バックエンドのFlexVol volumeと LUN の名前を変更します。あなたは合格することができます `--no-manage`管理されていないボリュームをインポートするためのフラグ。使用する場合 `--no-manage`Trident は、オブジェクトのライフサイクル中に PVC または PV に対して追加の操作を実行しません。 PV が削除されてもストレージ ボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などの他の操作も無視されます。

このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、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. 使用 `tridentctl import`ボリュームを含むTridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前 (例: ONTAP FlexVol、Element Volume、 Cloud Volumes Serviceパス) を指定するコマンド。その `-f`PVC ファイルへのパスを指定するには引数が必要です。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

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については、次のボリューム インポート例を確認してください。

ONTAP NAS およびONTAP NAS FlexGroup

Tridentはボリュームインポートをサポートしており、 `ontap-nas`そして `ontap-nas-flexgroup`ドライバー。

- Tridentは、 `ontap-nas-economy` ドライバ。
- その `ontap-nas`そして `ontap-nas-flexgroup`ドライバーは重複したボリューム名を許可しません。

各ボリュームは、 `ontap-nas`ドライバーは、 `ONTAP`クラスタ上の`FlexVol volume`です。 `FlexVol`ボリュームをインポートするには `ontap-nas`ドライバーは同じように動作します。 `ONTAP`クラスタにすでに存在する`FlexVol`ボリュームは、 `ontap-nas` PVC。同様に、 FlexGroupボリュームは次のようにインポートできます。 `ontap-nas-flexgroup` PVC。

tridentctl を使用した ONTAP NAS の例

以下に、管理対象ボリュームと管理対象外ボリュームのインポートの例を示します。

管理ボリューム

次の例では、 `managed_volume`バックエンドで `ontap_nas`:

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理されていないボリューム

使用する際は `--no-manage`引数が指定されていない場合、 Trident はボリュームの名前を変更しません。

次の例では、 `unmanaged_volume`上の `ontap_nas`バックエンド:

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

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はボリュームインポートをサポートしており、 `ontap-san` (iSCSI、NVMe/TCP、FC)および `ontap-san-economy` ドライバー。

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` ドライバは、各 PVC に対してFlexVol volumeを作成し、 FlexVol volume内に LUN を作成します。TridentはFlexVol volumeをインポートし、それを PVC 定義に関連付けます。Tridentは輸入できる `ontap-san-economy` 複数の LUN を含むボリューム。

ONTAP SANの例

次の例は、管理対象ボリュームと管理対象外ボリューム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 volumeの名前を `pvc-<uuid>`FlexVol volume内のフォーマットとLUNを `lun0`。

次の例では、 `ontap-san-managed` FlexVol volumeは、 `ontap_san_default`バックエンド:

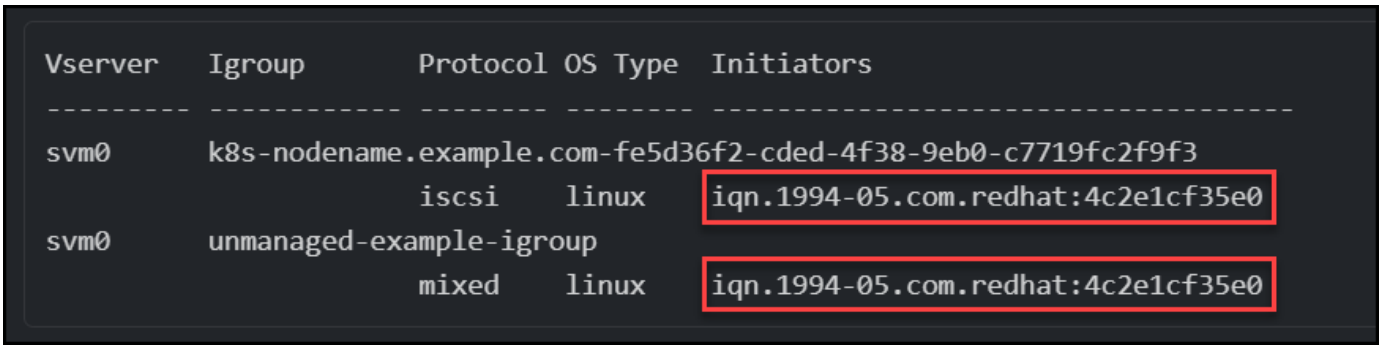
```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理されていないボリューム

次の例では、 `unmanaged_example_volume`上の `ontap_san`バックエンド:

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

次の例に示すように、Kubernetes ノード 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。



要素

TridentはNetApp Elementソフトウェアと NetApp HCIボリュームのインポートをサポートしています。 `solidfire-san`ドライバ。

> Element ドライバーは重複したボリューム名をサポートします。ただし、ボリューム名が重複している場合、 Trident はエラーを返します。回避策として、ボリューム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。

次の例では、 `element-managed`バックエンドのボリューム `element_default。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+----------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+----------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+----------+
```

Google Cloud Platform

Tridentはボリュームインポートをサポートしており、 `gcp-cvs`ドライバ。

> Google Cloud Platform でNetApp Cloud Volumes Serviceによってバックアップされた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、ボリュームのエクスポートパスの `:/`。たとえば、エクスポートパスが `10.0.0.1:/adroit-jolly-swift`ボリュームパスは `adroit-jolly-swift`。

Google Cloud Platform の例

次の例では、 `gcp-cvs`バックエンドのボリューム `gcpcvs_YEppr`ボリュームパスの `adroit-jolly-swift。

```
tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-
file> -n trident

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Tridentはボリュームインポートをサポートしており、 `azure-netapp-files`ドライバ。

> 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、ボリュームのエクスポートパスの `:/`。たとえば、マウントパスが `10.0.0.2:/importvol1`ボリュームパスは `importvol1。

次の例では、 `azure-netapp-files`バックエンドのボリューム `azurenetappfiles_40517`ボリュームパス `importvol1。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetApp Volumes

Tridentはボリュームインポートをサポートしており、 `google-cloud-netapp-volumes`ドライバ。

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、 `google-cloud-netapp-volumes`同じリージョンに 2 つのボリュームが存在する場合のボリューム:

```
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:

1. ボリュームの作成、インポート、およびクローン操作。
2. ontap-nas-economy ドライバーの場合、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。
3. ontap-san-economy ドライバーの場合、LUN 名のみが名前テンプレートに準拠します。

制限事項

1. カスタマイズ可能なボリューム名は、 ONTAPオンプレミス ドライバーとのみ互換性があります。
2. カスタマイズ可能なボリューム名は既存のボリュームには適用されません。

カスタマイズ可能なボリューム名の主な動作

1. 名前テンプレートの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が失敗した場合、ボリュームは既存の命名規則に従って名前が付けられます。
2. バックエンド構成の名前テンプレートを使用してボリューム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 レベルで定義できます。

ルートレベルの例

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

プールレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

名前テンプレートの例

例**1**:

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

例**2**:

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

考慮すべき点

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 プレフィックス付き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、 Trident はランダムな文字をいくつか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の長さが 64 文字を超える場合、 Trident は既存の命名規則に従ってボリュームに名前を付けます。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は失敗します。

名前空間間で**NFS**ボリュームを共有する

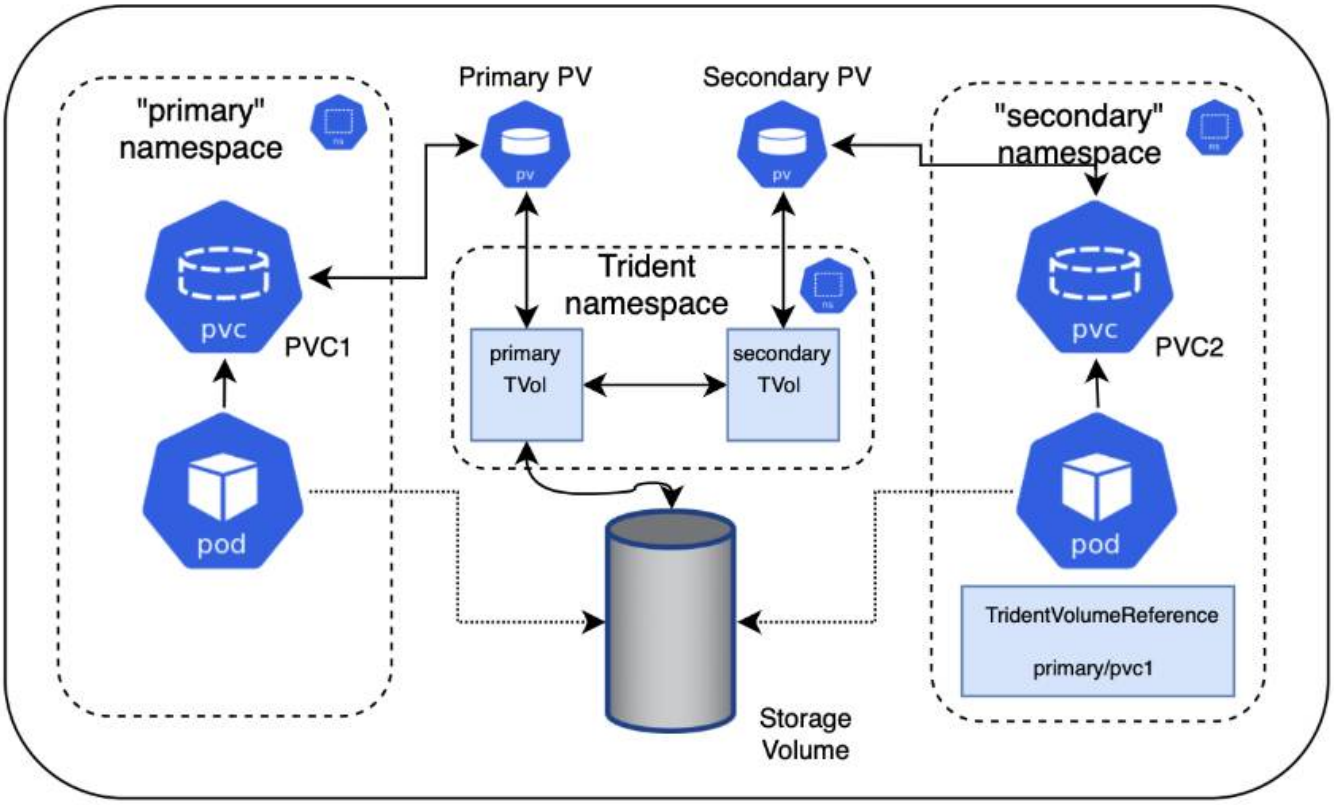
Trident を使用すると、プライマリ名前空間にボリュームを作成し、それを 1 つ以上のセカンダリ名前空間で共有できます。

機能

TridentVolumeReference CR を使用すると、1 つ以上の Kubernetes 名前空間間で ReadWriteMany (RWX) NFS ボリュームを安全に共有できます。この Kubernetes ネイティブ 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 ドライバーで動作します
- 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**を構成する

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 宛先名前空間に **CR** を作成する権限を付与する

クラスター管理者は、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 を参照するための TridentVolumeReference CR を作成します。

4 宛先名前空間に従属**PVC**を作成する

宛先名前空間の所有者は、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。

ソースと宛先の名前空間を構成する

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. ソース名前空間の所有者: PVCを作成する(`pvc1`）をソース名前空間に作成し、宛先名前空間と共有する権限を付与する(`namespace2`）を使用して `shareToNamespace`注釈。

```
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、 `*` 。例えば、`trident.netapp.io/shareToNamespace: *`
- PVCを更新して、 `shareToNamespace`いつでも注釈を付けることができます。

2. クラスター管理者: 宛先名前空間の所有者に宛先名前空間に TridentVolumeReference CR を作成するための権限を付与するための適切な RBAC が設定されていることを確認します。
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. 宛先名前空間の所有者: PVCを作成する(`pvc2`）を宛先名前空間に(`namespace2`）を使用して `shareFromPVC`送信元 PVC を指定するための注釈。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

宛先 PVC のサイズは、送信元 PVC のサイズ以下である必要があります。

結果

Tridentは `shareFromPVC`宛先 PVC にアノテーションを追加し、宛先 PV を、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、独自のストレージ リソースを持たない従属ボリュームとして作成します。宛先 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、ソース名前空間上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。

使用 `tridentctl get`従属ボリュームを照会する

使用して[`tridentctl`ユーティリティを実行すると、 `get`従属ボリュームを取得するコマンド。詳細については、次のリンクを参照してください:../trident-reference/tridentctl.html[`tridentctl`コマンドとオプション]。

```
Usage:
  tridentctl get [option]
```

フラグ:

- `-h, --help`: ボリュームのヘルプ。
- `--parentOfSubordinate string`: クエリを従属ソース 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、宛先名前空間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へのアクセスは、 `shareToNamespace`または `shareFromNamespace`注釈や削除 `TridentVolumeReference`CR。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、以下を参照してください。

- 訪問"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"。
- デモを見る"ネットアップTV"。

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`そして `ontap-nas`ストレージ ドライバー。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. ボリュームを複製するためのソース**PVC**を構成する

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. 宛先名前空間に **CR** を作成する権限を付与する

クラスター管理者は、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 を参照するための TridentVolumeReference CR を作成します。

4. 宛先名前空間にクローン**PVC**を作成する

宛先名前空間の所有者は、ソース名前空間から PVC を複製するための PVC を作成します。

ソースと宛先の名前空間を構成する

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. ソース名前空間の所有者: PVCを作成する(`pvc1`）ソース名前空間(`namespace1`）は、宛先名前空間と共有する権限を付与します(`namespace2`）を使用して `cloneToNamespace`注釈。

```
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、 `*` 。例えば、`trident.netapp.io/cloneToNamespace: *`
- PVCを更新して、 `cloneToNamespace`いつでも注釈を付けることができます。

2. クラスタ管理者: 宛先名前空間の所有者に、宛先名前空間に TridentVolumeReference CR を作成する権限を付与するための適切な RBAC が設定されていることを確認します。(`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. 宛先名前空間の所有者: PVCを作成する(`pvc2`）を宛先名前空間に(`namespace2`）を使用して `cloneFromPVC`または `cloneFromSnapshot`、 そして `cloneFromNamespace`ソース PVC を指定するための注釈。

```
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を使用してボリュームを複製する

Trident は、災害復旧のためにデータを複製するために、1 つのクラスター上のソース ボリュームとピア クラスター上の宛先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

- 鏡の関係を破る
- 災害発生時にセカンダリボリュームを昇格する（フェイルオーバー）
- 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使用するソース Kubernetes クラスターと宛先 Kubernetes クラスターの両方に、 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: データ保護バンドルを使用するONTAP SnapMirror非同期ライセンスは、ソースと宛先の両方のONTAPクラスタで有効にする必要があります。参照 "ONTAPにおけるSnapMirrorライセンスの概要"詳細についてはこちらをご覧ください。

  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。参照"ONTAP Oneに含まれるライセンス"詳細についてはこちらをご覧ください。

  SnapMirror非同期保護のみがサポートされます。

ピアリング

- クラスタと **SVM**: ONTAPストレージ バックエンドをピアリングする必要があります。参照 "クラスタとSVMのピアリングの概要"詳細についてはこちらをご覧ください。

  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ます。

- * Tridentと SVM*: ピア接続されたリモート SVM は、宛先クラスター上の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ー

NetApp Trident は、次のドライバー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 (最低でも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、以下を参照してください。"ASA r2 ストレージシステムについて学ぶ" 。

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 クラスターで次の手順を実行します。
   a. StorageClassオブジェクトを作成し、 `trident.netapp.io/replication: true`パラメータ。

      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. ローカル情報を使用して 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 のステータス 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 クラスター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 を作成します。

例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

b. 宛先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) を使用して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、セカンダリ (SnapMirror の宛先) として機能する PVC を作成します。

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 をチェックし、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 volumeが、MirrorRelationship で定義されたリモート SVM とピアリングされている SVM 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 Mirror Relationship (TMR) は、PVC 間のレプリケーション関係の一方の端を表す CRD です。宛先 TMR には状態があり、これによってTrident に目的の状態が伝えられます。宛先 TMR の状態は次のとおりです。

- 確立済み: ローカル PVC はミラー関係の宛先ボリュームであり、これは新しい関係です。
- 昇格: ローカル PVC は読み取り/書き込み可能でマウント可能であり、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 はミラー関係の宛先ボリュームであり、以前もそのミラー関係にありました。
  - 宛先ボリュームがソース ボリュームと関係があった場合、宛先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 クラスターで次の手順を実行します。

- TridentMirrorRelationshipの_spec.state_フィールドを次のように更新します。 `promoted` 。

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 クラスターで、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 クラスターで、内部詳細を取得するための SnapshotInfo CR を作成します。

   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 クラスターで、*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 クラスターで、TridentMirrorRelationship のステータス (status.state フィールド) が昇格され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

ミラー関係を復元する前に、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 クラスターで、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_フィールドを次のように更新します。 `reestablished` 。

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**に複製する

プライマリ PVC とセカンダリ PVC がすで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 (宛先) クラスターから PersistentVolumeClaim および TridentMirrorRelationship CRD を削除します。
2. プライマリ (ソース) クラスターから TridentMirrorRelationship CRD を削除します。
3. 確立する新しいセカンダリ (宛先) PVC のプライマリ (ソース) クラスターに新しい 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 は通常どおりサイズを変更できますが、データの量が現在のサイズを超えると、 ONTAP は宛先 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 ボリュームに対して次のいずれかの操作を実行します。

- セカンダリ PVC 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

ミラー関係の一方の TMR を削除すると、 Trident が削除を完了する前に、残りの TMR が *promoted* 状態に移行します。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、ボリュームのレプリケーション状態が確立された新しい 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。使用することができます `state: promoted`または `state: reestablished`関係を更新するためのフィールド。宛先ボリュームを通常の ReadWrite ボリュームに昇格する場合、*promotedSnapshotHandle* を使用して、現在のボリュームを復元する特定のスナップショットを指定できます。

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されていなくても、 SnapMirror の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`TridentActionMirrorUpdate CRD の状態を反映します。 *Succeeded*、*In Progress*、*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、 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。 "CSIトポロジ機能" 。

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 アーキテクチャでの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。

- と `VolumeBindingMode`に設定 `Immediate`Trident はトポロジを考慮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。これはデフォルトです `VolumeBindingMode`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。
- と `VolumeBindingMode`に設定 `WaitForFirstConsumer`、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。

その `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。

- Kubernetesクラスタは"サポートされているKubernetesバージョン"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードにはトポロジー認識を導入するラベルが必要です (`topology.kubernetes.io/region`そして `topology.kubernetes.io/zone`）。 Tridentがトポロジを認識するためには、 Tridentがインストールされる前に、これらのラベルが クラスター内のノード上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。各バックエンドはオプションで `supportedTopologies`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ブロック。このようなバックエンドを利用する 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。

ヤムル

```
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

`supportedTopologies`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 StorageClasses の場合、 Trident はバックエンドにボリュームを作成します。

定義できます `supportedTopologies`ストレージ プールごとにも同様です。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`そして `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`そして `topology.kubernetes.io/zone`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:** トポロジーを考慮したストレージクラスを定義する

クラスター内のノードに提供されるトポロジ ラベルに基づいて、トポロジ情報を格納するように StorageClasses を定義できます。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`設定されている `WaitForFirstConsumer`。この StorageClass で要求された PVC は、ポッドで参照されるまで処理されません。そして、 `allowedTopologies`使用するゾーンとリージョンを提供します。その `netapp-san-us-east1`StorageClassはPVCを `san-backend-us-east1`上記で定義されたバックエンド。

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ので、PVC を作成できるようになりました。

例を見る `spec`下に：

```
---
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 が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、 `us-east1`地域を選択し、その地域にある任意のノードから選択します。 `us-east1-a`または `us-east1-b`ゾーン。

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含める `supportedTopologies`

既存のバックエンドを更新して、 `supportedTopologies`使用して `tridentctl backend update`。これは、すでにプロビジョニングされているボリュームには影響せず、後続の 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"
- "ノードセレクタ"
- "親和性と反親和性"
- "汚名と寛容"

スナップショットの操作

永続ボリューム (PV) の Kubernetes ボリューム スナップショットにより、ボリュームのポイントインタイム 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の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 データを回復したりできます。

概要

ボリュームスナップショットは以下でサポートされています `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `gcp-cvs`、 `azure-netapp-files`、 そして `google-cloud-netapp-volumes`ドライバー。

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、[ボリュームスナップショットコントローラを展開する] 。

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。

ボリュームスナップショットを作成する

手順

1. 作成する `VolumeSnapshotClass`詳細については、"ボリュームスナップショットクラス" 。
   - その `driver`Trident CSI ドライバーを指します。
   - `deletionPolicy`できる `Delete`または `Retain`。に設定すると `Retain`ストレージクラスター上の基礎となる物理スナップショットは、 `VolumeSnapshot`オブジェクトは削除されます。

     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、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ます。 `pvc1`スナップショットの名前は `pvc1-snap`。 VolumeSnapshotはPVCに類似しており、 `VolumeSnapshotContent`実際のスナップショットを表すオブジェクト。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- あなたは、 `VolumeSnapshotContent`のオブジェクト `pvc1-snap`VolumeSnapshot を記述します。その `Snapshot Content Name`このスナップショットを提供する VolumeSnapshotContent オブジェクトを識別します。その `Ready To Use`パラメーターは、スナップショットを使用して新しい PVC を作成できることを示します。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

ボリュームスナップショットから**PVC**を作成する

使用できます `dataSource`VolumeSnapshotを使用してPVCを作成します `<pvc-name>`データのソースとして。 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。参照"KB: Trident PVC スナップショットから PVC を作成すると、代替バックエンドでは作成できません"。

次の例では、PVCを作成します。 `pvc1-snap`データソースとして。

```
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
```

ボリュームスナップショットをインポートする

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"クラスタ管理者が`VolumeSnapshotContent`オブジェクトを作成し、 Tridentの外部で作成されたスナップショットをインポートします。

開始する前に

Trident は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いる必要があります。

手順

1. クラスタ管理者: `VolumeSnapshotContent`バックエンドスナップショットを参照するオブジェクト。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。
   - バックエンドスナップショットの名前を指定します `annotations`として `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 特定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`で `snapshotHandle`これは、外部スナップショット装置からTridentに提供される唯一の情報です。 `ListSnapshots`電話。

     その `<volumeSnapshotContentName>`CR 命名制約により、バックエンド 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、 `VolumeSnapshotContent`バックエンドスナップショットを参照するオブジェクト `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. クラスタ管理者: `VolumeSnapshot` CRは、 `VolumeSnapshotContent`物体。これは、 `VolumeSnapshot`特定の名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap`を参照する `VolumeSnapshotContent`名前 `import-snap-content`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. 内部処理（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`そして、 `ListSnapshots`電話。Tridentは `TridentSnapshot`。
   - 外部スナップショットは、 `VolumeSnapshotContent`に `readyToUse`そして `VolumeSnapshot`に `true`。
   - Tridentが復活 `readyToUse=true`。
4. 任意のユーザー: 作成 `PersistentVolumeClaim`新しいものを参照する `VolumeSnapshot`、ここで `spec.dataSource`（または `spec.dataSourceRef`）の名前は `VolumeSnapshot`名前。

   例

次の例では、次のものを参照するPVCを作成します。 `VolumeSnapshot`名前 `import-snap。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

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

スナップショットディレクトリは、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、デフォルトでは非表示になっています。 `ontap-nas`そして `ontap-nas-economy`ドライバー。有効にする `.snapshot`スナップショットからデータを直接回復するためのディレクトリ。

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
```

スナップショット コピーを復元すると、既存のボリューム構成が上書きされます。スナップショット 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。

スナップショットからのインプレースボリューム復元

Tridentは、スナップショットからボリュームを迅速に復元する機能を提供します。 `TridentActionSnapshotRestore` （TASR）CR。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、スナップショットの復元をサポートしています。 `ontap-san` 、 `ontap-san-economy` 、 `ontap-nas` 、 `ontap-nas-flexgroup` 、 `azure-netapp-files` 、 `gcp-cvs` 、 `google-cloud-netapp-volumes` 、 そして `solidfire-san`ドライバー。

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のCRを作成します `pvc1`ボリュームスナップショット `pvc1-snapshot`。

   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。この例ではスナップショットから復元します `pvc1`。

```
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、失敗した場合にTrident は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ボリュームが「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。

ボリュームスナップショットコントローラを展開する

Kubernetes ディストリビューションにスナップショット コントローラーと 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. スナップショット コントローラーを作成します。

```
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
```

必要に応じて開く `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`更新 `namespace`あなたの名前空間に。

関連リンク

- "ボリュームスナップショット"
- "ボリュームスナップショットクラス"

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 グループのスナップショットは、同じ時点で取得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。VolumeGroupSnapshot に必要な最小バージョンは Kubernetes 1.32 です。

ボリュームグループのスナップショットを作成する

ボリュームグループスナップショットは、 `ontap-san`ドライバーは iSCSI プロトコル専用であり、ファイバー チャネル (FCP) や NVMe/TCP ではまだサポートされていません。始める前に

- 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、[ボリュームスナップショットコントローラを展開する] 。

  > 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`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するには、機能ゲートを 'true' に設定します。
- 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。
- VolumeGroupSnapshot を作成できるようにするには、すべての PVC/ボリュームが同じ SVM 上にあることを確認します。

手順

- VolumeGroupSnapshot を作成する前に、VolumeGroupSnapshotClass を作成します。詳細については、"ボリュームグループスナップショットクラス" 。

```
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、PVCを作成します。 `pvc1-group-snap`データソースとラベルとして `consistentGroupSnapshot: groupA`。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
```

- 同じラベルのVolumeGroupSnapshotを作成する(`consistentGroupSnapshot: groupA`) を PVC で指定します。

  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

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復元できます。ボリューム グループ スナップショットを単位として復元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
```

スナップショット コピーを復元すると、既存のボリューム構成が上書きされます。スナップショット 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。

スナップショットからのインプレースボリューム復元

Tridentは、スナップショットからボリュームを迅速に復元する機能を提供します。`TridentActionSnapshotRestore` （TASR）CR。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、 "スナップショットからのインプレースボリューム復元"。

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:

- グループ内の個々のスナップショットではなく、VolumeGroupSnapshots 全体を削除できます。
- PersistentVolume のスナップショットが存在している間に PersistentVolume が削除された場合、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、 Trident はそのボリュームを「削除中」状態に移行します。
- 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を作成し、その後グループを削除する場合は、クローン時に分割操作が開始され、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。

ボリュームスナップショットコントローラを展開する

Kubernetes ディストリビューションにスナップショット コントローラーと 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. スナップショット コントローラーを作成します。

```
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
```

必要に応じて開く `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`更新 `namespace`あなたの名前空間に。

関連リンク

- "ボリュームグループスナップショットクラス"
- "ボリュームスナップショット"

Tridentの管理と監視

Tridentをアップグレード

Tridentをアップグレード

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の新機能を活用できるよう、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。
- Google CloudのCloud Volumes Serviceを"CVSサービスタイプ"バックエンド構成を更新して、`standardsw`または `zoneredundantstandardsw`Trident 23.01 からアップグレードする場合のサービス レベル。更新に失敗した `serviceLevel`バックエンドでボリュームが失敗する可能性があります。参照 "CVSサービスタイプのサンプル" 詳細については。
- アップグレードする際には、 `parameter.fsType`で `StorageClasses`Tridentによって使用されます。削除して再作成できます `StorageClasses`既存のボリュームを中断することなく。
  - これは、強制するための要件です "セキュリティコンテキスト"SAN ボリューム用。
  - sample inputディレクトリには、https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templなどの例が含まれています。[`storage-class-basic.yaml.templ` ^]とリンク:https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-bronze-default.yaml[`storage-class-bronze-default.yaml` ^]。
  - 詳細については、"既知の問題" 。

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`命名規則。「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`規則では、「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。アップグレード手順については、"以前のバージョン"アップグレード元のバージョンを、4 つのリリースの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. 23.07 から 24.06 への最初のアップグレード。

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上の問題の詳細" 。

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. 使用 `kubectl get pods -n trident`ポッドを検査します。
   - オペレーターポッドがない場合、Tridentは次のようにインストールされます。 `tridentctl` 。
   - オペレーター ポッドがある場合、 Trident は手動または Helm を使用してTridentオペレーター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 torc` Trident がHelm を使用してインストールされているかどうかを判断します。
   - 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ています。

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オペレーターを使用してアップグレードする"

> 確認することをお勧めします"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"オペレータにアップグレードする前に。

*

オペレーターによるアップグレード

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 プロセスを理解しておく必要があります。これには、ローリング アップデートを有効にするTridentコントローラー、コントローラー Pod とノード Pod、およびノード DaemonSet への変更が含まれます。

Tridentオペレーターのアップグレード処理

数ある中の一つ"Tridentオペレータを使用する利点"Trident をインストールおよびアップグレードすると、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および Kubernetes オブジェクトが自動的に処理

されます。このようにして、Tridentはダウンタイムなしでアップグレードをサポートできます。"ローリングアップデート"。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コントローラーのデプロイメントとノード DaemonSet を削除して再作成します。
- Tridentコントローラー ポッドとTridentノード ポッドを新しいバージョンに置き換えます。
  - 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、以下を参照してください。"Tridentアーキテクチャ"。

オペレーターのアップグレードワークフロー

Tridentオペレータを使用してアップグレードを開始すると、次のようになります。

1. トライデント**Trident**:
   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コントローラーのデプロイメントを削除します。
   d. バージョン *n+1* のTridentコントローラーのデプロイメントを作成します。
2. **Kubernetes** は *n+1* 用のTridentコントローラー ポッドを作成します。
3. トライデント**Trident**:
   a. *n* のTrident Node DaemonSet を削除します。オペレーターはノード ポッド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 1 つしか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、手動でも Helm を使用してもTrident をアップグレードできます。 Tridentオペレータのインストールから別のTridentオペレータのインストールにアップグレードしたり、 `tridentctl` Tridentオペレータ バージョンへのインストール。レビュー"アップグレード方法を選択してください"Tridentオペレータ インストールをアップグレードする前に。

手動インストールのアップグレード

クラスター スコープのTridentオペレーター インストールから別のクラスター スコープのTridentオペレーターインストールにアップグレードできます。すべてのTridentバージョンは、クラスター スコープの演算子を使用します。

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"Tridentの。

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、"バンドル_pre_1_25.yaml" 。
- Kubernetes 1.25以降を実行しているクラスターの場合は、"バンドルポスト1_25.yaml" 。

開始する前に

Kubernetesクラスタを実行していることを確認してください"サポートされているKubernetesバージョン"。

手順

1. Tridentのバージョンを確認してください:

```
./tridentctl -n trident version
```

2. 更新する `operator.yaml`、 `tridentorchestrator_cr.yaml` 、 そして `post_1_25_bundle.yaml`アップグレードするバージョン (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を入力します。
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カスタマイズした場合 `TridentOrchestrator`属性を編集できます `TridentOrchestrator`インストールパラメータを変更するオブジェクト。これには、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 レジストリを指定したり、デバッグ ログを有効にしたり、イメージ プル シークレットを指定したりするための変更が含まれる場合があります。
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. トライデント トルクを編集して、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は、values.yamlを更新して設定する必要があります。 `excludePodSecurityPolicy`に`true`または追加 `--set excludePodSecurityPolicy=true`に `helm upgrade`クラスターをアップグレードする前にコマンドを実行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. helm-mapkubeapisプラグインをインストールする https://github.com/helm/helm-mapkubeapis。
2. Tridentがインストールされている名前空間で、 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リストされます。

```
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. もしあなたが"Helmを使用してTridentをインストールしました"、使用することができます `helm upgrade trident netapp-trident/trident-operator --version 100.2506.0`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は、次の手順を実行します。
   a. 最新のTridentリリースをダウンロードするには、"GitHubの_Assets_セクション" 。
   b. 使用 `helm upgrade`コマンドの場所 `trident-operator-25.06.0.tgz`アップグレードするバージョンを反映します。

```
helm upgrade <name> trident-operator-25.06.0.tgz
```

> 初期インストール時にカスタムオプションを設定する場合（ TridentおよびCSIイメージのプライベートミラーレジストリを指定するなど）、 `helm upgrade`コマンド使用 `--set`これらのオプションがアップグレード コマンドに含まれていることを確認してください。含まれていない場合、値はデフォルトにリセットされます。

2. 走る `helm list`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。走る `tridentctl logs`デバッグ メッセージを確認します。

アップグレード `tridentctl`Tridentオペレーターへのインストール

Tridentオペレーターの最新リリースにアップグレードするには、 `tridentctl`インストール。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、"インストール方法の変更" 。

手順

1. 最新のTridentリリースをダウンロードしてください。

```
# Download the release required [25.06.0]
mkdir 25.06.0
cd 25.06.0
wget
https://github.com/NetApp/trident/releases/download/v25.06.0/trident-
installer-25.06.0.tar.gz
tar -xf trident-installer-25.06.0.tar.gz
cd trident-installer
```

2. 作成する `tridentorchestrator`マニフェストからの CRD。

```
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. 作成する `TridentOrchestrator`Trident をインストールするための CR。

```
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:                v25.06.0
```

tridentctl によるアップグレード

既存のTridentインストールを簡単にアップグレードできます。 `tridentctl` 。

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 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

レビュー"アップグレード方法を選択してください"アップグレードする前に `tridentctl`。

手順

1. アンインストールコマンドを実行する `tridentctl`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。参照"tridentctlを使用してTridentをインストールする"。

> アップグレード プロセスを中断しないでください。インストーラが完了するまで実行されたことを確認します。

tridentctl を使用してTrident を管理する

その "Tridentインストーラーバンドル"含まれるもの `tridentctl`Tridentへの簡単なアクセスを提供するコマンドライン ユーティリティ。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ポッドを含む名前空間を管理したりできます。

コマンドとグローバルフラグ

走れる `tridentctl help`利用可能なコマンドのリストを取得するには `tridentctl`または追加する `--help`任意のコマンドにフラグを付けると、その特定のコマンドのオプションとフラグのリストが取得されます。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl`このユーティリティは、次のコマンドとグローバル フラグをサポートしています。

コマンド

create
Tridentにリソースを追加します。

delete
Tridentから 1 つ以上のリソースを削除します。

get
Tridentから 1 つ以上のリソースを取得します。

help
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内の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内のリソースをアップグレードします。

version
Tridentのバージョンを出力します。

グローバルフラグ

-d、 --debug

デバッグ出力。

-h、 --help

ヘルプ `tridentctl`。

-k、 --kubeconfig string

指定する `KUBECONFIG`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するためのパス。

あるいは、 `KUBECONFIG`特定のKubernetesクラスタと問題を指す変数 `tridentctl`そのクラスターにコマンドを送信します。

-n、 --namespace string

Tridentデプロイメントの名前空間。

-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

使用 `create`Tridentにリソースを追加するコマンド。

```
tridentctl create [option]
```

オプション

`backend`: Tridentにバックエンドを追加します。

消去

使用 `delete`Tridentから 1 つ以上のリソースを削除するコマンド。

```
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 スナップショットを削除します。

`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。

得る

使用 `get`Tridentから 1 つ以上のリソースを取得するコマンド。

```
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

使用 `images`Tridentに必要なコンテナ イメージのテーブルを印刷するためのフラグ。

```
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。

輸入量

使用 `import volume`既存のボリュームをTridentにインポートするコマンド。

```
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` : 音量のヘルプ。
`--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

使用 `install`Trident をインストールするためのフラグ。

```
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 テレメトリを送信するためのプロキシのアドレス/ポート。
--enable-node-prep : 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 タイムアウトをオーバーライドします (デフォルトは 1 分 30 秒)。
--image-registry string : 内部イメージ レジストリのアドレス/ポート。
--k8s-timeout duration : すべての Kubernetes 操作のタイムアウト (デフォルトは 3 分 0 秒)。
--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 (テキスト、json) (デフォルトは「text」)。
--node-prep : Trident が指定されたデータ ストレージ プロトコルを使用してボリュームを管理するために Kubernetes クラスターのノードを準備できるようにします。 現在、 **iscsi**`サポートされている唯一の値です。 **OpenShift 4.19** 以降、この機能でサポートされる**Trident** の最小バージョンは **25.06.1** です。
`**--pv string** : Tridentによって使用されるレガシー 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によって使用されるレガシー 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 : 自動サポート 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 100、オプション)。
--use-custom-yaml : セットアップ ディレクトリに存在する既存の YAML ファイルを使用します。
--use-ipv6 : Trident の通信には IPv6 を使用します。

ログ

使用 `logs`Tridentからのログを印刷するためのフラグ。

```
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

使用 `send`Tridentからリソースを送信するコマンド。

```
tridentctl send [option]
```

オプション

`autosupport`: Autosupport アーカイブをNetAppに送信します。

uninstall

使用 `uninstall`Trident をアンインストールするためのフラグ。

```
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

使用 `update`Trident内のリソースを変更するコマンド。

```
tridentctl update [option]
```

オプション

`backend`: Tridentのバックエンドを更新します。

バックエンドの状態を更新する

使用 `update backend state`バックエンド操作を一時停止または再開するコマンド。

```
tridentctl update backend state <backend-name> [flag]
```

考慮すべき点

- バックエンドがTridentBackendConfig（tbc）を使用して作成された場合、バックエンドは `backend.json` ファイル。
- もし `userState`tbcに設定されている場合、 `tridentctl update backend state <backend-name> --user-state suspended/normal`指示。
- 設定する能力を取り戻すには `userState`tbc経由で設定した後、tridentctl経由で `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`指示。その後 `userState`フィールドが削除された場合は、 `tridentctl update backend state`変更するコマンド `userState`バックエンドの。
- 使用 `tridentctl update backend state`変更するには `userState`。更新することもできます `userState`使用して `TridentBackendConfig`または `backend.json`ファイル; これにより、バックエンドの完全な再初期化がトリガーされ、時間がかかる場合があります。

  フラグ

  `-h、 --help` : バックエンドの状態に関するヘルプ。
  `--user-state` : に設定 `suspended`バックエンド操作を一時停止します。設定 `normal`バックエンド操作を再開します。に設定すると `suspended`:

- `AddVolume`そして `Import Volume`一時停止されます。
- `CloneVolume`、 `ResizeVolume` 、 `PublishVolume` 、 `UnPublishVolume` 、 `CreateSnapshot` 、 `GetSnapshot` 、 `RestoreSnapshot` 、 `DeleteSnapshot` 、 `RemoveVolume` 、

`GetVolumeExternal`、`ReconcileNodeAccess`引き続きご利用いただけます。

バックエンドの状態を更新することもできます。`userState`バックエンド構成ファイルのフィールド`TridentBackendConfig`または`backend.json`。詳細については、"バックエンドを管理するためのオプション"そして"kubectl を使用してバックエンド管理を実行する"。

例：

JSON

以下の手順に従って更新してください `userState`使用して `backend.json`ファイル：

1. 編集する `backend.json`ファイルに `userState`値が「suspended」に設定されたフィールド。
2. バックエンドを更新するには、 `tridentctl update backend`コマンドと更新されたパス `backend.json`ファイル。

   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"
}
```

ヤムル

適用後にTBCを編集するには、 `kubectl edit <tbc-name> -n <namespace>`指示。次の例では、バックエンドの状態をsuspendに更新します。 `userState: suspended`オプション：

```
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

使用 `version`バージョンを印刷するためのフラグ `tridentctl`そして実行中のTridentサービス。

```
tridentctl version [flags]
```

フラグ

`--client`: クライアント バージョンのみ (サーバーは不要)。
`-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っており、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ある場合に、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を指定して検索範囲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。

モニターTrident

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 メトリック エンドポイントのセットを提供します。

概要

Tridentが提供するメトリックを使用すると、次のことが可能になります。

- Trident の状態と構成を監視します。操作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操作を実行するのにかかる時間を確認できます。

> デフォルトでは、Tridentのメトリクスはターゲットポートに公開されます。 `8001`で`/metrics`終点。これらのメトリックは、 Tridentがインストールされると デフォルトで有効 になります。

要件

- Tridentがインストールされた Kubernetes クラスター。
- Prometheus インスタンス。これは、 "コンテナ化されたPrometheusのデプロイメント"またはPrometheusを "ネイティブアプリケーション"。

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や作成するボリュームなどの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。これ "ブログ"Prometheus と Grafana をTridentと組み合わせて使用してメトリックを取得する方法について説明します。このブログでは、Kubernetes クラスタ

ーで Prometheus をオペレーターとして実行する方法と、 Tridentメトリックを取得するための ServiceMonitor を作成する方法について説明します。

ステップ2: Prometheus ServiceMonitorを作成する

Tridentのメトリクスを利用するには、監視するPrometheus ServiceMonitorを作成する必要があります。 `trident-csi`サービスと `metrics`ポート。サンプルの 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`サービスと特に `metrics`サービスのエンドポイント。その結果、Prometheus は Trident のメトリックを理解するように設定されるようになりました。

Tridentから直接入手できるメトリクスに加えて、kubeletは多くの `kubelet_volume_*`独自のメトリック エンドポイントを介してメトリックを取得します。 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。参照 "ここをクリックしてください。"。

ステップ3: PromQLでTridentメトリクスをクエリする

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の健康情報を入手する

- **Trident**からの **HTTP 2XX** 応答の割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- ステータス コード経由の**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の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テレメトリについて学ぶ

デフォルトでは、 Trident はPrometheus メトリックと基本的なバックエンド情報を毎日NetAppに送信します。

- TridentがPrometheusメトリックと基本的なバックエンド情報をNetAppに送信するのを止めるには、 `--silence-autosupport` Trident のインストール中にフラグを設定します。
- TridentはコンテナログをNetAppサポートにオンデマンドで送信することもできます。 `tridentctl send autosupport` 。ログをアップロードするには、 Tridentをトリガーする必要があります。ログを提出する前に、NetAppのhttps://www.netapp.com/company/legal/privacy-policy/["プライバシーポリシー"^]。

- 指定されない限り、 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。例えば： `tridentctl send autosupport --since=1h` 。この情報は、 `trident-autosupport` Tridentと一緒にインストールされるコンテナ。コンテナイメージは次の場所から入手できます。 "TridentAutoSupport" 。
- 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"これは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性への取り組みについて詳しくは、 "ここをクリックしてください。" 。

Tridentによって送信されるペイロードの例は次のようになります。

```
---
items:
  - backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
    protocol: file
    config:
      version: 1
      storageDriverName: ontap-nas
      debug: false
      debugTraceFlags: 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。
- インストール YAML ファイルを生成してプロキシ URL を構成することもできます。これは、 `tridentctl install --generate-custom-yaml` YAMLファイルを作成し、 `--proxy-url`の議論 `trident-autosupport`コンテナ内 `trident-deployment.yaml`。

Tridentメトリックを無効にする

メトリクスの報告を無効にするには、カスタムYAMLを生成する必要があります（ `--generate-custom-yaml`フラグ）を編集して削除します `--metrics`フラグが呼び出されないようにする `trident-main`容器。

Tridentをアンインストールする

Trident をアンインストールするには、 Tridentをインストールしたときと同じ方法を使用する必要があります。

タスク概要

- 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なアップグレードの修正が必要な場合は、 Tridentをアンインストールし、そのバージョンの特定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。"version" 。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。

- アップグレードと再インストールを容易にするために、 Tridentをアンインストールしても、 Tridentによって作成された CRD 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とCRDを完全に削除する" 。

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によって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 ノードで未公開になることが保証されます。

元のインストール方法を決定する

Trident をアンインストールするには、インストールに使用したのと同じ方法を使用する必要があります。アンインストールする前に、最初にTrident をインストールしたときに使用したバージョンを確認してください。

1. 使用 `kubectl get pods -n trident`ポッドを検査します。
   - オペレーターポッドがない場合、Tridentは次のようにインストールされます。 `tridentctl` 。
   - オペレーター ポッドがある場合、 Trident は手動または Helm を使用してTridentオペレーター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` Trident がHelm を使用してインストールされている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ています。

Tridentオペレータのインストールをアンインストールする

Trident 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

オペレータを使用してTrident をインストールした場合は、次のいずれかの方法でアンインストールできます。

1. 編集 **`TridentOrchestrator`CR** 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}}'
```

   いつ `uninstall`フラグが設定されている `true`、 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. 消去 **`TridentOrchestrator`**: 削除することにより `TridentOrchestrator`Trident を展開するために使用さ

れた CR の場合、オペレーターにTrident をアンインストールするように指示します。オペレーターは、`TridentOrchestrator`そして、 Tridentデプロイメントとデーモンセットを削除し、インストールの一環として作成された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、以下を使用してアンインストールできます。 `helm uninstall` 。

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

アンインストール `tridentctl`インストール

使用 `uninstall`コマンドイン `tridentctl`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>
```

Docker 用のTrident

展開の前提条件

Trident を展開する前に、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構成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する"要件"。
- サポートされているバージョンの Docker がインストールされていることを確認します。 Dockerのバージョンが古い場合は、 "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、構成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
```

確保する `etc/multipath.conf`含む `find_multipaths no`下 `defaults`。

5. 確実に `iscsid`そして `multipathd`実行中:

```
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
```

確保する `etc/multipath.conf`含む `find_multipaths no`下 `defaults`。

5. 確実に `open-iscsi`そして `multipath-tools`有効になっていて実行中である:

```
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でRHEL/Red Hat Enterprise Linux CoreOS (RHCOS)を実行するワーカーノードを使用する場合は、 `discard`インライン スペース再利用を実行するには、StorageClass の mountOption を使用します。参照 "Red Hat ドキュメント"。

RHEL 8以降

1. 次のシステム パッケージをインストールします。

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

2. マルチパスを有効にする:

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

確保する `etc/multipath.conf`含む `find_multipaths no`下 `defaults`。

3. 確実に `multipathd`実行中:

```
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
```

確保する `etc/multipath.conf`含む `find_multipaths no`下 `defaults`。

3. 確実に `multipath-tools`有効になっていて実行中:

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

Tridentを展開する

Trident for Docker は、 NetAppストレージ プラットフォームの Docker エコシステムと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。

Tridentの複数の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

参照"展開の前提条件"。前提条件が満たされていることを確認したら、 Trident をデプロイ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

従来のデーモン方式で Docker 1.13/17.03 より前のTrident を使用していた場合は、マネージド プラグイン方式を使用する前に、必ずTridentプロセスを停止し、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`ただし、 `config`ファイル名にオプションを指定します。設定ファイルは、 `/etc/netappdvp`ホスト システム上のディレクトリ。
   - `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。交換する `<version>`使用しているプラグインのバージョン (xxx.xx.x) に合わせてください。

```
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、 "配布の指示に従ってください" 。

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 Volume Plugin をインストールして構成します。
   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/v25.06.0/trident-
      installer-25.06.0.tar.gz
      tar zxf trident-installer-25.06.0.tar.gz
      ```

   b. ビンパス内の場所に移動します。

      ```
      sudo mv trident-installer/extras/bin/trident /usr/local/bin/
      sudo chown root:root /usr/local/bin/trident
      sudo chmod 755 /usr/local/bin/trident
      ```

   c. 構成ファイルの場所を作成します。

      ```
      sudo mkdir -p /etc/netappdvp
      ```

   d. 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. バイナリを配置して設定ファイルを作成したら、目的の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、次の場所にあります。 `contrib/trident.service.example` Git リポジトリ内。 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。

   ```
   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、 `systemctl daemon-reload`変更を認識させるためのコマンドです。

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、 Trident for Docker を安全にアップグレードできます。アップグレードプロセス中に、 `docker volume`プラグインに向けたコマンドは成功せず、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

Trident for Docker をアップグレードするには、以下の手順を実行します。

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 に代わるものです。直接アップグレードする必要があります `netapp/ndvp-plugin`イメージを `netapp/trident-plugin`画像。

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 より前) 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" 。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、追加の構成パラメータを渡して必要なTridentバージョンをインストールする必要があります。 `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

アンインストール

Trident for Docker をアンインストールするには、以下の手順を実行します。

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

標準のボリューム作成、クローン作成、削除を簡単に行うことができます。 `docker volume`必要に応じてTridentドライバー名を指定したコマンド。

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します。

```
docker volume create -d ntap_bronze --name bronzeVolume
```

何も指定しない場合は"options"、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
```

使用する際は `solidfire-san`ドライバーの場合、上記の例ではボリュームを削除して消去します。

Trident for Docker をアップグレードするには、以下の手順を実行します。

ボリュームをクローニングする

使用する際は `ontap-nas`、 `ontap-san` 、 `solidfire-san` 、 そして `gcp-cvs storage drivers`Trident はボリュームを複製できます。使用する際は `ontap-nas-flexgroup`または `ontap-nas-economy`ドライバーの場合、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。

- 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

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Tridentを使用するコンテナからアクセスできるのは、そのブロックデバイスにパーティションがなく、そのファイルシステムがTridentでサポートされている場合のみです（例： `ext4` -フォーマット済み `/dev/sdc1`Trident経由ではアクセスできなくなります)。

ドライバー固有のボリュームオプション

各ストレージ ドライバー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。オプションと値を指定するには、 `-o`

CLI 操作中のオペレータ。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	説明
`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)。
`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`つまり、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、すべてのONTAPシステムに「default」というポリシーが存在し、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、および 2 つの週次スナップショットが作成され、保持されます。スナップショットに保存されたデータは、 `.snapshot`ボリューム内の任意のディレクトリ内のディレクトリ。
`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。デフォルトでは値がありません。つまり、snapshotPolicy を選択した場合はONTAP はsnapshotReserve (通常 5%) を選択し、snapshotPolicy が none の場合は 0% を選択します。すべてのONTAPバックエンドの構成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。
`splitOnClone`	ボリュームのクローンを作成すると、 ONTAP はクローンをその親から直ちに分割します。デフォルトは `false`。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。

オプション	説明
`encryption`	新しいボリュームでNetAppボリューム暗号化（NVE）を有効にします。デフォルトは `false`。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。 バックエンドで NAE が有効になっている場合、Tridentでプロビジョニングされたすべてのボリュームで NAE が有効になります。 詳細については、以下を参照してください。"TridentがNVE および NAE と連携する仕組み" 。
`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まります。

次の追加オプションは NFS のみに有効です。

オプション	説明
`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、権限は次のように設定されます。 `---rwxr-xr-x 、または数値表記0755、および `root`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`	これを設定 `true`作るだろう `.snapshot`ボリュームにアクセスするクライアントに表示されるディレクトリ。デフォルト値は `false`つまり、 `.snapshot`ディレクトリはデフォルトで無効になっています。一部のイメージ、例えば公式のMySQLイメージは、 `.snapshot`ディレクトリが表示されます。
`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`。
`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`。有効な値は `unix`そして `mixed`。

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	説明
`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`。有効な値は `ext3`、 `ext4` 、 そして `xfs`。

オプション	説明
`spaceAllocation`	これを設定 `false`LUN のスペース割り当て機能をオフにします。デフォルト値は `true`つまり、ボリュームのスペースが不足し、ボリューム内の LUN が書き込みを受け付けることができなくなったときに、ONTAP はホストに通知します。また、このオプションで、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 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 20 MiB 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`Trident は0% のスナップショット予約を使用します。

- 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 ポリシーを使用して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 リザーブ (通常は 5%) を受け入れます。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Elementソフトウェアのボリュームオプション

Element ソフトウェア 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`命名法。

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	説明
`size`	ボリュームのサイズ。デフォルトは 1 GiB または構成エントリ…「defaults」: {"size": "5G"}。
`blocksize`	512 または 4096 を使用します。デフォルトは 512 または構成エントリ 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 のゴールド 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Tridentれる管理プラグイン方式（つまり、 `docker plugin`コマンド) については、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。デバッグログを有効にするには、以下を指定します。 `-debug`プラグインを実行するとき。

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`インストールされ実行されています。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`実行中です。 rpcbindサービスの状態を確認するには、 `systemctl status rpcbind`またはそれと同等のもの。

複数のTridentインスタンスを管理する

複数のストレージ構成を同時に利用したい場合は、 Tridentの複数のインスタンスが必要になります。複数のインスタンスを作成するための鍵は、 `--alias`コンテナ化されたプラグインのオプション、または `--volume-driver`ホスト上でTrident をインスタンス化するときのオプション。

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. エイリアスと構成ファイルを指定して最初のインスタンスを起動します。

   ```
   docker plugin install --grant-all-permissions --alias silver
   netapp/trident-plugin:21.07 config=silver.json
   ```

2. 別のエイリアスと構成ファイルを指定して、2 番目のインスタンスを起動します。

   ```
   docker plugin install --grant-all-permissions --alias gold
   netapp/trident-plugin:21.07 config=gold.json
   ```

3. 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. 各ドライバーインスタンスの 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構成で利用可能な構成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

これらの構成オプションは、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	説明	例
`version`	設定ファイルのバージョン番号	`1`
`storageDriverName`	ストレージドライバーの名前	`ontap-nas`、 `ontap-san` 、 `ontap-nas-economy` 、 `ontap-nas-flexgroup` 、 `solidfire-san`

オプション	説明	例
`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`

使用しないでください `storagePrefix`(デフォルトを含む) Element バックエンド用。デフォルトでは、 `solidfire-san`ドライバーはこの設定を無視し、プレフィックスを使用しません。NetApp、名前の変更が行われた可能性がある場合には、Docker ボリューム マッピングに特定の tenantID を使用するか、Docker のバージョン、ドライバー情報、および Docker からの生の名前が設定された属性データを使用することを推奨しています。

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。その `size`この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	説明	例
`size`	新しいボリュームのオプションのデフォルト サイズ。デフォルト：`1G`	`10G`

ONTAP構成

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を使用できます。

オプション	説明	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	説明	例
dataLIF	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*: NetAppは以下を指定することを推奨します dataLIF。指定されない場合、 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) を指定できるため、複数のデータ LIF 間で負荷分散を行うラウンドロビン 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選択的 LUN マップ" 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。警告が発生するのは、 `dataLIF`明示的に定義されています。	10.0.0.2
svm	使用するストレージ仮想マシン（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ー名	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)。のために `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。 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	説明	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。 のみ利用可能 **`ontap-nas`**そして **`ontap-nas-economy`**ドライバー。 "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はノードごとに作成および管理します `igroups`として `netappdvp。  この値は変更または省略できません。  のみ利用可能 **`ontap-san`**ドライバ。	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。 *については `ontap-nas-economy`ドライバーでは、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	*対応機種 `ontap-san`ドライバーのみ。*選択するには使用 `iscsi`iSCSIの場合、 `nvme` NVMe/TCPの場合または `fcp`SCSI over Fibre Channel (FC) 用。	`iscsi`空白の場合
limitVolumePoolSize	対応機種 **`ontap-san-economy`**そして **`ontap-san-economy`**ドライバーのみ。 ONTAP ontap-nas-economy および ontap-SAN-economy ドライバーのFlexVol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	説明	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（厚い）	none
snapshotPolicy	使用するスナップショットポリシー、デフォルトは none	none
snapshotReserve	スナップショットの予約率。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。デフォルトでは false	false
encryption	新しいボリュームでNetAppボリューム暗号化（NVE）を有効にします。デフォルトは false。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。 バックエンドで NAE が有効になっている場合、Tridentでプロビジョニングされたすべてのボリュームで NAE が有効になります。 詳細については、以下を参照してください。"TridentがNVE および NAE と連携する仕組み" 。	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	アクセスのためのNASオプション `.snapshot`ディレクトリ。	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	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サポート mixed`そして `unix`セキュリティスタイル。デフォルトは `unix。	unix
fileSystemType	SANオプションでファイルシステムの種類を選択します。デフォルトは ext4	xfs
tieringPolicy	使用する階層化ポリシー、デフォルトは none。	none

スケーリングオプション

その `ontap-nas`そして `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FlexVolをサポートし、クラスタの最大FlexVol数は12,000個です。Dockerボリュームの要件がこの制限内に収まる場合は、 `ontap-nas FlexVol が提供する Docker ボリュームのきめ細かなスナップショットやクローン作成などの追加機能により、このドライバーは推奨される NAS ソリューションです。

FlexVolの制限を超える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。

その `ontap-nas-economy`ドライバーは、自動的に管理されるFlexVolボリュームのプール内にONTAP Qtree として Docker ボリュームを作成します。 Qtree は、一部の機能を犠牲にして、クラスター ノードあたり最大 100,000、クラスターあたり最大 2,400,000 という、はるかに大きなスケーリングを提供します。その `ontap-nas-economy`ドライバーは、Docker ボリュームの粒度スナップショットまたはクローン作成をサポートしていません。

> その `ontap-nas-economy`Docker Swarm は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、このドライバーは現在 Docker Swarm ではサポートされていません。

その `ontap-san-economy`ドライバーは、自動的に管理されるFlexVolボリュームの共有プール内にONTAP LUN として Docker ボリュームを作成します。これにより、各FlexVol は1 つの LUN のみに制限されず、SAN ワークロードのスケーラビリティが向上します。ストレージ アレイに応じて、 ONTAP はクラスタごとに最大 16384 個の LUN をサポートします。ボリュームは下にある LUN であるため、このドライバーは Docker ボリューム単位のスナップショットとクローン作成をサポートします。

選択してください `ontap-nas-flexgroup`数十億のファイルでペタバイト単位にまで拡張可能な単一ボリュームへの並列処理を増やすドライバーです。 FlexGroups の理想的な使用例には、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ときに、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のFlexGroupサポートには、次の考慮事項もあります。

- ONTAPバージョン 9.2 以上が必要です。
- この記事の執筆時点では、FlexGroups は NFS v3 のみをサポートしています。
- SVM に対して 64 ビット NFSv3 識別子を有効にすることをお勧めします。
- 推奨されるFlexGroupメンバー/ボリュームの最小サイズは 100 GiB です。
- FlexGroupボリュームではクローン作成はサポートされていません。

FlexGroupとFlexGroupに適したワークロードの詳細については、 "NetApp FlexGroupボリュームのベストプラクティスと実装ガイド" 。

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。 `ontap-nas`そしてもう一つは `ontap-nas-economy。

Tridentのカスタム**ONTAP**ロール

最小限の権限を持つONTAPクラスタ ロールを作成すると、 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。

参照"Tridentカスタムロールジェネレーター"Tridentカスタム ロールの作成の詳細については、こちらをご覧ください。

ONTAP CLI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. クラスター レベルでカスタム ロールを作成するには、クラスター **>** 設定 を選択します。

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

   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. 役割*の下の+追加*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. 役割を**Trident**ユーザーにマップします: + ユーザーと役割 ページで次の手順を実行します。
   a. ユーザー*の下の追加アイコン+*を選択します。
   b. 必要なユーザー名を選択し、*役割*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"
- "役割とユーザーを操作する"

ONTAP構成ファイルの例

<code>ontap-nas</code> ドライバの NFS の例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

<code>ontap-nas-flexgroup</code> ドライバーの 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> ドライバーの 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> ドライバーの 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> ドライバーの 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> ドライバーの 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
}
```

要素ソフトウェア構成

Element ソフトウェア (NetApp HCI/ SolidFire) を使用する場合、グローバル設定値に加えて、次のオプションが使用できます。

オプション	説明	例
`Endpoint`	https://<ログイン>:<パスワード>@<mvip>/json-rpc/<要素バージョン>	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	説明	例
`TenantName`	使用する SolidFireF 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ースに制限する場合にインターフェースを指定する	`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プされた古いボリュームにアクセスするには、この値を設定する必要があります。	`netappdvp-`

その `solidfire-san`ドライバーは 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見つけます。

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. 追加の情報を提供してプラグインを再インストールします `config`パラメータ。

```
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ーの組み合わせをサポートできない原因とな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 クラスター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`そして `ontap-san`ドライバー機能は、これらの制限内で動作できる唯一のものです。

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にプロビジョニングされているFlexGroupと共通の 1 つ以上のアグリゲートが存在すると、 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 を展開するときは、ここに記載された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ポッドにアクセスすると、ユーザーはストレージ 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する

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。その "ストレージクォータメカニズム"管理者は、グローバルおよびストレージ クラス固有の容量とオブジェクト数の消費制限を名前空間ごとに実装できます。さらに、 "範囲制限"要求がプロビジョナーに転送される前に、PVC 要求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、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とCloud Volumes ONTAP のベストプラクティス

ONTAPおよびCloud Volumes ONTAP for Tridentを構成するためのベスト プラクティスについて説明します。

次の推奨事項は、 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用にONTAPを構成するためのガイドラインです。それぞれ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

ストレージ仮想マシン (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と適切な資格情報を提供し、SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、SVM 専用の管理インターフェイスを作成します。
- 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 に存在する必要があり、 Trident を構成するときに 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 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、最大限の柔軟性を実現する必要があります。 "SVM-DR" Tridentと併用してください。

最大音量数を制限する

ONTAPストレージ システムには最大ボリューム数があり、これはソフトウェア 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。参照 "NetApp Hardware Universe"正確な制限を確認するには、特定のプラットフォームとONTAPバージョンを参照してください。ボリューム数が使い果たされると、 Tridentだけでなく、すべてのストレージ要求のプロビジョニング操作が失敗します。

トライデントの `ontap-nas`そして `ontap-san`ドライバーは、作成された各 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。その `ontap-nas-economy`ドライバーは、約 200 PV ごとに 1 つの FlexVolume を作成します (50 ～ 300 の間で構成可能)。その `ontap-san-economy`ドライバーは、100 PV ごとに約 1 つの FlexVolume を作成します (50 ～ 200 の間で構成可能)。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これを実行できます:

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

の価値 `max-volumes`環境に固有のいくつかの基準によって異なります。

- ONTAPクラスタ内の既存ボリュームの数
- 他のアプリケーション用にTridentの外部にプロビジョニングする予定のボリュームの数
- Kubernetes アプリケーションによって消費されると予想される永続ボリュームの数

その `max-volumes`値は、個々のONTAPノードではなく、 ONTAPクラスタ内のすべてのノードにわたってプロビジョニングされたボリュームの合計です。その結果、 ONTAPクラスタ ノードに、他のノードよりもはる

かに多くの、または少ないTridentプロビジョニング ボリュームが存在する状況が発生する可能性があります。

たとえば、2 ノードのONTAPクラスタは、最大 2000 個のFlexVolボリュームをホストできます。最大ボリューム数を 1250 に設定するのは非常に合理的であると思われます。しかし、もし "アグリゲート"1 つのノードからのすべてのアグリゲートが 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がプロビジョニングできない場合 (たとえば、容量の問題など)、他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。これは、そのノードのボリューム制限に達する前に、`max-volumes`値に達すると、 Tridentと、そのノードを使用するその他のボリューム操作の両方に影響します。 この状況を回避するには、クラスター内の各ノードからのアグリゲートが、**Trident**が使用する **SVM** に均等に割り当てられていることを確認します。

ボリュームをクローニングする

NetApp Tridentは、 `ontap-nas` 、 `ontap-san` 、 `solidfire-san` 、 そして `gcp-cvs`ストレージ ドライバー。使用する際は `ontap-nas-flexgroup`または `ontap-nas-economy`ドライバーの場合、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 に関連付けられている 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ます。

Tridentによって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 `backend.json`意味。

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 **FlexVol** の最大サイズを制限する

ontap-san-economyおよびontap-nas-economyドライバのプールとして使用されるFlexVolの最大サイズを設定するには、 `limitVolumePoolSize`パラメータ `backend.json`意味。

双方向**CHAP**を使用するように**Trident**を設定する

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 上で CHAP を有効にすることができます。使用して `useCHAP`バックエンド構成のパラメータを指定すると、 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 に適用されたONTAP QoS ポリシーを活用すると、 Trident でプロビジョニングされたボリュームで消費可能な IOPS の数が制限されます。これは、 "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ないようにします。

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、ご使用のONTAPバージョン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のバージョンがサポートしている場合は、コンテナ化されたワークロードへのスループットの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、QoS ポリシーが低いと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。

ONTAPバージョンが 9.8 より前の場合にのみ、QoS ポリシー グループを SVM に割り当てる必要があります。

Trident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、以下を参照してください。 "QoSによるスループットの保証" 。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。

- 設定する必要があります `qosPolicy`キー入力 `defaults`バックエンド構成のブロック。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、以下を参照してください。 "ONTAPコマンド リファレンス" 。

Kubernetes クラスター 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、および 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の重要な要素です。そうすることで、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"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ので、 "ポッドセキュリティポリシー" 。

Kubernetes と外部ホストの両方からのアクセスが必要なボリュームの場合、PV は管理者によって導入され、Tridentによって管理されない従来の方法でストレージを管理する必要があります。これにより、Kubernetes と外部ホストの両方が切断され、ボリュームを使用し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 (OpenShift など) 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があるデプロイメントでは、ストレージ リソースへのアクセスをさらに制限するために、別のエクスポート ポリシーを使用する必要があります。これには、インフラストラクチャ ノードにデプロイされるサービス (OpenShift Metrics サービスや Logging サービスなど) と、非インフラストラクチャ ノードにデプロイ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。このようにして、 Trident はKubernetes クラスター内のノードにプロビジョニングする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。

- 使用 `vserver export-policy create`エクスポート ポリシーを作成するためのONTAP CLI コマンド。
- 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。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にする **`showmount`**アプリケーション**SVM**

その `showmount`この機能により、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るようになります。 Kubernetesクラスタにデプロイされたポッドは、 `showmount -e`に対してコマンドを実行し、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ります。これ自体はセキュリティ侵害にはなりませんが、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します。

無効にする必要があります `showmount`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のベストプラクティス

Trident用にSolidFireストレージを構成するためのベスト 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) 資格情報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、アカウント名と相対 CHAP 資格情報を使用するか、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

多数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は、 SolidFireのサービス品質 (QoS) ポリシーを使用します。

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。参照 "SolidFireサービス品質"QoS とパフォーマンスの詳細については、こちらをご覧ください。

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、よりシンプルでスケーリングの制限がないため、認証には CHAP を使用することを推奨しています。

拡張 CSI プロビジョナーを備えたTrident は、 CHAP 認証の使用をサポートします。 VAG は、従来の非 CSI 動作モードでのみ使用する必要があります。

CHAP 認証 (イニシエータ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。検索 "NetAppライブラリ"最新バージョンについては。

ONTAP

- "NFS 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 エクスプレス構成"

エレメントソフトウェア

- "Linux用SolidFireの設定"

NetApp HCI

- "NetApp HCI導入の前提条件"
- "NetApp導入エンジンにアクセスする"

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上の MySQL のベストプラクティス"
- "NetApp SolidFireと Cassandra"
- "SolidFireにおけるOracleのベストプラクティス"
- "SolidFireにおける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ので、 NetAppチームと協力し、"NetAppライブラリ"最新のドキュメントを見つけます。

Tridentを統合

Trident を統合するには、ドライバーの選択とデプロイメント、ストレージ クラスの設計、仮想プールの設計、ストレージ プロビジョニングに対する Persistent Volume Claim (PVC) の影響、ボリューム操作、およびTrident を使用した OpenShift サービスのデプロイメントといった設計およびアーキテクチャ要素の統合が必要です。

ドライバーの選択と展開

ストレージ システムのバックエンド ドライバーを選択して展開します。

ONTAPバックエンド ドライバー

ONTAPバックエンド ドライバーは、使用されるプロトコルと、ストレージ システム上でボリュームがプロビジョニングされる方法によって区別されます。したがって、どのドライバーを展開するかを決定する際に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 チームの快適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る各 PV は qtree であり、FlexVolume ごとに設定可能な数の qtree があります (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、独自の FlexVolume 内の LUN です。
- `ontap-san-economy`: プロビジョニングされる各 PV は LUN であり、FlexVolume ごとに構成可能な数の LUN (デフォルトは 100) があります。

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	はい脚注:5[]	はい	はい脚注:1[]	はい	はい脚注:1[]
`ontap-nas-economy`	脚注:3[]	脚注:3[]	はい脚注:5[]	はい	脚注:3[]	はい	脚注:3[]
`ontap-nas-flexgroup`	はい脚注:1[]	いいえ	はい脚注:5[]	はい	はい脚注:1[]	はい	はい脚注:1[]

Trident はONTAP用の 2 つの SAN ドライバーを提供しており、その機能を以下に示します。

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	はい脚注:4[]	はい	はい脚注:1[]	はい	はい脚注: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	はい脚注:4[]	はい	脚注:3[]	はい	脚注:3[]

> 上記の表の脚注: はい脚注:1[]: Tridentによって管理されません はい脚注:2[]: Tridentによって管理されますが、PV 細分化されません いいえ脚注:3[]: Tridentによって管理されず、PV 細分化されません はい脚注:4[]: 生のブロック ボリュームでサポートされます はい脚注:5[]: Tridentによってサポートされます

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 FlexVol 内の qtree または LUN) が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`そして `ontap-nas-economy`同じです。しかし、 `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します。これは特に災害復旧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。

> その `solidfire-san`ドライバーは PVC の複製も可能です。

Cloud Volumes ONTAPバックエンド ドライバー

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 ケースに対応するエンタープライズ クラスのストレージ機能とともにデータ制御を提供します。 Cloud Volume ONTAPと互換性のあるドライバーは ontap-nas、 ontap-nas-economy 、 ontap-san`そして `ontap-san-economy。これらは、Cloud Volume ONTAP for Azure、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ー

Amazon FSx for NetApp ONTAP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、スケーラビリティを活用できます。 FSx for ONTAP は、多くのONTAPファイル システム機能と管理 API をサポートしています。 Cloud Volume ONTAPと互換性のあるドライバーは ontap-nas、 ontap-nas-economy 、 ontap-nas-flexgroup 、 ontap-san`そして `ontap-san-economy。

NetApp HCI/ SolidFireバックエンド ドライバー

その `solidfire-san`NetApp HCI/ SolidFireプラットフォームで使用されるドライバーは、管理者が QoS 制限に基づいてTridentの Element バックエンドを構成するのに役立ちます。 Tridentによってプロビジョニングされ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、 `type`バックエンド ファイルのパラメータ。管理者は、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。 `limitVolumeSize`パラメータ。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。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ドライバー	Snapshot数	クローン	マルチアタッチ	チャップ	QoS	サイズ変更	レプリケーション
`solidfire-san`	はい	はい	はい脚注:2[]	はい	はい	はい	はい脚注:1[]

> 脚注: はい脚注:1[]: Tridentでは管理されません はい脚注:2[]: raw ブロックボリュームでサポートされます

Azure NetApp Filesバックエンド ドライバー

Tridentは `azure-netapp-files`ドライバーを管理する"Azure NetApp Files"サービス。

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" 。

Azure NetApp Filesドライバー	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	はい脚注:1[]

> 脚注: はい脚注:1[]:Tridentによって管理されていない

Google Cloud バックエンド ドライバー上のCloud Volumes Service

Tridentは `gcp-cvs`Google Cloud 上のCloud Volumes Serviceにリンクするためのドライバー。

その `gcp-cvs`ドライバーは仮想プールを使用してバックエンドを抽象化し、 Trident がボリュームの配置を決定できるようにします。管理者は仮想プールを `backend.json`ファイル。ストレージ クラスはセレクターを使用して、ラベルによって仮想プールを識別します。

- バックエンドで仮想プールが定義されている場合、 Trident はそれらの仮想プールが制限されている Google Cloud ストレージ プールにボリュームを作成しようとします。
- バックエンドで仮想プールが定義されていない場合、 Trident はリージョン内の利用可能なストレージ プールから Google Cloud ストレージ プールを選択します。

TridentでGoogle Cloudバックエンドを構成するには、以下を指定する必要があります。 `projectNumber` 、 `apiRegion` 、 そして `apiKey`バックエンドファイル内。プロジェクト番号は Google Cloud コンソールで確認できます。 API キーは、Google Cloud でCloud Volumes Serviceの API アクセスを設定するときに作成したサービス アカウントの秘密キー ファイルから取得されます。

Google CloudのCloud Volumes Serviceのサービスタイプとサービスレベルの詳細については、以下を参照してください。"GCP 向け CVS のTridentサポートについて学ぶ" 。

Google Cloud ドライバー用のCloud Volumes Service	Snapshot 数	クローン	マルチアタッチ	QoS	拡張	レプリケーション
gcp-cvs	はい	はい	はい	はい	はい	CVS-Performance サービスタイプでのみ利用可能です。

レプリケーションノート

- レプリケーションはTridentによって管理されません。
- クローン は、ソース ボリュームと同じストレージ プールに作成されます。

ストレージクラスの設計

Kubernetes ストレージ クラス オブジェクトを作成するには、個々のストレージ クラスを構成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`。

その `storagePools`パラメーターは、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。その `additionalStoragePools`パラメータは、属性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。 `storagePools`パラメータ。いずれかのパラメータを単独で、または両方を一緒に使用して、適切なストレージ プールのセットが選択されていることを確認できます。

その `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したい場合は、 `media`属性として `hdd`または `ssd`。に基づいて `media`ストレージクラスで指定された属性に基づいて、 Tridentは適切なバックエンドを選択します。 `hdd`または `ssd`アグリゲートをメディア属性と一致させてから、特定のアグリゲートに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、PREMIUMというストレージクラスを作成すると、 `media`属性設定 `ssd`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、STANDARD QoS ポリシーとして分類できる別のストレージ クラス STANDARD を作成できます。また、ストレージ クラスの「IOPS」属性を使用して、QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを活用する

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のプロビジョニングを直接行う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ーを使用して、任意のバックエンドの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 はすべてのセレクタ ラベルに一致するバックエンドを選択してボリュームを配置します。ストレージ クラス セレクターのラベルが複数のストレージ プールと一致する場合、 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際には、一般的に一連のパラメータを指定できます。管理者が同じストレージ認証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、Kubernetesストレージクラスを介してセレクタとし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を使用して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、GCPのCloud Volumes Service 、 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調べてみましょう。異なるパフォーマンス レベルを表す複数のラベルを使用して、 Azure NetApp Filesバックエンドを構成します。セット `servicelevel`側面を適切なパフォーマンス レベルに設定し、各ラベルの下にその他の必要な側面を追加します。次に、異なる仮想プールにマップする異なる Kubernetes ストレージ クラスを作成します。使用して `parameters.selector`フィールドでは、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。

特定の側面のセットを割り当てる

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、Kubernetesストレージクラスを作成します。 `parameters.selector`異なる仮想プールにマップされるフィールド。バックエンドでプロビジョニングされるボリュームには、選択した仮想プールで定義された側面が設定されます。

ストレージの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ーは、PVC の作成時にTridentプロビジョニングの決定プロセスに影響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、使用されるストレージ プロトコルと指定されたアクセス メソッドを一致させようとします。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。

	一度だけ読み書き可能	読み取り専用	読み取り書き込み多数
iSCSI	はい	はい	はい（生のブロック）
NFS	はい	はい	はい

NFS バックエンドが構成されていないTridentデプロイメントに 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、リクエスト側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。

オンデマンドのボリュームスナップショットを作成する

Trident は、 CSI フレームワークを使用したオンデマンドのボリューム 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ます。スナップショットは、データの特定時点のコピーを維持する便利な方法を提供し、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 を複製できます。

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 PersistentVolume 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`ボリュームを作成するために必要なスナップショットとして。 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。この機能を使用すると、リージョン間でデータを複製したり、テスト環境を作成したり、破損または壊れた本番ボリューム全体を交換したり、特定のファイルやディレクトリを取得して別の接続されたボリュームに転送したりすることが可能になります。

クラスター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 コンシューマーに支障をきたすことなく、 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移動できます。宛先アグリゲートがTridentが使用している SVM がアクセスできるものである限り、この操作はTridentまたは Kubernetes クラスターに影響を与えません。重要なのは、集計が SVM に新しく追加された場合、それをTridentに再度追加してバックエンドを更新する必要があることです。これにより、 Trident はSVM の再インベントリを実行し、新しいアグリゲートが認識されるようになります。

ただし、バックエンド間でのボリュームの移動は、 Tridentでは自動的にはサポートされません。これには、同じクラスター内の SVM 間、クラスター間、または異なるストレージ プラットフォーム (そのストレージ システムがTridentに接続されている場合も含む) が含まれます。

ボリュームを別の場所にコピーする場合は、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、および FC PV のサイズ変更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のサイズを直接変更できるようになります。ボリューム拡張は、 ONTAP、 SolidFire/ NetApp HCI 、 Cloud Volumes Serviceバックエンドを含むすべての主要なNetAppストレージ プラットフォームで可能です。後で拡張できるように設定するには `allowVolumeExpansion`に `true`ボリュームに関連付けられた StorageClass 内。永続ボリュームのサイズを変更する必要がある場合は、 `spec.resources.requests.storage`永続ボリューム要求の注釈を必要なボリューム サイズに設定します。 Trident は、ストレージ クラスター上のボリュームのサイズ変更を自動的に処理します。

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` 、 `azure-netapp-files` 、 そして `gcp-cvs`ドライバー。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、災害復旧シナリオ時に役立ちます。

ONTAPを使用する場合 `solidfire-san`ドライバーの場合は、コマンド `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`既存のボリュームを Kubernetes にインポートして、 Tridentで管理できるようにします。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、ボリュームの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複製します。

もし `azure-netapp-files`または `gcp-cvs`ドライバーを使用する場合は、コマンド `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`Tridentで管理できるようにボリュームを Kubernetes にインポートします。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 はバックエンド上の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)。次に、 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ー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、コンテナー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ー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展開と管理については、"ネットアップ"の中で"ブログ"。

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) によって提供される構成パラメーターを使用して Ansible を使用してデプロイされます。説明するインストール方法は 2 つあります。OpenShift の初期インストール時にログをデプロイする方法と、OpenShift のインストール後にログをデプロイする方法です。

> Red Hat OpenShift バージョン 3.9 の時点では、データ破損の懸念があるため、公式ドキュメントではログ 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 サーバーにはこれらの問題はなく、ログの展開を簡単にバックアップできます。最終的に、ログ サービスのプロトコルの選択はユーザー次第ですが、 NetAppプラットフォームを使用する場合はどちらも問題なく機能し、NFS が好みであれば 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変数を設定する必要があります。 `openshift_enable_unsupported_configurations`に `true`インストーラーが失敗するのを防ぐためです。

始めましょう

オプションで、ログ サービスは、アプリケーションだけでなく、OpenShift クラスター自体のコア操作にもデプロイできます。操作ログを展開することを選択した場合は、変数を指定して `openshift_logging_use_ops`として `true`、サービスのインスタンスが 2 つ作成されます。操作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 変数を構成することが重要です。それぞれの展開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 のログ記録ドキュメント"これらは、展開に応じて確認、構成、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くなりますが、既存のボリュームが利用できる場合はオプションとして使用できます。

変数	詳細
`openshift_logging_storage_kind`	設定 `nfs`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、ボリュームが次のようにジャンクションされている場合 `/openshift_logging`、この変数にはそのパスを使用します。

変数	詳細
`openshift_logging_storage_volume_name`	名前、例 `pv_ose_logs`、作成するPVの。
`openshift_logging_storage_volume_size`	NFSエクスポートのサイズ、例 `100Gi`。

OpenShift クラスターがすでに実行されており、 Trident がデプロイおよび構成されている場合は、インストーラー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`	ログ サービスによって使用される PVC のプレフィックス。
`openshift_logging_es_ops_pvc_dynamic`	設定 `true`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。
`openshift_logging_es_ops_pvc_size`	オペレーション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

ログ記録を OpenShift の初期インストール プロセスの一部としてデプロイする場合は、標準のデプロイ プロセスに従うだけで済みます。 Ansible は必要なサービスと OpenShift オブジェクトを構成およびデプロイし、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多少異なる可能性がありますので、必ず読んで従ってください。"Red Hat OpenShift Container Platform 3.11 ドキュメント"あなたのバージョン用。

メトリクスサービス

メトリクス サービスは、OpenShift クラスターのステータス、リソース使用率、可用性に関する貴重な情報を管理者に提供します。これはポッドの自動スケール機能にも必要であり、多くの組織はチャージバックやショーバック アプリケーションにメトリック サービスのデータを使用しています。

ログ サービスや OpenShift 全体と同様に、メトリック サービスのデプロイには Ansible が使用されます。また、ログ サービスと同様に、メトリック サービスは、クラスター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、メトリック 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、展開に応じて確認、構成、使用する必要があります。

変数	詳細
`openshift_metrics_storage_kind`	設定 `nfs`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、ボリュームが次のようにジャンクションされている場合 `/openshift_metrics`、この変数にはそのパスを使用します。
`openshift_metrics_storage_volume_name`	名前、例 `pv_ose_metrics`、作成するPVの。
`openshift_metrics_storage_volume_size`	NFSエクスポートのサイズ、例 `100Gi`。

OpenShift クラスターがすでに実行されており、 Trident がデプロイおよび構成されている場合は、インストーラー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。
`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デプロイする

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、使用されます。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、OpenShift のインストール後に Ansible が必要な PVC を作成し、 Trident がそれらのストレージをプロビジョニングした後、サービスをデプロイします。

上記の変数とデプロイのプロセスは、OpenShift のバージョンごとに変更される可能性があります。必ず確認して従ってください"Red HatのOpenShiftデプロイメントガイド"お使いの環境に合わせて構成されるように、バージョンに応じて変更してください。

データ保護とディザスタ リカバリ

TridentおよびTrident を使用して作成されたボリュームの保護および回復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、データ保護および回復戦略を用意する必要があります。

Tridentの複製と回復

災害発生時にTrident を復元するためのバックアップを作成できます。

Trident複製

Trident は、 Kubernetes CRD を使用して自身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クラスタetcdをバックアップするには"Kubernetes: etcd クラスターのバックアップ"。
2. バックアップアーティファクトをFlexVol volumeに配置する

> NetApp、 FlexVol が存在する SVM を別の SVM とのSnapMirror関係で保護することを推奨しています。

Tridentの回復

Kubernetes CRD と Kubernetes クラスター etcd スナップショットを使用して、 Trident を回復できます。

手順

1. 宛先 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コピーします。 `/etc/kubernetes/pki`そして、以下のetcdメンバーファイル `/var/lib/etcd。
3. etcdバックアップからKubernetesクラスターを復元するには、"Kubernetes: etcd クラスターの復元" 。
4. 走る `kubectl get crd`すべてのTridentカスタム リソースが起動していることを確認し、 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設定することはできませんが、ストレージ管理者は"ONTAPSnapMirror"SVM を複製します。

災害が発生した場合は、 SnapMirror の宛先 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場合は、次の点を考慮してください。

- 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

使用できます"ONTAP: SnapMirror SVM レプリケーション"SVM レプリケーション関係を作成します。

SnapMirror を使用すると、複製する内容を制御するオプションを設定できます。実行時に選択したオプションを知っておく必要がありますTridentを使用したSVM回復。

- "-アイデンティティ保存真"SVM 構成全体を複製します。
- "-discard-configs ネットワーク"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 偽"ボリュームとセキュリティ構成のみを複製します。

Tridentを使用したSVM回復

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、管理者は新しい SVM へのTridentフェイルオーバーを手動で開始できます。

手順

1. スケジュールされた進行中の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宛先 SVM をアクティブ化します。
2. 指定した場合 `-identity-preserve false`または `-discard-config network`SVMレプリケーションを設定するときは、 `managementLIF`そして `dataLIF`Tridentバックエンド定義ファイル内。
3. 確認する `storagePrefix`Tridentバックエンド定義ファイルに存在します。このパラメータは変更できません。省略 `storagePrefix`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、新しい宛先 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. 指定した場合 `-identity-preserve false`または `discard-config network`、すべてのアプリケーション ポッドをバウンスする必要があります。

> 指定した場合 `-identity-preserve true`宛先 SVM がアクティブ化されると、 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がデータの提供を開始します。

ボリュームの複製と回復

TridentはSnapMirrorのレプリケーション関係を設定することはできませんが、ストレージ管理者は"ONTAP SnapMirrorのレプリケーションとリカバリ"Tridentによって作成されたボリュームを複製します。

その後、回復したボリュームをTridentにインポートすることができます。"tridentctl ボリュームインポート"。

> インポートはサポートされていません `ontap-nas-economy`、 `ontap-san-economy` 、 または `ontap-flexgroup-economy`ドライバー。

スナップショットデータ保護

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 を回復したりします。

  "ONTAPスナップショット"

セキュリティ

セキュリティ

ここで挙げた推奨事項に従って、 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

信頼性の高いストレージを確保し、潜在的な悪意のあるアクティビティをブロックするには、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オブジェクト定義またはポッドにアクセスできないようにすることが重要です。

他のアプリケーションやユーザーを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ます。(`trident` ）。 Trident を独自の名前空間に配置すると、Kubernetes 管理担当者だけがTridentポッドと、名前空間の 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るようにし、 `tridentctl`応用。

ONTAP SANバックエンドで**CHAP**認証を使用する

TridentはONTAP SANワークロードのCHAPベースの認証をサポートします（ `ontap-san`そして `ontap-san-economy`ドライバー)。 NetApp、ホストとストレージ バックエンド間の認証に、 Tridentを使用した双方向 CHAP を使用することを推奨しています。

SANストレージドライバを使用するONTAPバックエンドの場合、 Tridentは双方向CHAPを設定し、CHAPのユーザー名とシークレットを管理できます。 `tridentctl` 。参照"ONTAP SAN ドライバーを使用してバックエンドを構成する準備をする"Trident がONTAPバックエンドで CHAP を設定する方法を理解します。

NetApp HCIおよび**SolidFire**バックエンドで**CHAP**認証を使用する

NetApp、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ています。 Trident は、テナントごとに 2 つの CHAP パスワードを含む秘密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、それを `tridentvolume`それぞれの PV の CR オブジェクト。 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 を**NVE** および **NAE** と併用する

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" 。

- バックエンドで NAE が有効になっている場合、 Tridentでプロビジョニングされたすべてのボリュームは NAE が有効になります。
  - NVE暗号化フラグを設定するには `""`NAE 対応ボリュームを作成します。
- NAEがバックエンドで有効になっていない場合、NVE暗号化フラグが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対応になります。 `false` (デフォルト値) をバックエンド構成で指定します。

NAE 対応バックエンド上のTridentで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。

- NVE暗号化フラグを設定するには `true`Tridentバックエンド構成で NAE 暗号化をオーバーライドし、ボリュームごとに特定の暗号化キーを使用します。
- NVE暗号化フラグを設定する `false`NAE 対応のバックエンドでは、`NAE` 対応のボリュームが作成されます。 `NVE`暗号化フラグを次のように設定して`NAE`暗号化を無効にすることはできません。 `false` 。

- TridentでNVEボリュームを手動で作成するには、NVE暗号化フラグを明示的に設定します。 `true` 。

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 統合キー設定 (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、aes-xts-plain64暗号とモードを使用します。これは、"NIST" 。

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、以下を参照してください。"ASA r2 ストレージシステムについて学ぶ" 。

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: 暗号化セットアップ" 。
- パフォーマンス上の理由から、 NetApp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ています。 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。"インテル: 高度暗号化標準命令 (AES-NI)" 。

LUKS暗号化を有効にする

ONTAP SAN およびONTAP SAN ECONOMY ボリュームでは、Linux Unified Key Setup (LUKS) を使用して、ボリュームごとのホスト側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" 。

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. 使用 `parameters.selector`LUKS 暗号化を使用してストレージ プールを定義します。例えば：

```
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。例えば：

```
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、 `luksEncryption`に(`true`バックエンドで。その`luksEncryption`オプションはボリュームがLUKS準拠かどうかをTridentに伝える(`true`）またはLUKSに準拠していない(`false`) を次の例のように入力します。

```
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、アノテーションを設定します `trident.netapp.io/luksEncryption`に `true`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することができます。

> 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ようにしてください。参照された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. 追加する `node-publish-secret-name`そして `node-publish-secret-namespace`StorageClass パラメータ。例えば：

```
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識別します。

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。確保する `previous-luke-passphrase-name`そして `previous-luks-passphrase`以前のパスフレーズと一致します。

```
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。これは回転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"]`、回転は不完全です。新しいポッドをトリガーして、回転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ボリューム拡張を有効にすることができます。

手順

1. 有効にする `CSINodeExpandSecret`機能ゲート (ベータ 1.25 以上)。参照 "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" 詳細については。
2. 追加する `node-expand-secret-name`そして `node-expand-secret-namespace`StorageClass パラメータ。例えば：

```
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 クラスターとストレージ 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の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 およびアップストリーム Kubernetes クラスターからオンプレミスONTAPボリュームへの NFSv3 および NFSv4 接続を介した 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。

オンプレミスの**ONTAP**ボリュームでインフライト **Kerberos** 暗号化を構成する

管理対象クラスターとオンプレミスの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 暗号化を有効にできます。

> オンプレミスの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。

開始する前に

- アクセスできることを確認してください `tridentctl`ユーティリティ。
- 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ます。
- 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 を準備していることを確認します。参照 "データLIFでKerberosを有効にする"手順についてはこちらをご覧ください。

- 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ドメイン構成セクション（13ページ）を参照してください。 "NetApp NFSv4 の機能強化とベストプラクティスガイド" 。

ONTAPエクスポートポリシーを追加または変更する

既存のONTAPエクスポート ポリシーにルールを追加するか、 ONTAPストレージ VM ルート ボリュームと、アップストリーム Kubernetes クラスターと共有されているすべてのONTAPボリュームに対して Kerberos 暗号化をサポートする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許可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構成できます。

- **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** - (アイデンティティとプライバシー保護を備えた認証と暗号化)

適切なアクセス権限を使用してONTAPエクスポート ポリシー ルールを設定します。たとえば、クラスターが 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 ルールを作成する方法については、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、次のオプションを使用して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`パラメータ:

- `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`（アイデンティティ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<>内の値は、環境の情報で置き換えます。

```
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、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、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`パラメータ:

- `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`（アイデンティティ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 クラスが作成されたことを確認します。

```
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
```

プロビジョニングボリューム

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" 。

Azure NetApp Filesボリュームでインフライト Kerberos 暗号化を構成する

マネージド クラスターと単一の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 トラフィックに対して Kerberos 暗号化を有効にできます。

開始する前に

- 管理対象 Red Hat OpenShift クラスターでTridentが有効になっていることを確認します。
- アクセスできることを確認してください `tridentctl`ユーティリティ。
- 要件に注意し、以下の手順に従って、Kerberos暗号化用のAzure NetApp Filesストレージバックエンドを準備したことを確認します。 "Azure NetApp Files のドキュメント" 。
- 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ドメイン構成セクション（13ページ）を参照してください。 "NetApp NFSv4 の機能強化とベストプラクティスガイド" 。

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。

- *ストレージバックエンドレベル*は、 `spec.kerberos`分野
- *仮想プールレベル*を使用して `spec.storage.kerberos`分野

仮想プール レベルで構成を定義する場合、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`（アイデンティティとプライバシー保護を備えた認証と暗号化）

手順

1. 管理対象クラスターで、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<>内の値は、環境の情報で置き換えます。

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、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 クラスが作成されたことを確認します。

```
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
```

プロビジョニングボリューム

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" 。

Trident Protectでアプリケーションを保護する

Tridentプロテクトについて学ぶ

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、その要件について知ることができます。

- "Tridentプロテクトの要件"

Trident Protectをインストールする

Tridentプロテクトの要件

まず、運用環境、アプリケーション クラスター、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 を展開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 クラスターの互換性

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 製品と互換性があります。

- Amazon Elastic Kubernetes サービス (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes サービス (AKS)
- レッドハット オープンシフト
- SUSE ランチャー
- 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ーに、実行中のスナップショット コントローラーと関連する CRD が設定されていることを確認します。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、 "これらの指示" 。

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。

- クラスターに接続されている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ます。
- 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 はこれらの名前空間を作成しません。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エコノミーボリュームの要件

Trident Protect は、NAS エコノミー ボリュームへのバックアップおよび復元操作をサポートします。スナップショット、クローン、および NAS エコノミー ボリュームへのSnapMirrorレプリケーションは現在サポートされていません。Trident Protect で使用する予定の各 nas-economy ボリュームに対し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>`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
```

Tridentバックエンド設定オプションを設定することで、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ことができます。 `snapshotDir`に `true`。既存のボリュームは影響を受けません。

KubeVirt VM によるデータ保護

Trident Protect 24.10 と 24.10.1 以降では、KubeVirt VM 上で実行されているアプリケーションを保護する場合の動作が異なります。どちらのバージョンでも、データ保護操作中にファイルシステムのフリーズとフリーズ解除を有効または無効にすることができます。

復元操作中は、 `VirtualMachineSnapshots`仮想マシン (VM) 用に作成されたファイルは復元されません。

Tridentプロテクト **24.10**

Trident Protect 24.10 は、データ保護操作中に KubeVirt VM ファイルシステムの一貫した状態を自動的に保証しません。Trident Protect 24.10 を使用して KubeVirt VM データを保護する場合は、データ保護操作の前に、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一貫した状態になることが保証されます。

Trident Protect 24.10を設定して、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することができます。"仮想化の構成"そして次のコマンドを使用します。

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

Trident Protect 24.10.1以降

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
```

SnapMirrorレプリケーションの要件

NetApp SnapMirrorレプリケーションは、次のONTAPソリューションのTrident Protect で使用できます。

- オンプレミスのNetApp FAS、 AFF、 ASAクラスター
- NetApp ONTAP Select
- NetAppCloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

SnapMirrorレプリケーションのONTAPクラスタ要件

SnapMirrorレプリケーションを使用する予定の場合は、 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使用するソース Kubernetes クラスターと宛先 Kubernetes クラスターの両方に存在する必要があります。Trident Protect は、次のドライバーによって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 (最低でもONTAPバージョン 9.15.1 が必要)
- ライセンス: データ保護バンドルを使用するONTAP SnapMirror非同期ライセンスは、ソースと宛先の両方のONTAPクラスタで有効にする必要があります。参照 "ONTAPにおけるSnapMirrorライセンスの概要"詳細についてはこちらをご覧ください。

  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。参照"ONTAP Oneに含まれるライセンス"詳細についてはこちらをご覧ください。

  SnapMirror非同期保護のみがサポートされます。

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 バックエンドをピアリングする必要があります。参照 "クラスタとSVMのピアリングの概要"詳細についてはこちらをご覧ください。

  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ます。

- * NetApp Tridentと SVM*: ピアリングされたリモート SVM は、宛先クラスタ上のNetApp Tridentで使用できる必要があります。
- 管理対象バックエンド: 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および管理する必要があります。

SnapMirrorレプリケーションのためのTrident / ONTAP構成

Trident Protect では、ソース クラスターと宛先クラスター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 つ構成する必要があります。ソース クラスターと宛先クラスターが同じ場合、復元力を最大限に高めるには、宛先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クラスタ要件

Kubernetes クラスターが次の要件を満たしていることを確認します。

- **AppVault** のアクセシビリティ: アプリケーション オブジェクトのレプリケーションでは、ソース クラスターと宛先クラスターの両方に、AppVault の読み取りと書き込みを行うためのネットワーク アクセスが必要です。
- 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 許可リストを構成して、WAN を介した両方のクラスターと AppVault 間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構成する前に、インフラストラクチャ チームとこれらのネットワーク要件を確認してください。

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ーに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することができます。クラスター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をインストールする

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。交換する `<name-of-cluster>`クラスター名。このクラスター名はクラスターに割り当てられ、クラスターのバックアップとスナップショット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 100.2506.0 --create
-namespace --namespace trident-protect
```

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 クラスター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。

```
netapp/controller:25.06.0
netapp/restic:25.06.0
netapp/kopia:25.06.0
netapp/trident-autosupport:25.06.0
netapp/exechook:25.06.0
netapp/resourcebackup:25.06.0
netapp/resourcerestore:25.06.0
netapp/resourcedelete:25.06.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

例えば：

```
docker pull netapp/controller:25.06.0
```

```
docker tag netapp/controller:25.06.0 <private-registry-
url>/controller:25.06.0
```

```
docker push <private-registry-url>/controller:25.06.0
```

2. Trident Protect システム名前空間を作成します。

```
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. という名前のファイルを作成します `protectValues.yaml`。次のTrident Protect 設定が含まれていることを確認します。

```
---
image:
  registry: <private-registry-url>
imagePullSecrets:
  - name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
    - name: regcred
webhooksCleanup:
  imagePullSecrets:
    - name: regcred
```

7. Helm を使用してTrident Protect をインストールします。交換する `<name_of_cluster>`クラスター名。このクラスター名はクラスターに割り当てられ、クラスターのバックアップとスナップショット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 100.2506.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

Trident Protect CLIプラグインをインストールする

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の拡張機能です。 `tridentctl` Trident Protect カスタム リソース (CR) を作成し、操作するためのユーティリティです。

Trident Protect CLIプラグインをインストールする

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 またはARM CPU を使用しているかどうかに応じて、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/25.06.0/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/25.06.0/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/25.06.0/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/25.06.0/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、次のように入力してプラグインを使用できるようになります。 `tridentctl-protect`または `tridentctl protect`どこからでも。

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後、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. 完了スクリプトをダウンロードします:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/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：

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. 次の行を `~/.bashrc`ホームディレクトリ内のファイル:

```
source ~/.bash/completions/tridentctl-completion.bash
```

Z シェルの自動補完を有効にする

手順

1. 完了スクリプトをダウンロードします:

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/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：

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. 次の行を `~/.zprofile`ホームディレクトリ内のファイル:

```
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 のデフォルト構成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 をインストールした後、構成ファイルを使用してTrident Protect コンテナー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ーのリソースの量を制御できます。

手順

1. という名前のファイルを作成します `resourceLimits.yaml`。
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。

   次の例の構成ファイルは、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. 値を適用する `resourceLimits.yaml`ファイル：

```
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 をインストールした後、構成ファイルを使用して、 Trident Protect コンテナーの 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 クラスター内のポッドのセキュリティ制限を定義します。

手順

1. という名前のファイルを作成します `sccconfig.yaml`。
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	説明	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リソースは次の場合にのみ作成されます。 `scc.create`設定されている `true`Helm インストール プロセスでは OpenShift 環境が識別されます。 OpenShift上で動作していない場合、または `scc.create`設定されている `false`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。	トライデントプロテクトジョブ
優先度	SCC の優先度を定義します。優先度の高い SCC は、優先度の低い SCC よりも先に評価されます。	1

3. の値を適用します `sccconfig.yaml`ファイル：

```
helm upgrade 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。

追加の**Trident Protect**ヘルムチャート設定を構成する

特定の要件に合わせて、 AutoSupport設定と名前空間フィルタリングをカスタマイズできます。次の表は、使用可能な構成パラメータを示しています。

パラメータ	タイプ	説明
自動サポートプロキシ	string	NetApp AutoSupport接続用のプロキシ URL を構成します。これを使用して、サポート バンドルのアップロードをプロキシ サーバー経由でルーティングします。例： `http://my.proxy.url` 。

パラメータ	タイプ	説明
autoSupport.insecure	ブーリアン	設定すると、 AutoSupportプロキシ接続のTLS検証をスキップします。 `true` 。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` ）
自動サポートが有効	ブーリアン	毎日のTrident Protect AutoSupportバンドルのアップロードを有効または無効にします。に設定すると `false`、スケジュールされた毎日のアップロードは無効になっていますが、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` ）
スキップ名前空間注釈の復元	string	バックアップおよび復元操作から除外する名前空間注釈のコンマ区切りリスト。注釈に基づいて名前空間をフィルタリングできます。
スキップ名前空間ラベルの復元	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 フラグを使用して構成できます。

YAMLファイルを使用する

手順

1. 設定ファイルを作成し、名前を付けます `values.yaml`。
2. 作成したファイルに、カスタマイズする構成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. 入力したら `values.yaml`正しい値を持つファイルの場合は、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. 次のコマンドを `--set`個々のパラメータを指定するためのフラグ:

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set restoreSkipNamespaceAnnotations="annotation1,annotation2" \
  --set restoreSkipNamespaceLabels="label1,label2" \
  --reuse-values
```

Trident Protectポッドを特定のノードに制限する

Kubernetes nodeSelector ノード選択制約を使用すると、ノード ラベルに基づいて、どのノードがTrident Protect ポッドを実行できるかを制御できます。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ています。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. という名前のファイルを作成します `nodeSelectorConfig.yaml`。
2. ファイルに nodeSelector オプションを追加し、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変更します。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. 値を適用する `nodeSelectorConfig.yaml`ファイル：

```
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。

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、ロールベースのアクセス制御 (RBAC) の Kubernetes モデルを使用します。デフォルトでは、 Trident Protect は単一のシステム名前空間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。組織内に多数のユーザーや特定のセキュリティ ニーズがある場合は、 Trident Protect の RBAC 機能を使用して、リソースや名前空間へのアクセスをより細かく制御できます。

クラスタ管理者は常にデフォルトのリソースにアクセスできます。 `trident-protect`名前空間だけでなく、他のすべての名前空間のリソースにもアクセスできます。リソースとアプリケーションへのアクセスを制御するには、追加の名前空間を作成し、それらの名前空間にリソースとアプリケーションを追加する必要があります。

デフォルトでは、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。`trident-protect`名前空間。アプリケーション名前空間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名前空間にアプリケーション データ管理 CR を作成します)。

特権のあるTrident Protect カスタム リソース オブジェクトには、管理者のみがアクセスできる必要があります。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
- **AutoSupportBundle**: メトリック、ログ、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" 。

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" 。

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、組織にはクラスター管理者、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。クラスター管理者は、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 グループの名前空間を作成します。

```
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

作成する新しい名前空間にはそれぞれデフォルトのサービス アカウントが付属しますが、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. マーケティング サービス アカウントのシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

ステップ**4: 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するための**RoleBinding**オブジェクトを作成する

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。RoleBinding オブジェクトを作成して適用することで、この ClusterRole をサービス アカウントにバインドできます。

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、クラスター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. ユーザーに AppVault へのアクセスを許可する AppVault とシークレットの組み合わせ YAML ファイルを作成して適用します。たとえば、次のCRはユーザーにAppVaultへのアクセスを許可します。 `eng-user` ：

```
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。例えば：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. RoleBinding CR を作成して適用し、権限をユーザー eng-user にバインドします。例えば：

```
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 オブジェクト情報を取得しようとします。

   ```
   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. ユーザーが現在アクセス権限を持っている 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、アプリケーション データ管理操作のために承認された AppVault オブジェクトを使用でき、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リソースを監視する

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 によって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 と併用すると、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する方法が提供されます。

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 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、Promethus、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。例えば：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. Prometheus ServiceMonitor CRD をクラスターに適用します。

```
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。例えば：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. kube-state-metricsを設定して、 Trident Protectが使用するカスタムリソースのメトリックを生成するには、以下の手順に従ってください。 "kube-state-metrics カスタムリソースのドキュメント" 。

Prometheusをインストールする

Prometheusは、以下の手順に従ってインストールできます。 "Prometheusのドキュメント" 。

Alertmanagerをインストールする

Alertmanagerは、以下の手順に従ってインストールできます。 "Alertmanager ドキュメント" 。

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設定ファイルを編集する(`prometheus.yaml`) をクリックし、kube-state-metrics サービス情報を追加します。例えば：

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設定ファイルを編集する(`prometheus.yaml`) に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構成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、アラートを送信する場所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に設定されたときにトリガーされる重要なアラートを定義します。 `Error` 5秒以上。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:

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は、電子メール、PagerDuty、Microsoft Teams、その他の通知サービスなどの他のチャネルに通知を送信するように設定できます。 `alertmanager.yaml`ファイル。

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`お使いの環境で使用されている Slack Webhook URL にキーを設定します。

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 を使用すると、管理者は、管理対象のクラスタとアプリケーションに関するログ、メトリック、トポロジ情報など、 NetAppサポートに役立つ情報を含むバンドルを生成できます。インターネットに接続している場合は、カスタム リソース (CR) ファイルを使用して、サポート バンドルをNetAppサポート サイト (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. 次の属性を構成します。
   - **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、スケジュールに従って生成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。有効な値は次のとおりです。
     - スケジュール済み
     - 手動
   - **spec.uploadEnabled**: (オプション) サポート バンドルを生成後にNetAppサポート サイトにアップロードするかどうかを制御します。指定しない場合は、デフォルトは `false`。有効な値は次のとおりです。
     - 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が開始する日時を指定する RFC 3339 形式の日付文字列。指定しない場合は、デフォルトで 24 時間前になります。指定できる最も早いウィンドウ日付は 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. 入力したら `trident-protect-support-bundle.yaml`正しい値を持つファイルには、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。その `trigger-type`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。デフォルト設定は Manual。

   例えば：

   ```
   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. 待つ `status.generationState`到達する `Completed`州。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
   ```

   見つける `kubectl cp`出力からコマンドを取得して実行し、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プロテクトのアップグレード

新しい機能やバグ修正を利用するには、 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。

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 をアップグレードします。

> 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 チャートに含まれるようになったため、この手順は必須です。

a. このコマンドを実行すると、CRDの管理を `trident-protect-crds`に `trident-protect`:

```
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Helmシークレットを削除します `trident-protect-crds`チャート：

> アンインストールしないでください `trident-protect-crds`Helm を使用してチャートを作成しないでください。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プロテクトのアップグレード:

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2506.0 --namespace trident-protect
```

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 と呼ばれます。AppVault オブジェクトは、ストレージ バケットの宣言型 Kubernetes ワークフロー表現です。AppVault CR には、バックアップ、スナップショット、復元操作、SnapMirrorレプリケーションなどの保護操作でバケットを使用するために必要な構成が含まれています。AppVault を作成できるのは管理者のみです。

アプリケーションに対してデータ保護操作を実行する際は、AppVault CR を手動で作成するか、コマンドラインから作成する必要があります。AppVaultCR は環境によって異なりますので、このページの例を参考にして AppVault CR を作成してください。

AppVault CR がTrident Protect がインストールされているクラスター上にあることを確認します。CRが存在しない場合、または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の認証とパスワードを設定する

AppVault CR を作成する前に、選択した AppVault とデータ ムーバーがプロバイダーおよび関連リソースに対して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 または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ときに、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。秘密を指定しない場合、 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`秘密を指定するための引数。

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ときに、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Trident Protect 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入力する必要があります。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にデータを復元する際に必要になります。クラスターまたは `trident-protect`名前空間が削除されると、パスワードなしでバックアップやスナップショットを復元できなくなります。

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

Amazon S3、Generic S3などのS3互換ストレージにアクセスする場合、"ストレージグリッドS3" 、 または "ONTAP S3"Trident Protect を使用する場合は、提供したユーザー認証情報にバケットにアクセスするために必要な権限があることを確認する必要があります。以下は、 Trident Protect によるアクセスに必要な最小限の権限を付与するポリシーの例です。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ーに適用できます。

```
{
  "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" 。

Amazon S3 (AWS) 認証用の EKS ポッド ID

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 クラスターで 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"。
- 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`

ポッドアイデンティティを有効にし、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 ポッドアイデンティティのドキュメント" 。

AppVault 構成 EKS Pod Identity を使用する場合は、AppVault CR を次のように構成します。 `useIAM: true` 明示的な資格情報の代わりにフラグを使用します:

```
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
```

クラウドプロバイダー向けの AppVault キー生成の例

AppVault CR を定義するときは、IAM 認証を使用していない限り、プロバイダー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資格情報を含める必要があります。資格情報のキーを生成する方法はプロバイダーによって異なります。以下は、いくつかのプロバイダーのコマンド ライン キー生成の例です。次の例を使用して、各クラウド プロバイダーの資格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
```

ストレージグリッド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

次の CR 例を使用して、各クラウド プロバイダーの 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。参照[データムーバーリポジトリのパスワード] 詳細についてはこちらをご覧ください。
- 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、プロバイダーのキーを生成するときに作成されます。クラウドプロバイダー向けの AppVault キー生成の例 。
- S3 AppVaultオブジェクトの場合、オプションで、アウトバウンドS3トラフィックの出力プロキシURLを次のように指定できます。 `spec.providerConfig.S3.proxyURL`鍵。

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を使用するEKS環境では、 `providerCredentials`セクションを追加 `useIAM: true`の下で `s3`代わりに構成してください。

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

```yaml
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
```

ストレージグリッド**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。参照[データムーバーリポジトリのパスワード] 詳細についてはこちらをご覧ください。
- S3 AppVaultオブジェクトの場合、オプションで、アウトバウンドS3トラフィックの出力プロキシURLを次のように指定できます。 `--proxy-url <ip_address:port>`口論。

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
```

ストレージグリッド**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
```

AppVault情報を表示する

Trident Protect CLI プラグインを使用して、クラスター上に作成した 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. オプションとして、各リソースのAppVaultPathを表示するには、フラグを使用します。 `--show-paths` 。

   表の最初の列にあるクラスター名は、 Trident Protect Helm インストールでクラスター名が指定された場合にのみ使用できます。例えば： `--set clusterName=production1` 。

AppVaultを削除する

AppVault オブジェクトはいつでも削除できます。

> 取り外さないでください `finalizers`AppVault オブジェクトを削除する前に、AppVault CR にキーを追加します。これを実行すると、AppVault バケットにデータが残り、クラスター内に孤立したリソースが存在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およびバックアップ CR が削除されていることを確認します。

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、 `appvault-name`削除する AppVault オブジェクトの名前:

```
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、 `appvault-name`削除する AppVault オブジェクトの名前:

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

Trident Protectで管理するアプリケーションを定義する

アプリケーション CR と関連する AppVault CR を作成することで、 Trident Protect で管理するアプリケーションを定義できます。

AppVault CRを作成する

アプリケーションでデータ保護操作を実行するときに使用する AppVault CR を作成する必要があります。また、AppVault CR は、 Trident Protect がインストールされているクラスター上に存在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、以下を参照してください。"AppVault カスタム リソース。"

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。アプリケーション CR を手動で作成するか、 Trident Protect CLI を使用して、管理対象のアプリケーションを定義できます。

CRを使用してアプリケーションを追加する

手順

1. 宛先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. 次の属性を構成します。
      - **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) 名前空間とラベル セレクターを使用して、アプリケーションが使用する名前空間とリソースを指定します。アプリケーション名前空間はこのリストの一部である必要があります。ラベル セレクターはオプションであり、指定された各名前空間内のリソースをフィルター処理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、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 KubeVirt 環境などの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。このアプリケーションが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、注釈は無視されます。指定されていない場合は、アプリケーションは"グローバルTrident Protectフリーズ設定"。

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) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

> 両方が `resourceFilter`そして `labelSelector`使用される、 `resourceFilter`最初に実行し、その後 `labelSelector`結果のリソースに適用されます。

例えば：

```
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。例えば：

```
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、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 KubeVirt 環境などの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`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。設定すると `false`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、アプリケーションのアプリケーション定義に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フリーズ設定"。

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。

- 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">
```

使用できます `--resource-filter-include`そして `--resource-filter-exclude`リソースを含めるか除外するかのフラグ `resourceSelectionCriteria`次の例に示すように、グループ、種類、バージョン、ラベル、名前、名前空間などです。

```
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する

自動保護ポリシーを使用するかアドホック ベースでスナップショットやバックアップを取得することにより、 Trident Protect によって管理されるすべてのアプリを保護できます。

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ようにTrident Protectを構成できます。"Trident Protect によるファイルシステムのフリーズ設定の詳細"。

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。

> クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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、名前を付けます `trident-protect-snapshot-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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 に何が起こるかを定義します。つまり、 `Retain`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. 入力したら `trident-protect-snapshot-cr.yaml`正しい値を持つファイルには、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。例えば：

```
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。

> クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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 バックアップ操作に十分であることを確認します。バックアップ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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-backup-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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) バックアップが要求から解放されたときに何が起こるかを定義します。有効な値は次のとおりです。
     - `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、一時的なスナップショットが作成され、バックアップされます。
   - **metadata.annotations.protect.trident.netapp.io/full-backup**: (オプション) この注釈は、バックアップを非増分にするかどうかを指定するために使用されます。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。ただし、この注釈が `true`、バックアップは非増分になります。指定しない場合は、バックアップはデフォルトの増分バックアップ設定に従います。復元に伴うリスクを最小限に抑えるために、定期的に完全バックアップを実行し、完全バックアップの間に増分バックアップを実行するのがベストプラクティスです。

     YAMLの例:

```yaml
---
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
  annotations:
    protect.trident.netapp.io/full-backup: "true"
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. 入力したら `trident-protect-backup-cr.yaml`正しい値を持つファイルには、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。例えば：

```
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`バックアップを非増分にするかどうかを指定するフラグ。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。復元に伴うリスクを最小限に抑えるために、定期的に完全バックアップを実行し、完全バックアップの間に増分バックアップを実行するのがベストプラクティスです。

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することによっ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、増分以外の完全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。定期的に完全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すると、復元に関連するリスクを軽減できます。

- スナップショットのスケジュールを作成するには、以下を設定します。 `backupRetention`ゼロにし、 `snapshotRetention`ゼロより大きい値に設定します。設定 `snapshotRetention`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スナップショットが作成されますが、それらは一時的なものであり、バックアップが完了するとすぐに削除されます。
- クラスター スコープのリソースは、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、スナップショット、またはクローンの中に含められます。

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、名前を付けます `trident-protect-schedule-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: 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.snapshotRetention**: 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.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`（指定する必要があります `spec.minute`）
     - `Daily`（指定する必要があります `spec.minute`そして `spec.hour`）
     - `Weekly`（指定する必要があります `spec.minute, spec.hour`、 そして `spec.dayOfWeek`）
     - `Monthly`（指定する必要があります `spec.minute, spec.hour`、 そして `spec.dayOfMonth`）
     - `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「」に設定されている場合、このフィールドは必須です。 `Monthly` 。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「」に設定されている場合、このフィールドは必須です。 `Weekly` 。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「」に設定されている場合、このフィールドは必須です。 `Daily` 、 `Weekly` 、 または `Monthly`。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分 (0 - 59)。粒度が「」に設定されている場合、このフィールドは必須です。 `Hourly` 、 `Daily` 、 `Weekly` 、 または `Monthly`。値は文字列として提供する必要があります。
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (オプション) このアノテーションは、完全バックアップをスケジュールするためのルールを指定するために使用されます。設定できるのは `always`継続的な完全バックアップを実現するか、要件に応じてカスタマイズしま

す。たとえば、毎日の粒度を選択した場合は、完全バックアップを実行する曜日を指定できます。

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
  annotations:
    protect.trident.netapp.io/full-backup-rule: "Monday,Thursday"
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. 入力したら `trident-protect-schedule-cr.yaml`正しい値を持つファイルには、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。例えば：

   使用できます `tridentctl-protect create schedule --help`このコマンドの詳細なヘルプ情報を表示します。

```
tridentctl-protect create schedule <my_schedule_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> --backup
-retention <how_many_backups_to_retain> --data-mover
<Kopia_or_Restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain> -n <application_namespace>
--full-backup-rule <string>
```

   設定できるのは `--full-backup-rule`フラグを `always`継続的な完全バックアップを実現するか、要件に応じてカスタマイズします。たとえば、毎日の粒度を選択した場合は、完全バックアップを実行する曜日を指定できます。例えば、 `--full-backup-rule "Monday,Thursday"`月曜日と木曜日に完全バックアップをスケジュールします。

   スナップショットのみのスケジュールの場合は、 `--backup-retention 0` 0より大きい値を指定する `--snapshot-retention`。

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された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されたバックアップまたはオンデマンドのバックアップを削除します。

回収ポリシーが設定されていることを確認する `Delete`オブジェクト ストレージからすべてのバックアップ データを削除します。ポリシーのデフォルト設定は `Retain`偶発的なデータ損失を回避するため。政策が変更されない場合 `Delete`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復元を有効にする

Trident Protect をインストールしている場合は、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、スペース効率の高いバックアップと復元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。
- あなたが持っている `azure-netapp-files`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 にアップグレードする前に作成された場合は、 Tridentで次の操作を実行します。

   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は、通常のバックアップ機能が選択され、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、バックアップ対象の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十分なスペースが容量プールに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 を使用してバックアップおよび復元する準備ができています。各 PVC は、バックアップや復元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復元する

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。
- qtree ボリュームでは、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がサポートされています。ただし、qtree ボリュームでは、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復元設定を使用する"。

バックアップから別の名前空間に復元する

BackupRestore CR を使用してバックアップを別の名前空間に復元すると、 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復元しても、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復元するには、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復元します。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンド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、名前を付けます `trident-protect-backup-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     ```
     ---
     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、いくつかのリソースを自動的に選択します。たとえば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、 Trident Protect は関連付けられたポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**: (フィルタリングに必須) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-backup-restore-cr.yaml`正しい値を持つファイルには、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。その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。例えば：

```
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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンド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、名前を付けます `trident-protect-backup-ipr-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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。

     例えば：

```
---
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、いくつかのリソースを自動的に選択します。たとえば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、 Trident Protect は関連付けられたポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**: (フィルタリングに必須) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-backup-ipr-cr.yaml`正しい値を持つファイルには、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。その `backup` 引数は、名前空間とバックアップ名を次の形式で使用します。 `<namespace>/<name>` 。例えば：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

元のクラスターに問題がある場合は、バックアップを別のクラスターに復元できます。

> 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。参照 "Kopiaのドキュメント"設定できるオプションの詳細については、こちらをご覧ください。使用 `tridentctl-protect create --help`Trident Protect CLI で注釈を指定する方法の詳細については、コマンドを参照してください。

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。

- 宛先クラスターにはTrident Protect がインストールされています。
- 宛先クラスターは、バックアップが保存されているソース クラスターと同じ AppVault のバケット パスにアクセスできます。
- 実行時に、ローカル環境が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。 `tridentctl-protect get appvaultcontent`指示。ネットワーク制限によりアクセスできない場合は、代わりに宛先クラスターのポッド内からTrident Protect CLI を実行します。
- 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。
  - 参照 "AWS API ドキュメント"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、こちらをご覧ください。
  - 参照 "AWSのドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

手順

1. Trident Protect CLI プラグインを使用して、宛先クラスター上の 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復元を目的とした名前空間が宛先クラスターに存在することを確認します。

2. 宛先クラスターから利用可能な 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 内の使用可能なバックアップ（元のクラスター、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 パスなど）が表示されます。

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 名とアーカイブ パスを使用して、アプリケーションを宛先クラスターに復元します。

CRを使用する

1. カスタムリソース（CR）ファイルを作成し、名前を付けます `trident-protect-backup-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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 が使用できない場合は、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**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     例えば：

```
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. 入力したら `trident-protect-backup-restore-cr.yaml`正しい値を持つファイルには、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。例えば：

```
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名前空間に復元する

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。

> SnapshotRestoreは、 `spec.storageClassMapping`属性ですが、ソース ストレージ クラスと宛先ストレージ 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。復元しようとすると、 `StorageClass`異なるストレージ バックエンドを使用する場合、復元操作は失敗します。

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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-snapshot-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.namespaceMapping**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     ```
     ---
     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、いくつかのリソースを自動的に選択します。たとえば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、 Trident Protect は関連付けられたポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**: (フィルタリングに必須) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-restore-cr.yaml`正しい値を持つファイルには、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名前空間に復元します。
   - 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>` 。
   - その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。

例えば：

```
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復元できます。

> 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。

- 参照 "AWS API ドキュメント"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、こちらをご覧ください。
- 参照 "AWS IAM ドキュメント"AWS リソースの認証情報の詳細については、こちらをご覧ください。

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、名前を付けます `trident-protect-snapshot-ipr-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 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、いくつかのリソースを自動的に選択します。たとえば、永続ボリューム要求リソースを選択し、それに関連付けられたポッドがある場合、 Trident Protect は関連付けられたポッドも復元します。

   - **resourceFilter.resourceSelectionCriteria**: (フィルタリングに必須) 使用 `Include`または `Exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-ipr-cr.yaml`正しい値を持つファイルには、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。例えば：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <snapshot_to_restore> \
-n <application_namespace>
```

復元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、または失敗した復元操作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**復元設定を使用する

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。

復元およびフェイルオーバー操作中の名前空間の注釈とラベル

復元およびフェイルオーバー操作中に、宛先名前空間のラベルと注釈は、ソース名前空間のラベルと注釈と一致するように作成されます。宛先名前空間に存在しないソース名前空間のラベルまたは注釈が追加され、既存のラベルまたは注釈はソース名前空間の値と一致するように上書きされます。宛先名前空間にのみ存在するラベルまたは注釈は変更されません。

> 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、復元されたポッドが OpenShift のセキュリティ コンテキスト制約 (SCC) によって定義された適切な権限とセキュリティ構成に準拠し、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 コンテキスト制約のドキュメント" 。

Kubernetes環境変数を設定することで、宛先名前空間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。 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`復元またはフェイルオーバー操作を実行する前に。例えば：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_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` そして `restoreSkipNamespaceLabels` 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、 "AutoSupportと名前空間フィルタリングオプションを構成する"。

ソースアプリケーションをHelmを使用してインストールした場合、 `--create-namespace`旗には特別な扱いが与えられます `name`ラベルキー。復元またはフェイルオーバー プロセス中に、 Trident Protect はこのラベルを宛先名前空間にコピーしますが、ソースからの値がソース名前空間と一致する場合は、値を宛先名前空間の値に更新します。この値がソース名前空間と一致しない場合は、変更されずに宛先名前空間にコピーされます。

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。

復元またはフェイルオーバー操作の前に

次の表は、復元またはフェイルオーバー操作前のサンプルのソース名前空間と宛先名前空間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
名前空間 ns-1 (ソース)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true"	• 環境=本番環境 • コンプライアンス=HIPAA • 名前=ns-1
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "true" • 注釈.three/キー: "false"	• 役割=データベース

復元操作後

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
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true" • 注釈.three/キー: "false"	• 名前=ns-2 • コンプライアンス=HIPAA • 環境=本番環境 • 役割=データベース

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

その `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 クラスター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。これは、異なるストレージ クラスを持つクラスター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注釈

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。ユーザーが注釈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	説明	デフォルト値
protect.trident.netapp.io/データムーバータイムアウト秒	string	データ ムーバー操作を停止できる最大時間 (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」

NetApp SnapMirrorとTrident Protectを使用してアプリケーションを複製する

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、データとアプリケーションの変更を、同じクラスター上または異なるクラスター間で、あるストレージ バックエンドから別のストレージ バックエンドに複製できます。

復元およびフェイルオーバー操作中の名前空間の注釈とラベル

復元およびフェイルオーバー操作中に、宛先名前空間のラベルと注釈は、ソース名前空間のラベルと注釈と一致するように作成されます。宛先名前空間に存在しないソース名前空間のラベルまたは注釈が追加され、既存のラベルまたは注釈はソース名前空間の値と一致するように上書きされます。宛先名前空間にのみ存在するラベルまたは注釈は変更されません。

> 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、復元されたポッドが OpenShift のセキュリティ コンテキスト制約 (SCC) によって定義された適切な権限とセキュリティ構成に準拠し、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 コンテキスト制約のドキュメント" 。

Kubernetes環境変数を設定することで、宛先名前空間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。 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`復元またはフェイルオーバー操作を実行する前に。例えば：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_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` そして `restoreSkipNamespaceLabels` 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、 "AutoSupportと名前空間フィルタリングオプションを構成する"。

ソースアプリケーションをHelmを使用してインストールした場合、 `--create-namespace`旗には特別な扱いが与えられます `name`ラベルキー。復元またはフェイルオーバー プロセス中に、 Trident Protect はこのラベルを宛先名前空間にコピーしますが、ソースからの値がソース名前空間と一致する場合は、値を宛先名前空間の値に更新します。この値がソース名前空間と一致しない場合は、変更されずに宛先名前空間にコピーされます。

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。

復元またはフェイルオーバー操作の前に

次の表は、復元またはフェイルオーバー操作前のサンプルのソース名前空間と宛先名前空間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
名前空間 ns-1 (ソース)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true"	• 環境=本番環境 • コンプライアンス=HIPAA • 名前=ns-1
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "true" • 注釈.three/キー: "false"	• 役割=データベース

復元操作後

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
名前空間 ns-2 (宛先)	• アノテーション.one/キー: "更新された値" • アノテーション.2/キー: "true" • 注釈.three/キー: "false"	• 名前=ns-2 • コンプライアンス=HIPAA • 環境=本番環境 • 役割=データベース

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ようにTrident Protectを構成できます。"Trident Protect によるファイルシステムのフリーズ設定の詳細"。

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注意する必要がある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 クラスターから宛先クラスター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宛先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、以下を参照してください。"Trident Protect実行フックを管理する" 。

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 が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) を作成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撮影する時間を設定する

手順

1. ソース クラスターで、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、以下の例を変更します。"AppVault カスタムリソース"あなたの環境に合わせて:

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. 次の属性を構成します。

- **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構成を保存します。プロバイダーの bucketName とその他の必要な詳細を選択します。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。参照"AppVault カスタムリソース"他のプロバイダーとの AppVault CR の例。
- **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資格情報への参照を格納します。
  - **spec.providerCredentials.valueFromSecret**: (必須) 資格情報の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。たとえば、 NetApp ONTAP S3、汎用 S3、Google Cloud、Microsoft Azure などです。有効な値は次のとおりです。
  - AWS
  - 紺碧
  - gcp
  - ジェネリックS3
  - オンタップS3
  - ストレージグリッドS3

c. 入力したら `trident-protect-appvault-primary-source.yaml`正しい値を持つファイルには、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. 括弧内の値を環境の情報に置き換えて、AppVault を作成します。

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. ソース クラスターで、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. 次の属性を構成します。

- **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) 名前空間と関連ラベルの配列。ここにリストされている名前空間内に存在するリソースを指定するには、名前空間名を使用し、オプションでラベルを使用して名前空間の範囲を絞り込みます。アプリケーション名前空間はこの配列の一部である必要があります。

  YAMLの例:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. 入力したら `trident-protect-app-source.yaml`正しい値を持つファイルには、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。例えば：

```
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、宛先クラスター上のアプリケーションのベースとして使用されます。この手順を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"。

4. ソース クラスターで、レプリケーション スケジュール CR を作成します。

以下に示すスケジュールに加えて、ピア接続された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、保持期間が 7 日間の個別の毎日のスナップショット 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、保持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。この方法では、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. 次の属性を構成します。

   - **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**: に設定する必要があります `Custom`。
   - **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. 入力したら `trident-protect-schedule.yaml`正しい値を持つファイルには、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. 宛先クラスタで、ソースクラスタに適用したAppVault CRと同一の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. 宛先クラスター上の宛先アプリケーション用の宛先 AppVault CR を作成します。ストレージプロバイダに応じて、以下の例を変更します。"AppVault カスタムリソース"あなたの環境に合わせて:
   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. 次の属性を構成します。
      - **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構成を保存します。選択してください `bucketName`およびプロバイダーに関するその他の必要な詳細情報。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。参照"AppVault カスタムリソース"他のプロバイダーとの AppVault CR の例。
      - **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資格情報への参照を格納します。
        - **spec.providerCredentials.valueFromSecret**: (必須) 資格情報の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。たとえば、 NetApp ONTAP S3、汎用 S3、Google Cloud、Microsoft Azure などです。有効な値は次のとおりです。
      - AWS
      - 紺碧
      - gcp
      - ジェネリックS3
      - オンタップS3
      - ストレージグリッドS3

c. 入力したら `trident-protect-appvault-secondary-destination.yaml`正しい値を持つファイルには、CR を適用します。

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n trident-protect
```

8. 宛先クラスターで、AppMirrorRelationship CR ファイルを作成します。

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. 次の属性を構成します。

- **metadata.name:** (必須) AppMirrorRelationship カスタム リソースの名前。
- **spec.destinationAppVaultRef**: (必須) この値は、宛先クラスター上の宛先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) 宛先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 で定義されている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) クラスター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、クラスター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. 入力したら `trident-protect-relationship.yaml`正しい値を持つファイルには、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) 宛先クラスターで、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
```

宛先クラスタへのフェイルオーバー

Trident Protect を使用すると、複製されたアプリケーションを宛先クラスターにフェイルオーバーできます。この手順により、レプリケーション関係が停止され、宛先クラスターでアプリがオンラインになります。Trident Protect は、ソース クラスター上のアプリが動作中であった場合、そのアプリを停止しません。

手順

1. 宛先クラスタで、AppMirrorRelationship CRファイルを編集します（例： `trident-protect-relationship.yaml` ）の*spec.desiredState*の値を次のように変更します。 `Promoted` 。
2. CR ファイルを保存します。
3. CR を適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. (オプション) フェールオーバーされたアプリケーションで必要な保護スケジュールを作成します。
5. (オプション) レプリケーション関係の状態とステータスを確認します。

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

フェイルオーバー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、宛先クラスター上の実行中のアプリケーションに加えられた変更はすべて破棄されます。

このプロセスでは、レプリケーションを再確立する前に、宛先クラスター上のアプリを停止します。

フェイルオーバー中に宛先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. 宛先クラスタで、AppMirrorRelationship CRファイルを編集します（例： `trident-protect-relationship.yaml` ）spec.desiredStateの値を次のように変更します。 `Established` 。
3. CR ファイルを保存します。
4. CR を適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. フェールオーバーされたアプリケーションを保護するために宛先クラスターに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。

フェイルオーバーされ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、宛先アプリケーションがソース アプリケーションになり、ソースが宛先になります。フェイルオーバー中に宛先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. 元の宛先クラスターで、AppMirrorRelationship CR を削除します。これにより、宛先がソースになります。新しい宛先クラスター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ーに適用して、レプリケーション関係を設定します。
3. 新しい宛先 (元のソース クラスター) が両方の AppVault CR で構成されていることを確認します。
4. 反対方向の値を設定して、反対側のクラスター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宛先ストレージ 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、宛先アプリにフェールオーバーする前にデータを宛先に複製します。

この状況では、ソースと宛先が入れ替わっています。

手順

1. ソース クラスターで、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. 次の属性を構成します。

   - **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. 入力したら `trident-protect-shutdownsnapshot.yaml`正しい値を持つファイルには、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スナップショットを作成する

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。例えば：

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot> -n <application_namespace>
```

2. ソース クラスターで、シャットダウン スナップショットが完了したら、シャットダウン スナップショット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 クラスターで、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、最後のスラッシュの後のすべて) を記録します。

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 次の変更を加えて、新しい宛先クラスターから新しいソース クラスターへのフェイルオーバーを実行します。

> フェイルオーバー手順のステップ2では、 `spec.promotedSnapshot` AppMirrorRelationship CR ファイルのフィールドに追加し、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. 逆再同期の手順を実行します。[フェイルオーバーされたレプリケーション関係を逆再同期する] 。
6. 新しいソース クラスター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、複製されます。
- SnapMirror関係が解除され、宛先ボリュームは読み取り/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複製されたボリューム データを使用して、シャットダウン前のスナップショットから復元されます。
- 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、次の一連の操作によって、フェイルオーバー操作後の「フェイルバック」を実現できます。元のレプリケーション方向を復元するこのワークフローでは、 Trident Protect は、レプリケーション方向を反転する前に、アプリケーションの変更を元のソース アプリケーションに複製 (再同期) します。

このプロセスは、宛先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宛先クラスター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. 実行する[フェイルオーバーされたレプリケーション関係を逆再同期する]手順。
2. 実行する[アプリケーションのレプリケーション方向を逆にする]手順。

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ーで、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構成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。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。

バックアップと復元操作

次のシナリオでバックアップおよび復元操作を実行するには、特定のバックアップおよび復元タスクを自動化できます。

同じクラスターにクローンする

アプリケーションを同じクラスターに複製するには、スナップショットまたはバックアップを作成し、データを同じクラスターに復元します。

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. 同じクラスターで、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"。

別のクラスターにクローンする

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。宛先クラスターにTrident Protect がインストールされていることを確認します。

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" 。

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 が宛先クラスターで設定されていることを確認します。
3. 宛先クラスタでは、"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

バックアップを宛先ストレージ 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する) 次のようにします。

```
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、名前を付けます `trident-protect-snapshot-restore-cr.yaml`。
2. 作成したファイルで、次の属性を構成します。
   - **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**: スナップショットの内容が保存される AppVault 内のパス。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 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**: 復元操作のソース名前空間から宛先名前空間へのマッピング。交換する `my-source-namespace`そして `my-destination-namespace`あなたの環境からの情報を活用します。

     ```
     ---
     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) 使用 `include or exclude`resourceMatchers で定義されたリソースを含めるか除外するかを指定します。含めるまたは除外するリソースを定義するには、次の 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のmetadata.nameフィールドのラベルセレクタ文字列。 "Kubernetesドキュメント" 。例えば： `"trident.netapp.io/os=linux"` 。

  例えば：

```
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. 入力したら `trident-protect-snapshot-restore-cr.yaml`正しい値を持つファイルには、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名前空間に復元します。
   - 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>` 。
   - その `namespace-mapping`引数はコロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間にマッピングします。 `source1:dest1,source2:dest2` 。

     例えば：

```
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。

実行フックの種類

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 によるファイルシステムのフリーズ設定の詳細"。
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、事前スナップショット）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。

実行フック スクリプトを実稼働環境で有効にする前に、必ずテストする必要があります。'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。実稼働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、その後のすべての復元操作に含まれます。

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 96 KB 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スナップショット操作を開始したが、その後キャンセルした場合でも、バックアップまたはスナップショット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ーを追加して、フックが一致するコンテナーを管理できます。フィルターは、すべてのコンテナーで同じコンテナー 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。フィルターを使用すると、実行フックが必ずしもすべての同一コンテナーではなく一部のコンテナー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ーを作成すると、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 フィルターを設定できます。

実行フックに追加する各フィルターは、正規表現を使用してクラスター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。フィルター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ーを除外するフィルターの作成

はサポートされていません。Trident Protectが実行フックフィルタでサポートする正規表現の構文については、以下を参照してください。 "正規表現2（RE2）構文のサポート" 。

> 復元またはクローン操作後に実行される実行フックに名前空間フィルターを追加し、復元またはクローンのソースと宛先が異なる名前空間にある場合、名前空間フィルターは宛先の名前空間にのみ適用されます。

実行フックの例

訪問 "NetApp Verda GitHub プロジェクト"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ます。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

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、名前を付けます `trident-protect-hook.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **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。有効な値は次のとおりです。
     - 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。有効な値は次のとおりです。
     - 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。指定しない場合はデフォルト値は 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はデフォルト値は 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) 実行フック フィルターを構成する基準キー値のペアのオプション リスト。実行フックごとに最大 10 個のフィルターを追加できます。
   - **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。有効な値は次のとおりです。
     - コンテナイメージ
     - コンテナ名
     - ポッド名
     - ポッドラベル
     - 名前空間名
   - **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。例えば：

```
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、所有者、管理者、またはメンバーの権限が必要です。

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、名前を付けます `trident-protect-resource-backup.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **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**: バックアップ コンテンツが保存される AppVault 内のパス。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。例えば：

```
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、名前を付けます `trident-protect-hook-run.yaml`。
2. Trident Protect 環境とクラスタ構成に合わせて次の属性を構成します。
   - **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ます。
   - **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ます。
   - **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ます。

     ```
     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。有効な値は次のとおりです。
     - 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。有効な値は次のとおりです。
     - 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場合があります。

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を削除します。

   ```
   helm uninstall -n trident-protect trident-protect
   ```

3. Trident Protect 名前空間を削除します。

   ```
   kubectl delete ns trident-protect
   ```

TridentとTridentプロテクトのブログ

NetApp TridentとTrident Protect に関する優れたブログを以下でご覧いただけます。

Tridentブログ

- 2025年5月9日:"Amazon EKS アドオンを使用した FSx for ONTAPのTridentバックエンドの自動構成"
- 2025年8月19日:"データの一貫性の強化: Tridentを使用した OpenShift 仮想化のボリューム グループ スナップショット"
- 2025年4月15日:"NetApp TridentとGoogle Cloud NetApp Volumes for SMB プロトコル"
- 2025年4月14日:"Trident 25.02 でファイバーチャネルプロトコルを活用し、Kubernetes 上の永続ストレージを実現する"
- 2025年4月14日:"Kubernetes ブロック ストレージ向けNetApp ASA r2 システムのパワーを解き放つ"
- 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 のインストールの簡素化"
- 2025年3月27日:"Google Cloud NetApp Volumesを使用した中小企業向け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する: 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にファイバーチャネルのサポートを導入"
- 2024年11月26日:"Trident 25.01: 新機能と機能強化によるKubernetesストレージエクスペリエンスの強化"
- 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) の連携"
- 2024年10月29日:"ROSA とAmazon FSx for NetApp ONTAP上の OpenShift Virtualization 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ケーションでONTAPストレージを使用する"
- 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て災害復旧のためのレジストリ フェイルオーバーを自動化する"
- 2025年5月16日:"NetApp Trident Protect を使用した OpenShift Virtualization の災害復旧"
- 2025年5月13日:"Trident Protect バックアップ 復元によるストレージ クラスの移行"
- 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再スケールする"

- 2025年4月3日:"Trident Protect のパワーアップ: 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操作"
- 2025年3月11日:"NetApp Tridentによる GitOps パターンのアプリケーション データ保護への拡張"
- 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 ロールベースアクセス制御のご紹介"
- 2024年11月11日: "tridentctl protect のご紹介: Trident Protect の強力な CLI"
- 2024年11月11日:"Kubernetes によるデータ管理: Trident Protect による新時代"

知識とサポート

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可能性があります。

Trident は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。

Trident の完全な構成の詳細を取得するにはどうすればよいでしょうか**?**

使用 `tridentctl get`Trident の設定に関する詳細情報を取得するには、コマンドを使用します。

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るかに関するメトリックを取得できますか**?**

○管理されているバックエンドの数、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 の操作に関する情報を収集するために使用できる Prometheus エンドポイント。使用することもできます"Cloud Insights"監視および分析用。

Trident を**CSI** プロビジョナーとして使用すると、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`。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は、このTridentのインストール方法をお勧めします。

Kubernetes クラスターに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。参照"Tridentのインストールについて学ぶ"。

Trident をリモートでインストールできますか**?**

○Trident 18.10以降では、以下の機能を備えた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`クラスターへのアクセス。後 `kubectl`アクセスが確認された場合（例えば、 `kubectl get

nodes`リモート マシンからコマンドを実行して確認する場合は、インストール手順に従います。

Tridentで高可用性を構成できますか**?**

Trident は1 つのインスタンスを持つ Kubernetes デプロイメント (ReplicaSet) としてインストールされるため、HA が組み込まれています。デプロイメント内のレプリカの数を増やさないでください。 Tridentがインストールされているノードが失われた場合、またはポッドにアクセスできない場合、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、 Kubernetes API サーバーから読み取り、アプリケーションが新しい PVC をいつ要求するかを判断するため、kube-system にアクセスする必要があります。

Tridentが使用する役割と権限は何ですか**?**

Tridentインストーラーは、Kubernetes クラスターの PersistentVolume、PersistentVolumeClaim 、StorageClass、および Secret リソースへの特定のアクセス権を持つ Kubernetes ClusterRole を作成します。参照"tridentctl インストールのカスタマイズ"。

Tridentがインストールに使用するマニフェスト ファイルを正確にローカルで生成できますか**?**

必要に応じて、 Tridentがインストールに使用するマニフェスト ファイルをローカルで生成および変更できます。参照"tridentctl インストールのカスタマイズ"。

2 つの別々の **Kubernetes** クラスターの **2** つの別々の**Trident**インスタンスで同じ**ONTAP**バックエンド **SVM** を共有できますか**?**

推奨はされませんが、2 つのTridentインスタンスに同じバックエンド SVM を使用することもできます。インストール時に各インスタンスに一意のボリューム名を指定するか、一意の `StoragePrefix`パラメータの `setup/backend.json`ファイル。これは、両方のインスタンスで同じFlexVol volumeが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。

Trident は**Cloud Volumes Services** と連携しますか**?**

はい、 Trident はAzure のAzure NetApp Filesサービスと GCP のCloud Volumes Serviceをサポートしています。

トラブルシューティングとサポート

NetApp はTridentをサポートしていますか?

Tridentはオープンソースで無料で提供されていますが、 NetAppバックエンドがサポートされている限り、NetApp はTrident を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、チケットを発行するためのサポートを受けてください。
2. サポートケースを提起するには、 "NetAppサポート" 。

サポート ログ バンドルを生成するにはどうすればよいですか?

サポートバンドルを作成するには、次のコマンドを実行します。 `tridentctl logs -a` 。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?

問題を作成する "TridentGithub"問題の件名と説明に **RFE** を記載してください。

欠陥はどこに報告すればいいですか?

問題を作成する "TridentGithub"。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に関して簡単に質問があり、それについて説明が必要な場合はどうすればよいですか? コミュニティやフォーラムはありますか?

ご質問、問題、ご要望がございましたら、Tridentまでお問い合わせください。"Discordチャンネル"またはGitHub。

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更新する `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。交換する `myBackend`バックエンド名を例に挙げ、`/path/to_new_backend.json`正しい道への道 `backend.json`ファイル。

Trident がKubernetes ノードを見つけることができません。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は 2 つ考えられます。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、ノードをTridentに登録するためにTridentコントローラーと通信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行われた場合、クラスターに追加された新しい Kubernetes ノードでのみこの問題が発生します。

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は失われません。 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をアップグレード

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 をあるメジャー リリースから次のメジャー リリースにアップグレードすること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する"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します。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 Management LIF と DataLIF 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選択的 LUN マップ"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`明示的に定義されています。参照 "ONTAP SAN 構成オプションと例" 詳細については。
- ONTAP NAS: NetAppは以下を指定することを推奨します `dataLIF`。指定されない場合、 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) を指定できるため、複数のデータ LIF 間で負荷分散を行うラウンドロビン DNS を作成できます。参照"ONTAP NAS の構成オプションと例"詳細については

Trident はONTAPバックエンドに CHAP を設定できますか?

○Trident は、 ONTAPバックエンドの双方向 CHAP をサポートします。これには設定が必要です `useCHAP=true`バックエンド構成で。

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 はバージョン 20.04 以降、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 が作成するエクスポートポリシーにこれらの範囲内にあるノード IP を追加できるようになります。このようにして、 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、以下の IPv6 アドレスの定義をサポートしています。

- `managementLIF`そして `dataLIF`ONTAP NAS バックエンド用。
- `managementLIF`ONTAP SAN バックエンド用。指定できません `dataLIF`ONTAP SAN バックエンド上。

Tridentはフラグを使用してインストールする必要があります `--use-ipv6`（のために `tridentctl`インストール）、 `IPv6` （ Tridentオペレータの場合）、または `tridentTPv6`(Helm インストールの場合) IPv6 経由で機能させるため。

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、バックエンドの管理LIFを以下の方法で更新できます。 `tridentctl update backend`指示。

バックエンドで **DataLIF** を更新することは可能ですか**?**

DataLIFを更新できます `ontap-nas`そして `ontap-nas-economy`のみ。

Trident for Kubernetes で複数のバックエンドを作成できますか**?**

Trident は、同じドライバ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 はバックエンドの資格情報をどのように保存しますか**?**

Trident はバックエンドの資格情報を Kubernetes Secrets として保存します。

Trident はどのようにして特定のバックエンドを選択するのでしょうか**?**

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`そして `additionalStoragePools`パラメータは特定のプールのセットを選択するために使用されます。

Trident が特定のバックエンドからプロビジョニングされないようにするにはどうすればよいですか**?**

その `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、一致するプールは削除されます。

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、以下のパラメータに基づいて適切なバックエンドを選択します。 `StorageClass`そして `PersistentVolumeClaim`。例えば、複数のontap-nasドライババックエンドがある場合、 Tridentは `StorageClass`そして `PersistentVolumeClaim`組み合わせて、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ます `StorageClass`そして `PersistentVolumeClaim`。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**との双方向 **CHAP** をサポートしていますか**?**

○

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**?**

その `ontap-nas-economy`ドライバーは同じFlexVol volumeに最大 200 個の Qtree (50 ～ 300 の間で構成可能)、クラスタ ノードあたり 100,000 個の Qtree、クラスタあたり 2.4M 個の Qtree を作成します。新しい `PersistentVolumeClaim`エコノミー ドライバーによって処理される場合、ドライバーは新しい Qtree を処理できるFlexVol volume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 volumeが存在しない場合は、新しいFlexVol volumeが作成されます。

ONTAP NAS に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で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。

ボリュームをプロビジョニングするときに、 **ONTAP NFS** マウント オプションの明示的なセットを構成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、次の例に従ってください。"ここをクリックしてください。" 。

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、 `exportPolicy`バックエンド定義ファイルで設定されたパラメータ。

ONTAPで**Trident** を介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**?**

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て、 Tridentによってプロビジョニングされたボリュームに暗号化を設定できます。詳細については、以下を参照してください。"Trident がNVE および NAE と連携する仕組み"

Trident を介して**ONTAP**の **QoS** を実装する最適な方法は何ですか**?**

使用 `StorageClasses`ONTAPに QoS を実装します。

Tridentを通じてシン 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ドライバーは、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ドライバーはデフォルトでシン プロビジョニングになります。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`。両方が設定されている場合、 `StorageClass`優先されます。 ONTAPに対して次の設定を行います。

1. の上 `StorageClass`、設定します `provisioningType`属性を厚いとします。
2. バックエンド定義ファイルで、シックボリュームを有効にするには、次のように設定します。 `backend spaceReserve parameter`ボリュームとして。

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によって作成された **NFS PVC** を拡張できますか**?**

○Tridentによって作成された PVC を拡張できます。ボリュームの自動拡張は、 Tridentには適用されないONTAP機能であることに注意してください。

ボリュームが**SnapMirror Data Protection (DP)** モードまたはオフライン モードのときに、ボリュームを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 クォータは**NetApp**クラスター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するはずです。 NetAppストレージが容量不足のために Kubernetes のクォータ設定を尊重できない場合、 Trident はプロビジョニングを試行しますが、エラーが発生します。

Trident を使用してボリューム スナップショットを作成できますか**?**

○オンデマンドのボリューム スナップショットとスナップショットからの永続ボリュームの作成は、 Trident によってサポートされています。スナップショットからPVを作成するには、 `VolumeSnapshotDataSource` 機能ゲートが有効になりました。

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ーは何ですか**?**

本日より、オンデマンドスナップショットサポートが利用可能になりました。 `ontap-nas` 、 `ontap-nas-flexgroup` 、 `ontap-san` 、 `ontap-san-economy` 、 `solidfire-san` 、 `gcp-cvs` 、 そして `azure-netapp-files`バックエンドドライバー。

ONTAPを使用して**Trident**によっ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 バックアップを取得するにはどうすればよいですか**?**

これは以下で入手可能です `ontap-nas`、 `ontap-san` 、 そして `ontap-nas-flexgroup`ドライバー。指定することもできます `snapshotPolicy`のために `ontap-san-economy`FlexVolレベルのドライバー。

これは、 `ontap-nas-economy`ドライバーですが、`qtree` レベルの粒度ではなく、 `FlexVol volume`レベルの粒度に基づいています。 `Trident`によっ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を設定します。 `snapshotPolicy` 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 ポリシーに変更します。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を通じ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、 Tridentでスナップショットのコピーを保存するために、ディスクスペースの特定の割合を予約することができます。 `snapshotReserve`バックエンド定義ファイルの属性。設定した場合 `snapshotPolicy`そして `snapshotReserve`バックエンド定義ファイルでは、スナップショットの予約率は `snapshotReserve`バックエンド ファイルに記載されているパーセンテージ。もし `snapshotReserve`パーセンテージ数値が指定されていない場合、 ONTAP はデフォルトでスナップショット予約パーセンテージを 5 とします。もし `snapshotPolicy`オプションが none に設定されている場合、スナップショットの予約率は 0 に設定されます。

ボリューム スナップショット 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、 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するには、 `snapshotDir`バックエンド定義ファイル内のパラメータ。

Tridentを通じてボリュームの**SnapMirror** を設定できますか**?**

現在、 SnapMirror は、 ONTAP CLI またはOnCommand System Manager を使用して外部で設定する必要があります。

永続ボリュームを特定の**ONTAP**スナップショットに復元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 ポッドを静止します。
2. ONTAP CLI またはOnCommand System Managerを使用して、必要なスナップショットに戻します。
3. アプリケーション ポッドを再起動します。

Trident は、負荷共有ミラーが設定されている **SVM** 上で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、次の方法でサポートバンドルを作成してください。 `tridentctl logs -a -n trident` 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ポッドが正しく立ち上がらない場合は（例えば、Tridentポッドが `ContainerCreating`準備完了コンテナが2つ未満のフェーズ）実行中 `kubectl -n trident describe deployment trident`そして `kubectl -n trident describe pod trident-**`追加の洞察を提供できます。 `kubelet`ログの取得（例： `journalctl -xeu kubelet`) も役立ちます。
- Tridentのログに十分な情報がない場合、次の行を渡してTridentのデバッグモードを有効にしてみてください。 `-d`インストール オプションに基づいて、インストール パラメータにフラグを追加します。

  次に、デバッグが設定されていることを確認します。 `./tridentctl logs -n trident`そして探している `level=debug msg`ログに記録されます。

オペレーターと一緒にインストール

```
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場合があります。これは、出力の「AGE」列を観察することで確認できます。 `kubectl get pod -n trident`。

Trident 20.07および20.10の場合 `tprov`の代わりに `torc`。

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
```

- 各バックエンドのデバッグログを取得するには、以下を含めることもできます。 `debugTraceFlags`バックエンドの定義で。たとえば、 `debugTraceFlags: {"api":true, "method":true,}` Tridentログで API 呼び出しとメソッド トラバーサルを取得します。既存のバックエンドには `debugTraceFlags`構成 `tridentctl backend update`。
- Red Hat Enterprise Linux CoreOS (RHCOS)を使用する場合は、 `iscsid`ワーカーノードで有効になっており、デフォルトで起動します。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を使用する際によく発生する問題は、 "Azure NetApp Files"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の要件の完全なリストについては、以下を参照してください。"Azure NetApp Files"構成。
- PVをコンテナにマウントする際に問題がある場合は、 `rpcbind`インストールされ実行されています。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`実行中です。ステータスを確認することができます `rpcbind`サービスを実行することで `systemctl status rpcbind`またはそれと同等のもの。
- Tridentバックエンドが、 `failed`以前は動作していたにもかかわらず、この状態になった場合は、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。バックエンド情報を更新する `tridentctl update backend`または、 Tridentポッドをバウンスさせると、この問題は解決します。
- 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ときに権限の問題が発生した場合は、次の方法でTridentをインストールしてみてください。 `--in cluster=false`フラグ。これによりインストーラポッドは使用されず、 `trident-installer`ユーザー。
- 使用 `uninstall parameter <Uninstalling Trident>`実行が失敗した後のクリーンアップ用。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。
- Tridentの以前のバージョンにダウングレードしたい場合は、まず `tridentctl uninstall`Trident を削除するコマンド。希望するものをダウンロード "Tridentバージョン"インストールするには `tridentctl install`指示。

- インストールが成功した後、PVCが `Pending`フェーズ、実行中 `kubectl describe pvc`Trident がこの PVC に対して 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。

オペレーターを使用した**Trident**の展開は失敗に終わった

オペレーターを使用してTridentを展開している場合は、 `TridentOrchestrator`変更から `Installing`に `Installed`。あなたが観察すると `Failed`ステータスが「0」で、オペレーターが単独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ー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`状態。

```
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、すでに `TridentOrchestrator`Tridentのインストールに使用されました。各Kubernetesクラスタ はTridentのインスタンスを1つしか持つことができないため、オペレータは、常にアクティブなインスタンス が1つだけ存在するようにします。 `TridentOrchestrator`それが作成できること。

さらに、 Tridentポッドの状態を観察すると、何か異常があるかどうかがわか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、ポッド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。あるいは、削除することもできます `TridentOrchestrator`、修正された正確な定義で新しいものを作成します。

Trident配備失敗 `tridentctl`

何が問題だったのかを解明するために、インストーラーを再度実行してみましょう。`-d`引数を指定するとデバッグ モードがオンになり、問題が何であるか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`再度コマンドを実行します:

```
./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とCRDを完全に削除する

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。

この処理を元に戻すことはできません。Tridentを完全に新規にインストールする場合を除き、こ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をアンインストールする" 。

Tridentオペレーター

Tridentオペレーターを使用して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}}'
```

舵

Helm を使用してTridentをアンインストールし、CRD を完全に削除するには:

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>トライデントctl</code>

Tridentをアンインストールした後CRDを完全に削除するには `tridentctl`

```
tridentctl obliviate crd
```

Kubernetes 1.26 の RWX 生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。

名前空間とポッドを削除しました

Trident管理名前空間 (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しょう。ONTAPバックエンドからネームスペースを直接削除すると、ポッドを削除しようとした後に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 クラスターやその他の機能に影響を与えません。

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
追加する `hostNQN`サブシステムに戻ります。

ONTAPをアップグレードした後、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にもかかわらず、「無効な引数」を報告します。

ONTAPをアップグレードした後、NFSv4.2 クライアントは、NFSv4.2 エクスポートをマウントしようとしたときに「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、 `v4.2-xattrs` オプションは SVM で有効になっていません。回避策を有効にする `v4.2-xattrs` SVM でオプションを無効にするか、このオプションがデフォルトで有効になっているONTAP 9.12.1 以降にアップグレードしてください。

サポート

NetApp はさまざまな方法でTridentのサポートを提供します。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のサポート

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。参照"NetAppソフトウェア バージョンの定義のサポート"。

完全サポート
: Trident は、リリース日から 12 か月間、完全な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 か月間限定のサポートを提供します。

自立
: Trident のドキュメントは、リリース日から 25 ～ 36 か月間ご利用いただけます。

version	完全サポート	限定的なサポート	自立
"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"	2025年10月	2026年10月	2027年10月
"24.06"	2025年6月	2026年6月	2027年6月
"24.02"	2025年2月	2026年2月	2027年2月
"23.10"	—	2025年10月	2026年10月
"23.07"	—	2025年7月	2026年7月
"23.04"	—	2025年4月	2026年4月
"23.01"	—	—	2026年1月
"22.10"	—	—	2025年10月

自立

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、以下を参照してください。 "NetAppナレッジベース (ログインが必要です)" 。

コミュニティサポート

私たちのサイト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。"Discordチャンネル" 。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、次の方法でサポートバンドルを作成してください。 `tridentctl logs -a -n trident`送信先 `NetApp Support <Getting Help>`。

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について詳しく説明します。

Tridentポート

Trident は、Kubernetes 内の通信に次のポートを使用します。

ポート	目的
8443	バックチャネルHTTPS
8001	Prometheusメトリクスエンドポイント
8000	TridentRESTサーバー
17546	Tridentデーモンセット ポッドによって使用される生存/準備状況プローブ ポート

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。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。

TridentREST API

その間"tridentctl コマンドとオプション"これらは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役立ちます。

安全性を高めるために、Trident `REST API`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`ポッド構成内の引数。

REST APIの使用

これらのAPIの呼び出し例については、デバッグを渡してください。(`-d`） フラグ。詳細については、"tridentctl を使用してTrident を管理する" 。

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" 。
- 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" 。

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 (デバッグ、情報、警告、エラー、致命的) を設定します。デフォルトは info です。

Kubernetes

-k8s_pod

このオプションを使用するか、 `-k8s_api_server` Kubernetes サポートを有効にします。これを設定すると、 Trident はそれを含むポッドの Kubernetes サービス アカウントの資格情報を使用して API サーバーに接続します。これは、サービス アカウントが有効になっている Kubernetes クラスター内でTrident がポッドとして実行される場合にのみ機能します。

-k8s_api_server <insecure-address:insecure-port>

このオプションを使用するか、 `-k8s_pod` 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 をポッドの外部にデプロイできるようになります。ただし、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、 Tridentをポッドにデプロイし、 `-k8s_pod`オプション。

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をリッスンします。

-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。ローカルホストでリッスンし、Kubernetes ポッド内で実行している場合、ポッドの外部から REST インターフェースに直接アクセスすることはできません。使用 `-address ""`REST インターフェースをポッド IP アドレスから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 インターフェースを有効にします。デフォルトは true です。

KubernetesとTridentオブジェクト

リソース オブジェクトを読み書きすることで、REST API を使用して Kubernetes およびTridentと対話できます。 Kubernetes とTrident、 Tridentとストレージ、Kubernetes とストレージの関係を規定するリソース 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、およびそれらがどのように相互作用するかを理解するための最も簡単な方法は、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. ユーザーが `PersistentVolumeClaim`新しいリクエスト `PersistentVolume`Kubernetesから特定のサイズの `StorageClass`管理者によって以前に設定されたもの。
2. Kubernetes `StorageClass` Trident をプロビジョナーとして識別し、要求されたクラスのボリュームをプロビジョニングする方法をTrident に指示するパラメータが含まれます。

3. Tridentは自らの `StorageClass`一致するものを識別する同じ名前を持つ `Backends`そして `StoragePools`クラスのボリュームをプロビジョニングするために使用できます。
4. Tridentは、対応するバックエンドにストレージをプロビジョニングし、2つのオブジェクトを作成します。 `PersistentVolume` Kubernetesではボリュームの検索、マウント、処理方法をKubernetesに指示し、 Tridentではボリューム間の関係を保持する `PersistentVolume`そして実際の保管場所。
5. Kubernetesは `PersistentVolumeClaim`新しい `PersistentVolume`。ポッドには、 `PersistentVolumeClaim` PersistentVolume を、それが実行される任意のホストにマウントします。
6. ユーザーが `VolumeSnapshot`既存のPVCの `VolumeSnapshotClass`Tridentを指しています。
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` 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`使用して `VolumeSnapshot`ソースとして。
9. Tridentは必要なスナップショットを識別し、スナップショットを作成するのと同じ一連の手順を実行します。 `PersistentVolume`そして `Volume`。

Kubernetesオブジェクトの詳細については、以下をお読みください。 "永続ボリューム" Kubernetes ドキュメントのセクション。

Kubernetes `PersistentVolumeClaim`オブジェクト

Kubernetes `PersistentVolumeClaim`オブジェクトは、Kubernetes クラスター ユーザーによって行われたストレージの要求です。

標準仕様に加えて、 Trident、バックエンド構成で設定したデフォルトをオーバーライドする場合、ユーザーが次のボリューム固有のアノテーションを指定できます。

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ー
trident.netapp.io/ファイルシステム	ファイルシステム	ontapさん、solidfireさん、ontapさん-economy
trident.netapp.io/cloneFromPVC	クローンソースボリューム	ontap-nas、ontap-san、solidfire-san、azure-netapp-files、gcp-cvs、ontap-san-economy
trident.netapp.io/splitOnClone	クローン上で分割	オンタップナス、オンタップさん
trident.netapp.io/プロトコル	プロトコル	any
trident.netapp.io/エクスポートポリシー	エクスポートポリシー	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/スナップショットポリシー	スナップショットポリシー	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotReserve	スナップショット予約	ontap-nas、ontap-nas-flexgroup、ontap-san、gcp-cvs
trident.netapp.io/スナップショットディレクトリ	スナップショットディレクトリ	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup

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ー
trident.netapp.io/unixパーミッション	unixパーミッション	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/ブロックサイズ	ブロックサイズ	solidfireさん

作成されたPVが `Delete`回収ポリシーでは、`PV` が解放されると（つまり、ユーザーが `PVC` を削除すると）、 `Trident` はPV とバックアップ 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、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`ポリシーに違反している場合、 `Trident` はそれを無視し、管理者が `Kubernetes` とバックエンドからそれをクリーンアップすると想定し、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します。 `PV` を削除しても、 `Trident`バックアップ ボリュームが削除されるわけではないことに注意してください。 `REST API`を使用して削除する必要があります（`tridentctl`）。

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。この方法では、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。その後、スナップショットを使用して新しい PV を作成できます。見てみましょう `On-Demand Volume Snapshots`これがどのように機能するかを確認します。

Tridentはまた、 `cloneFromPVC`そして `splitOnClone`クローンを作成するための注釈。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 を複製できます。

例: ユーザーがすでにPVCを持っている場合 `mysql`ユーザーは、新しい`PVC`を作成することができます。`mysqlclone`注釈を使うと、例えば `trident.netapp.io/cloneFromPVC: mysql`。このアノテーションを設定すると、 Trident はボリュームを最初からプロビジョニングするのではなく、mysql PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- NetAppアイドル ボリュームのクローン作成を推奨しています。
- PVC とそのクローンは同じ Kubernetes 名前空間にあり、同じストレージ クラスを持つ必要があります。
- と `ontap-nas`そして `ontap-san`ドライバーによってはPVCアノテーションを設定することが望ましいかもしれません `trident.netapp.io/splitOnClone`と組み合わせて `trident.netapp.io/cloneFromPVC`。と `trident.netapp.io/splitOnClone`に設定 `true`Trident はクローン ボリュームを親ボリュームから分割し、ストレージ効率を多少犠牲にしてクローン ボリュームのライフ サイクルを親から完全に切り離します。設定なし `trident.netapp.io/splitOnClone`または設定する `false`親ボリュームとクローンボリュームの間に依存関係が作成されるため、クローンを最初に削除しないと親ボリュームを削除できなくなりますが、バックエンドでのスペース消費量は削減されます。クローンを分割することが理にかなっているシナリオは、ボリュームとそのクローンが大きく異なることが予想されるため、 ONTAPが提供するストレージ効率のメリットを享受できない、空のデータベース ボリュームのクローンを作成する場合です。

その `sample-input`ディレクトリには、 Tridentで使用するための PVC 定義の例が含まれています。参照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、こちらをご覧ください。

Kubernetes `PersistentVolume`オブジェクト

Kubernetes `PersistentVolume`オブジェクトは、Kubernetes クラスター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 `PersistentVolume`オブジェクトを作成し、プロビジョニングしたボリュームに基づいて Kubernetes クラスター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。

トライデントベースのPVCを参照するPVCを作成すると、 `StorageClass` 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 を構成する際に、 Trident は次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`オブジェクトは名前で指定されます `PersistentVolumeClaims`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 1 つは、 Trident バックエンド オブジェクトです。

Kubernetes `StorageClass` Trident を使用するオブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters: <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有であり、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	説明
attributes	マップ[文字列]文字列	いいえ	下記の属性セクションを参照してください
ストレージプール	map[文字列]文字列リスト	いいえ	バックエンド名とストレージプールのリストのマッピング
追加のストレージプール	map[文字列]文字列リスト	いいえ	バックエンド名とストレージプールのリストのマッピング

属性	タイプ	必須	説明
ストレージプールを除外する	map[文字列]文字列リスト	いいえ	バックエンド名とストレージプールのリストのマッピング

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	支援
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
プロビジョニングタイプ	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	指定されたプロビジョニング方法	厚い：すべてオンタップ、薄い：すべてオンタップとsolidfireさん
バックエンドタイプ	string	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san、gcp-cvs、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	真、偽	プールはスナップショット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、gcp-cvs
クローン	ブール	真、偽	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、gcp-cvs
暗号化	ブール	真、偽	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

属性	タイプ	値	オファー	要求	支援
IOPS	整数	正の整数	プールはこの範囲のIOPSを保証できる	ボリュームはこれらのIOPSを保証	solidfireさん

[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`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するには、単独で使用します。 Tridentは、すべての条件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`あなたが指定したもの。

使用できない場合は `attributes`クラスに適したプールを自動的に選択するには、 `storagePools`そして `additionalStoragePools`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するためのパラメータ。

使用することができます `storagePools`指定された条件に一致するプールのセットをさらに制限するパラメータ `attributes`。言い換えれば、Tridentは、 `attributes`そして `storagePools`プロビジョニングのパラメータ。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。

使用することができます `additionalStoragePools`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拡張します。 `attributes`そして `storagePools`パラメータ。

使用することができます `excludeStoragePools`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パラメーター。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。

の中で `storagePools`そして `additionalStoragePools`パラメータは、各エントリが次の形式を取る `<backend>:<storagePoolList>`、 どこ `<storagePoolList>`指定されたバックエンドのストレージ プールのコンマ区切りリストです。例えば、 `additionalStoragePools`こんな感じです `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`。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。使用できます `tridentctl get backend`バックエンドとそのプールのリストを取得します。

Kubernetesの属性

これらの属性は、動的プロビジョニング中にTridentによって選択されるストレージ プール/バックエンドには影響しません。代わりに、これらの属性は、Kubernetes 永続ボリューム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	説明	関連するドライバー	**Kubernetes**バージョン
fsタイプ	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	全て
ボリューム拡張を許可する	ブーリアン	真、偽	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、gcp-cvs、azure-netapp-files	1.11+
ボリュームバインディングモード	string	即時、WaitForFirstConsumer	ボリュームバインディングと動的プロビジョニングがいつ実行されるかを選択する	全て	1.19 - 1.26

- その `fsType`パラメーターは、SAN LUN に必要なファイル 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、 `fsType`ストレージ クラスで、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`ポッドのセキュリティコンテキストのみ `fsType`が設定されています。参照"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構成する"ボリューム所有権の設定方法の概要については、 `fsGroup`コンテクスト。 Kubernetesは `fsGroup`次の場合にのみ価値があります:

  - `fsType`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。使用するには `fsGroup`ストレージクラスは依然として `fsType`設定できます `nfs`または null 以外の値。

- 参照"ボリュームを拡張する"ボリューム拡張の詳細については、こちらをご覧ください。
- Tridentインストーラバンドルには、 Tridentで使用するためのストレージクラスの定義例がいくつか用意されています。`sample-input/storage-class-*.yaml` 。 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`オブジェクトは類似している `StorageClasses`。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。

あ `VolumeSnapshotClass`スナップショットを作成するには、管理者が定義する必要があります。ボリューム スナップショット 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
```

その `driver`Kubernetesにボリュームスナップショットのリクエストを指定します。 `csi-snapclass`クラスはTridentによって処理されます。その `deletionPolicy`スナップショット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。いつ `deletionPolicy`設定されている `Delete`スナップショットが削除されると、ボリューム スナップショット オブジェクトと、ストレージ クラスター上の基礎となるスナップショットが削除されます。あるいは、 `Retain`つまり `VolumeSnapshotContent`物理スナップショットは保持されます。

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からの要求を表すのと同様に、ボリューム スナップショットは、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来ると、 Tridentはバックエンドで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、一意のスナップショットを作成してスナップショットを公開します。 `VolumeSnapshotContent`物体。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データ ソースとして使用できます。

> VolumeSnapshot のライフサイクルはソース PVC に依存しません。つまり、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 を削除すると、 Trident はこの PVC のバックアップ ボリュームを 削除中 の状態でマークしますが、完全には削除しません。関連付けられている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`ストレージ クラスター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。類似 `PersistentVolumeClaim`そして `PersistentVolume`オブジェクトの場合、スナップショットが作成されると、 `VolumeSnapshotContent`オブジェクトは、 `VolumeSnapshot`スナップショットの作成を要求したオブジェクト。

その `VolumeSnapshotContent`オブジェクトには、スナップショットを一意に識別する詳細が含まれています。 `snapshotHandle` 。これ `snapshotHandle`PVの名前と `VolumeSnapshotContent`物体。

スナップショット要求が届くと、 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`オブジェクトを作成し、スナップショットを Kubernetes API に公開します。

通常、 `VolumeSnapshotContent`物体。例外は、"ボリュームスナップショットをインポートする" Tridentの外部で作成されました。

Kubernetes `VolumeGroupSnapshotClass`オブジェクト

Kubernetes `VolumeGroupSnapshotClass`オブジェクトは類似している `VolumeSnapshotClass`。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 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 グループ スナップショットは、単一のボリューム グループ スナップショット クラスに関連付けられます。

あ `VolumeGroupSnapshotClass`スナップショットのグループを作成するには、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
```

その `driver`Kubernetesにボリュームグループのスナップショットを要求することを指定します。 `csi-group-snap-class`クラスはTridentによって処理されます。その `deletionPolicy`グループ スナップショット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。いつ `deletionPolicy`設定されている `Delete`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 オブジェクトと、ストレージ クラスター上の基礎となるスナップショットが削除されます。あるいは、 `Retain`つまり `VolumeGroupSnapshotContent`物理スナップショットは保持されます。

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、一意のスナップショットを作成してスナップショットを公開します。 `VolumeGroupSnapshotContent`物体。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データ ソースとして使用できます。

VolumeGroupSnapshot のライフサイクルはソース PVC に依存しません。つまり、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 を削除すると、 Trident はこの PVC のバックアップ ボリュームを 削除中 の状態でマークしますが、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 スナップショットを表します。これは、 `PersistentVolume`ストレージ クラスター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。類似 `PersistentVolumeClaim`そして`PersistentVolume`オブジェクトの場合、スナップショットが作成されると、 `VolumeSnapshotContent`オブジェクトは、 `VolumeSnapshot`スナップショットの作成を要求したオブジェクト。

その `VolumeGroupSnapshotContent`オブジェクトには、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。 `volumeGroupSnapshotHandle`およびストレージ システム上に存在する個別の volumeSnapshotHandles。

スナップショット要求が届くと、 Trident はバックエンドにボリューム グループ 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`オブジェクトを作成し、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用の別個のストアが不要になります。

Tridentは `CustomResourceDefinition`Tridentバックエンド、 Tridentストレージ クラス、 Tridentボリュームなどの Tridentオブジェクトの ID を保持するためのオブジェクト。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 スナップショット フレームワークでは、ボリューム スナップショット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、バックエンドを次のように作成すると、 `tridentctl` 、対応する`tridentbackends`CRD オブジェクトは Kubernetes で使用するために作成されます。

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。
- Tridentをアンインストールする場合 `tridentctl uninstall`コマンドを実行すると、 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。参照"Tridentをアンインストールする"Trident を完全に削除し、最初から再構成する方法を理解します。

Trident `StorageClass`オブジェクト

Trident はKubernetes に適したストレージクラスを作成します `StorageClass`指定するオブジェクト`csi.trident.netapp.io`プロビジョナー分野で。ストレージクラス名はKubernetesの名前と一致します`StorageClass`それが表すオブジェクト。

> Kubernetesでは、Kubernetesが起動するとこれらのオブジェクトが自動的に作成されます。`StorageClass` Trident をプロビジョナーとして使用するものが登録されます。

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ためのストレージ 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を登録するときに作成されるものと想定されます。 `StorageClass`オブジェクト。

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。1 つのTridentインスタンスで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。もう一つはKubernetes `StorageClass`物体。

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" 。

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に使用できる個別の場所を表します。 ONTAPの場合、これらは SVM のアグリゲートに対応します。 NetApp HCI/ SolidFireの場合、これらは管理者が指定した QoS バンドに対応します。 Cloud Volumes Serviceの場合、これらはクラウド プロバイダーのリージョンに対応します。各ストレージ プールには、パフォーマンス特性とデータ保護特性を定義する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、これらは直接 `PersistentVolumes`。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。これらを表示して、 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 を削除すると、対応するTridentボリュームが 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	説明
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

属性	タイプ	必須	説明
ストレージクラス	string	はい	ボリュームをプロビジョニングするとき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。「ファイル」または「ブロック」
内部名	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。
クローンソースボリューム	string	いいえ	ontap (nas, san) & solidfire-*: クローン元のボリュームの名前
クローン上で分割	string	いいえ	ontap (nas, san): クローンを親から分離する
スナップショットポリシー	string	いいえ	ontap-*: 使用するスナップショットポリシー
スナップショット予約	string	いいえ	ontap-*: 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
エクスポートポリシー	string	いいえ	ontap-nas*: 使用するエクスポートポリシー
スナップショットディレクトリ	ブール	いいえ	ontap-nas*: スナップショットディレクトリが見えるかどうか
unixパーミッション	string	いいえ	ontap-nas*: 初期UNIX権限
ブロックサイズ	string	いいえ	solidfire-*: ブロック/セクターサイズ
ファイルシステム	string	いいえ	ファイルシステムの種類

Tridentは生成する `internalName`ボリュームを作成するとき。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）をボリューム名に追加して、次のような形式の名前にします。 `<prefix>-<volume-name>` 。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>`）。 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してREST APIを使用してボリュームを直接プロビジョニングできますが、Kubernetesのデプロイメントでは、ほとんどのユーザーが標準のKubernetesを使用することが想定されています。 `PersistentVolumeClaim`方法。 Trident は、プロビジョニング プロセスの一環としてこのボリューム オブジェクトを自動的に作成します。

Trident `Snapshot`オブジェクト

スナップショットはボリュームの特定時点のコピーであり、新しいボリュームをプロビジョニングしたり状態を復元したりするために使用できます。 Kubernetesでは、これらは直接 `VolumeSnapshotContent`オブジェクト。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。

それぞれ `Snapshot`オブジェクトには、以下にリストするプロパティが含まれます。

属性	タイプ	必須	説明
version	弦	はい	Trident APIのバージョン（「1」）
名前	弦	はい	Tridentスナップショット オブジェクトの名前
内部名	弦	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	弦	はい	スナップショットが作成される永続ボリュームの名前
ボリューム内部名	弦	はい	ストレージ システム上の関連付けられた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。これらを表示して、Trident がプロビジョニングした内容を確認できます。

Kubernetesの場合 `VolumeSnapshot`オブジェクト要求が作成されると、 `Trident` は、バックアップ ストレージ システムにスナップショット オブジェクトを作成して動作します。その `internalName`このスナップショットオブジェクトのプレフィックスを組み合わせることで生成されます `snapshot-`と `UID`の `VolumeSnapshot`オブジェクト（例： `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。 `volumeName`そして `volumeInternalName`バックアップボリュームの詳細を取得することによって入力されます。

Trident `ResourceQuota`物体

Tridentデーモンセットは `system-node-critical`優先度クラス (Kubernetes で使用できる最高の優先度クラス) により、 Trident は正常なノード 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るようになり、リソースの負荷が高いクラスターでは、優先度の低いワークロードをTridentデーモンセット ポッドがプリエンプトできるようになります。

これを達成するために、Tridentは `ResourceQuota`Tridentデーモンセットの「システム ノード クリティカル」優先度クラスが満たされていることを確認するオブジェクト。デプロイメントとデーモンセットの作成の前に、 Tridentは `ResourceQuota`オブジェクトを検索し、見つからない場合は適用します。

デフォルトのリソースクォータと優先度クラスをさらに制御する必要がある場合は、 `custom.yaml`または設定する `ResourceQuota`Helm チャートを使用したオブジェクト。

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、以下を参照してください。"Kubernetes: リソースクォータ" 。

掃除 `**ResourceQuota**`インストールに失敗した場合

まれに、インストールが失敗する場合もありますが、 `ResourceQuota`オブジェクトが作成されたら、まず"アンインストール"その後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、手動で削除してください。 `ResourceQuota`物体。

取り除く `ResourceQuota`

自分でリソースの割り当てを制御したい場合は、Tridentを削除できます。 `ResourceQuota`次のコマンドを使用してオブジェクトを作成します:

```
kubectl delete quota trident-csi -n trident
```

ポッドセキュリティ標準（PSS）とセキュリティコンテキスト制約（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 Engine 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。次のセクションでは、 Tridentによって設定される権限について詳しく説明します。

PSS は Pod 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、以下を参照してください。"Kubernetes: セキュリティ" 。

必要な **Kubernetes** セキュリティコンテキストと関連フィールド

許可	説明
特権階級	CSI ではマウント ポイントが双方向である必要があるため、 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" 。
ホストネットワーク	iSCSI デーモンに必要です。 `iscsiadm` 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	開始に必要 `rpc-statd`NFS 用。 `Trident`はホストプロセスを照会して、 `rpc-statd` NFS ボリュームをマウントする前に実行します。

機能	その `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`

セコンプ	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`（「スーパー特権コンテナ」）ドメインで実行され、非特権コンテナは `container_t`ドメイン。の上 `containerd`、 と `container-selinux`インストールされると、すべてのコンテナは `spc_t`ドメインは、SELinux を事実上無効にします。したがって、Tridentは `seLinuxOptions`コンテナに。
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。

ポッドセキュリティ標準（**PSS**）

ラベル	説明	デフォルト
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	Tridentコントローラーとノードがインストール名前空間に受け入れられることを許可します。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`変更されました。もしそうなら、 Trident を再インストールしてください。

ポッドセキュリティポリシー（PSP）

フィールド	説明	デフォルト
`allowPrivilegeEscalation`	特権コンテナは特権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。	空の
`allowedCapabilities`	非特権Tridentコンテナにはデフォルト セット以上の機能は必要なく、特権コンテナには可能なすべての機能が付与されます。	空の
`allowedFlexVolumes`	Tridentは、"FlexVolume ドライバー"したがって、許可されるボリュームのリストには含まれません。	空の
`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、このリストを設定しても利点はありません。	空の
`allowedProcMountTypes`	Tridentは `ProcMountTypes`。	空の
`allowedUnsafeSysctls`	Tridentは危険なものを必要としません `sysctls`。	空の
`defaultAddCapabilities`	特権コンテナに機能を追加する必要はありません。	空の
`defaultAllowPrivilegeEscalation`	権限昇格の許可は各Tridentポッドで処理されます。	`false`
`forbiddenSysctls`	いいえ `sysctls`許可されます。	空の
`fsGroup`	Tridentコンテナは root として実行されます。	`RunAsAny`
`hostIPC`	NFSボリュームをマウントするには、ホストIPCと通信する必要があります。 `nfsd`	`true`

フィールド	説明	デフォルト
`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、次の場合に必要です。 `rpc-statd`ノード上で実行されています。	`true`
`hostPorts`	Trident はホスト ポートを使用しません。	空の
`privileged`	Tridentノード ポッドは、ボリュームをマウントするために特権コンテナを実行する必要があります。	`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は使用しない `RuntimeClasses`。	空の
`seLinux`	Tridentは設定されない `seLinuxOptions`現在、コンテナ ランタイムと Kubernetes ディストリビューションが SELinux を処理する方法に違いがあるためです。	空の
`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`

セキュリティコンテキスト制約（SCC）

ラベル	説明	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`
`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`

ラベル	説明	デフォルト
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、次の場合に必要です。 `rpc-statd`ノード上で実行されています。	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	Tridentノード ポッドは、ボリュームをマウントするために特権コンテナを実行する必要があります。	true
allowedUnsafeSysctls	Tridentは危険なものを必要としません sysctls。	none
allowedCapabilities	非特権Tridentコンテナにはデフォルト セット以上の機能は必要なく、特権コンテナには可能なすべての機能が付与されます。	空の
defaultAddCapabilities	特権コンテナに機能を追加する必要はありません。	空の
fsGroup	Tridentコンテナは root として実行されます。	RunAsAny
groups	この SCC はTridentに固有であり、そのユーザーにバインドされます。	空の
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は設定されない `seLinuxOptions`現在、コンテナ ランタイムと Kubernetes ディストリビューションが SELinux を処理する方法に違いがあるためです。	空の
seccompProfiles	特権コンテナは常に「制限なしで」実行されます。	空の
supplementalGroups	Tridentコンテナは root として実行されます。	RunAsAny

ラベル	説明	デフォルト
`users`	この SCC をTrident名前空間内の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

Trident向けNetAppソフトウェア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については、各リリースの通知ファイルで確認できます。 https://github.com/NetApp/trident/ 。