Trident 26.10 ドキュメント

Trident

NetApp
October 30, 2026

This PDF was generated from https://docs.netapp.com/ja-jp/trident/index.html on October 30, 2026.
Always check docs.netapp.com for the latest.

目次

Trident 26.10 ドキュメント 1
リリースノート 2
新機能 2
26.10の新機能 2
26.06.2の新機能 6
26.06.1の新機能 6
26.06の新機能 7
26.02.1の新機能 10
26.02の新機能 11
25.10の新機能 14
25.06.2 の変更点 16
25.06.1 の変更点 16
25.06の変更点 17
25.02.1の変更点 19
25.02の変更点 19
24.10.1の変更点 21
24.10の変更点 21
24.06の変更点 23
24.02の変更点 24
23.10の変更点 25
23.07.1の変更点 25
23.07の変更点 25
23.04の変更点 26
23.01.1の変更点 28
23.01の変更点 28
22.10の変更 29
2007年22月の変更 30
2004年10月22日の変更 31
22.01.1 の変更 32
22.01.0 の変更 32
21.10.1 の変更点 33
21.10.0 の変更点 33
既知の問題 34
詳細については、こちらをご覧ください 35
以前のバージョンのドキュメント 35
ONTAP ASA r2ストレージシステムに対するNetApp Tridentのサポート 35
サポートされている操作 36
サポートされていない操作 36
既知の問題 36

Trident 22.10以前からのアップグレード後もボリュームは接続されたままになる 37
VolumeSnapshotsがReadyToUse状態に到達しない 37
大容量ファイルのResticバックアップのリストアが失敗することがある 37
はじめに 38
Tridentの詳細 38
Tridentの詳細 38
Tridentのアーキテクチャ 39
概念 42
クイックスタートガイド（Trident） 46
次の手順 47
Tridentの要件 47
Tridentに関する重要な情報 47
サポートされるフロントエンド（オーケストレーションツール） 48
サポートされるバックエンド（ストレージ） 48
TridentによるKubeVirtとOpenShiftによる仮想化のサポート 49
Kubernetes および OpenShift ホスト型コントロールプレーン上の Kubernetes に対する Trident サポート 49
機能の要件 50
テスト済みのホストオペレーティングシステム 51
ホストの設定 51
ストレージシステムの構成： 51
Tridentポート 51
コンテナイメージと対応する Kubernetes バージョン 51
Trident をインストール 53
Tridentのインストールについて 53
Trident 26.10に関する重要な情報 53
作業を開始する前に 53
インストール方法を選択します 54
インストールモードを選択します 55
メソッドとモードに基づいてプロセスを選択します 56
インストール方法を切り替える 56
その他の既知の設定オプション 57
Tridentオペレータを使用してインストール 57
Tridentオペレータを手動で導入（標準モード） 57
Tridentオペレータを手動で導入（オフラインモード） 62
Helm（標準モード）を使用してTridentを導入 68
Helm（オフラインモード）を使用したTridentのオペレータの導入 76
Tridentオペレータのインストールをカスタマイズ 84
tridentctlを使用してインストールします 95
tridentctlを使用したTridentのインストール 95
Trident tridentctl インストールのカスタマイズ 99

OpenShift認定オペレータを使用してインストール 100
OpenShift OperatorHubを使用したTridentのインストール 100
OpenShift認定Tridentオペレーターに切り替える 104
Tridentを使用 107
Tridentワーカーノードを準備する 107
適切なツールを選択する 107
ノードサービスの検出 107
NFS ボリューム 108
iSCSI ボリューム 108
NVMe/TCPホリユウム 112
SCSI over FCボリューム 113
SMBボリュームをプロビジョニングする準備をします 115
COSIを使用してオブジェクトストレージをプロビジョニングする 116
概要 117
要件および制限事項 117
COSI を有効にする 117
StorageGRIDオブジェクトストアを設定する 118
ONTAP S3オブジェクトストアを構成する 119
BucketClassを作成してバケットをリクエストする 120
ワークロードにバケットへのアクセス権限を付与する 120
バックエンドの構成と管理 121
Tridentバックエンドの設定 121
Azure NetApp Files の特長 122
Google Cloud NetAppボリューム 139
NetApp HCI または SolidFire バックエンドを設定します 177
ONTAP SAN ドライバ 182
ONTAP NAS ドライバ 211
ONTAPバックエンドでバランスの取れた配置を使用する 245
NetApp ONTAP 対応の Amazon FSX 247
kubectl を使用してバックエンドを作成します 282
バックエンドの管理 288
ストレージクラスの作成と管理 297
Tridentストレージクラスの作成 297
Tridentストレージクラスの管理 300
ボリュームのプロビジョニングと管理 301
Tridentボリュームをプロビジョニングする 301
ボリュームの移動 304
Tridentボリュームの拡張 308
RWX NVMe サブシステムの制限について 318
コントローラのスケーラビリティ 319
自動ボリューム拡張 325

ONTAPボリュームの自動拡張を設定 332
自動拡張ポリシーの管理 336
Tridentボリュームのインポート 345
Tridentのボリューム名とラベルのカスタマイズ 357
名前空間をまたいでTrident NFSボリュームを共有する 359
名前空間をまたいだTridentボリュームのクローン作成 363
SnapMirrorによるボリュームのレプリケート 365
CSI トポロジを使用します 372
Tridentスナップショットの操作 379
Tridentボリュームグループのスナップショットの操作 387
Tridentの管理と監視 392
Tridentのアップグレード 392
Tridentのアップグレード 392
オペレータにアップグレードしてください 393
tridentctlを使用したTridentのアップグレード 398
tridentctlを使用したTridentの管理 398
コマンドとグローバルフラグ 399
コマンドオプションとフラグ 400
プラグインのサポート 405
Tridentストレージ操作の監視 405
概要 405
手順 1 ： Prometheus ターゲットを定義する 406
手順 2 ： Prometheus ServiceMonitor を作成します 406
ステップ 3 ： PrompQL を使用して Trident 指標を照会する 408
Trident AutoSupportテレメトリの詳細 409
Trident指標を無効にする 410
Trident をアンインストールします 410
元のインストール方法を決定する 410
Tridentオペレータのインストールをアンインストールする 411
のアンインストール `tridentctl` インストール 411
Trident for Docker 413
導入の前提条件 413
要件を確認します 413
NVMeツール 415
FCツール 416
Tridentの導入 417
Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降） 417
従来の方法（バージョン 1.12 以前） 419
システム起動時にTridentを起動する 421
Trident をアップグレードまたはアンインストールする 422
アップグレード 422

をアンインストールします 423
ボリュームを操作します 424
ボリュームを作成します 424
ボリュームを削除します 424
ボリュームのクローンを作成します 424
外部で作成されたボリュームにアクセス 426
ドライバ固有のボリュームオプション 426
ログを収集します 431
トラブルシューティング用にログを収集する 431
一般的なトラブルシューティングのヒント 432
複数のTridentインスタンスの管理 432
Docker Managed Plugin （バージョン 1.13 / 17.03 以降）の手順 432
従来の（バージョン 1.12 以前）の場合の手順 433
ストレージ構成オプション 433
グローバル構成オプション 433
ONTAP の設定 434
Element ソフトウェアの設定 442
既知の問題および制限事項 444
Trident Docker Volume Plugin を旧バージョンから 20.10 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。 444
ボリューム名は 2 文字以上にする必要があります。 444
Docker Swarmには特定の動作があり、ストレージとドライバの組み合わせごとにTridentがDocker Swarmをサポートできないようになっています。 444
FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。 445
ベストプラクティスと推奨事項 446
Trident導入に関する推奨事項 446
専用のネームスペースに導入します 446
クォータと範囲制限を使用してストレージ消費を制御します 446
Tridentストレージ構成のベストプラクティス 446
プラットフォームの概要 446
ONTAP と Cloud Volumes ONTAP のベストプラクティス 446
SolidFire のベストプラクティス 451
詳細情報の入手方法 453
TridentをKubernetesと統合する 453
ドライバの選択と展開 453
ストレージクラスの設計 456
仮想プールの設計 457
ボリューム操作 458

指標サービス 462
Tridentデータ保護と災害復旧 463
Tridentのレプリケーションとリカバリ 463
SVMのレプリケーションとリカバリ 464
ボリュームのレプリケーションとリカバリ 465
Snapshotデータの保護 465
Tridentによるステートフルアプリケーションのフェイルオーバーの自動化 465
フォースデタッチの詳細 466
自動フェイルオーバーの詳細 466
セキュリティ 471
Tridentセキュリティに関する推奨事項 471
Trident Linux統合キー設定（LUKS） 472
RWX Linux Unified Key Setup (LUKS) ボリュームのリースが停止した状態を修復する 479
Trident Kerberosによる通信中の暗号化 481
Trident Protectでアプリケーションを保護する 489
Tridentプロテクトについて学ぶ 489
次の手順 489
Trident Protectをインストールする 489
Tridentプロテクトの要件 489
Trident Protectのインストールと設定 493
Trident Protect CLIプラグインをインストールする 496
Trident Protectのインストールをカスタマイズする 499
Trident Protectの管理 504
Trident Protectの認証とアクセス制御を管理する 504
Trident Protectリソースを監視する 510
Trident Protect サポートバンドルを生成する 514
Tridentプロテクトのアップグレード 516
アプリケーションの管理と保護 518
Trident Protect AppVault オブジェクトを使用してバケットを管理する 518
Trident Protectのスナップショットとバックアップ用のAppVaultメタデータを暗号化する 529
Trident Protectで管理するアプリケーションを定義する 536
Trident Protectを使用してアプリケーションを保護する 541
Trident Protectでボリュームグループのスナップショットを使用する 551
リストアアプリケーション 556
NetApp SnapMirrorでアプリケーションをレプリケート 579
Trident Protectを使用してアプリケーションを移行する 595
KubeVirt 仮想マシンを Trident Protect で保護する 599
Trident Protect実行フックを管理する 611
Trident Protectをアンインストールする 618
TridentとTridentプロテクトのブログ 620
Tridentブログ 620

Trident Protectブログ . . . 620
知識とサポート . . . 622
よくある質問 . . . 622
一般的な質問 . . . 622
KubernetesクラスタへのTridentのインストールと使用 . . . 622
トラブルシューティングとサポート . . . 623
Tridentのアップグレード . . . 624
バックエンドとボリュームを管理 . . . 625
トラブルシューティング . . . 629
全般的なトラブルシューティング . . . 629
オペレータを使用したTridentの導入に失敗 . . . 631
Tridentの導入に失敗しました `tridentctl` . . . 632
TridentとCRDを完全に取り外します。 . . . 633
RWX rawブロックネームスペースo Kubernetes 1.26でNVMeノードのステージング解除が失敗する . . . 633
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。 . . . 634
サポート . . . 634
Tridentのサポートライフサイクル . . . 635
セルフサポート . . . 635
コミュニティサポート . . . 635
NetAppテクニカルサポート . . . 635
を参照してください。 . . . 636
参照 . . . 637
Tridentポート . . . 637
概要 . . . 637
Trident REST API . . . 639
REST APIを使用する状況 . . . 639
REST APIを使用する . . . 639
コマンドラインオプション . . . 640
ロギング . . . 640
Kubernetes . . . 640
Docker です . . . 641
REST . . . 641
Kubernetes オブジェクトと Trident オブジェクト . . . 641
オブジェクトは相互にどのように相互作用しますか。 . . . 641
Kubernetes `PersistentVolumeClaim` オブジェクト . . . 642
Kubernetes `PersistentVolume` オブジェクト . . . 644
Kubernetes `StorageClass` オブジェクト . . . 644
Kubernetes `VolumeSnapshotClass` オブジェクト . . . 648
Kubernetes `VolumeSnapshot` オブジェクト . . . 648
Kubernetes `VolumeSnapshotContent` オブジェクト . . . 649

`VolumeGroupSnapshotClass`Kubernetesオブジェクト 649
`VolumeGroupSnapshot`Kubernetesオブジェクト 649
`VolumeGroupSnapshotContent`Kubernetesオブジェクト 650
Kubernetes `CustomResourceDefinition` オブジェクト 650
Trident `StorageClass`オブジェクト 651
Trident バックエンドオブジェクト 651
Trident `StoragePool`オブジェクト 651
Trident `Volume`オブジェクト 651
Trident `Snapshot`オブジェクト 653
Trident `ResourceQuota`オブジェクト 654
PODセキュリティ標準（PSS）およびセキュリティコンテキストの制約（SCC） 655
必須のKubernetes Security Contextと関連フィールド 655
PODセキュリティ標準（PSS） 656
PoDセキュリティポリシー（PSP） 656
セキュリティコンテキストの制約（SCC） 658
法的通知 660
著作権 660
商標 660
特許 660
プライバシーポリシー 660
オープンソース 660

Trident 26.10 ドキュメント

リリースノート

新機能

リリース ノートには、NetApp Trident の最新バージョンの新機能、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。

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

26.10の新機能

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

Trident

拡張機能

- * Kubernetes * ：
  - StorageGRIDからのCOSIベースのオブジェクトストレージバケットの動的プロビジョニングのサポートを追加しました。詳細については、"COSIを使用してオブジェクトストレージをプロビジョニングする"を参照してください。
  - NVMeデータLIFを定期的に更新し、構成変更を検知して対応する機能を追加しました。詳細については、"NVMeデータLIFを更新する"を参照してください。
  - ONTAP-SANおよびONTAP-SAN-Economyドライバによってプロビジョニングされた新規ボリュームにおけるONTAP Autosizeのサポートを追加しました。詳細については、"ONTAPボリュームの自動拡張を設定"を参照してください。
  - Kubernetes 1.37のサポートを追加しました。サポートされているバージョンと必要なイメージについては、"Tridentの要件"を参照してください。
  - ONTAPボリュームリカバリキューをスキップするために、CommVaultおよびRubrikバックアップのバックアップクローンPVC認識を追加しました（問題 "#1182"）。
  - 新しいNVMe RWXボリューム用の `maxNamespacesPerSuperSubsystem`バックエンドオプションを追加しました。詳細については、"NVMe RWXサブシステムの制限"を参照してください。
  - KubernetesデプロイメントのHTTPSコントローラバックチャネルに代わり、 `TridentNode`の仕様とステータスを介したCRDベースのCSIノード登録を追加しました。
  - OpenShift上のGoogle Cloud Workload Identity Federation（WIF）向けに、構成可能なオーディエンスを使用してサービスアカウントトークンを投影するためのインストーラサポートを追加しました。インストールおよびバックエンドの構成の詳細については、"Workload Identity フェデレーションを使用して Google Cloud NetApp Volumes を設定する"を参照してください。
  - `trident_backend_info`メトリクスに `backend_state`ラベルを追加し、Prometheus で個々のバックエンドの健全性を識別できるようにしました（問題 "#1088"）。Trident メトリクスの詳細については、"Tridentの監視"を参照してください。
- ONTAP-SANドライバがiSCSI経由でプロビジョニングするRWXボリュームに対するLUKS暗号化のサポートを追加しました。詳細については、"Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"

を参照してください。

- ONTAP-NAS-FlexGroupドライバーとDockerのGA同時実行サポートを追加しました。ASA r2の同時実行機能がテクニカルプレビューから一般提供（GA）に移行しました。サポートされているドライバーと動作については、"コントローラのスケーラビリティ"を参照してください。
- すべてのiSCSIパスが利用可能ではない場合の、パスの削除とiSCSIの自己修復を使用したCSIボリューム拡張のサポートを追加しました。詳細については、"ボリュームを展開します"を参照してください。
- root squashが有効なONTAP-NASおよびONTAP-NAS-FlexGroupボリュームの作成時にONTAPボリュームのUNIXグループIDを設定するための `trident.netapp.io/unixGroupID` PVCアノテーションを追加しました（問題 "#1079"）。
- NVMe/TCP、NFSv4、iSCSIのTalos OSサポートを追加しました（問題 "#806"）。サポートされているプラットフォームとワーカーノードの要件については、"Tridentの要件"および"ワーカーノードを準備します"を参照してください。
- バックエンドごとにONTAP RESTおよびZAPIクライアントのタイムアウトを設定するための、オプションの `storageAPITimeout`バックエンドパラメータを追加しました。
- ONTAP REST 名前ルックアップに `SQL=false`を追加するための ONTAP バックエンドオプション `useRESTSQLBypass`が追加されました。
- ONTAP-NAS-Economyのqtreeインポートがテクニカルプレビューから GAに昇格しました。インポート手順については、"Tridentボリュームのインポート"を参照してください。

の修正

- * Kubernetes *：
  - `enableConcurrency`が `true`の場合に、バックエンドの更新時またはコントローラの再起動時にONTAP-NASのエクスポートポリシールールが誤って削除される問題を修正しました（問題 "#1166"、"#1179"）。エクスポートポリシーの設定については、"ONTAP NASの設定オプションと例"を参照してください。
  - 認証情報やQoSの変更など、重要度の低いバックエンドの更新によって、ボリュームごとの孤立スキャンがトリガーされ、大規模なバックエンドでCSI操作がブロックされる可能性がある問題を修正しました。スキャンは、SVMまたはストレージプレフィックスが変更された場合にのみ実行されるようになりました。
  - CSI gRPCリクエストダンプ、起動環境出力、ログに記録された `TridentBackendConfig`仕様など、デバッグログに認証情報が表示される問題を修正しました。起動時の出力では、Trident `TRIDENT_*`変数の値のみがログに記録され、その他の変数は名前でリストされるようになりました。
  - CSIの公開とボリュームの移動の間に競合が発生し、古いアクセス情報を持つボリュームがアタッチされる可能性があった問題を修正しました。ボリューム移動の詳細については、"ボリュームの移動" を参照してください。
  - 修正 `NodeGetInfo`：ノード登録が完了する前に空のトポロジラベルが返され、トポロジ対応プロビジョニングが失敗する可能性がある問題（問題 "#1131"）。トポロジ対応プロビジョニングについては、"CSIトポロジを使用したストレージのプロビジョニング"を参照してください。
  - 別のポッドが接続中に共有サブシステムが切断される可能性があった、まれなNVMe切断競合を修正しました。
  - 同時実行環境でマウント済みボリュームをクローンする際に発生していた、NVMe PVCクローンの公開失敗および親ネームスペースのマッピングの誤りを修正しました。
  - `nameTemplate`で構成されたONTAP-SAN-Economyバックエンド上のPVCのスナップショット作成の失敗を修正しました。バックエンド構成の例については、"ONTAP のSAN構成オプションと例"を参照

してください。

- RESTネームインデックスが追いつく前に削除された場合、ONTAP RESTバックエンドが新しく作成されたFlexVolを孤立させる問題を修正しました。TridentはUUIDによってボリュームを削除するようになりました。
- バックエンドLIFにアクセスできない場合に、コントローラーの起動がハングアップする問題を修正しました。影響を受けるバックエンドはマークされ `Failed`、起動が続行されます。
- マルチキーノードセレクタで構成されている場合に、オペレータが管理するコントローラおよびノードポッドがリコンサイルのたびに再起動する問題を修正しました。ノードセレクタの設定については、"Tridentオペレータのインストールをカスタマイズ"を参照してください。
- TridentConfiguratorで作成されたAzure NetApp Files `TridentBackendConfig`リソースで、作成後の検証で必要となる `trident.netapp.io/configurator`ラベルが欠落している問題を修正しました。
- Trident CSIサービスがIPv6 ClusterIPを使用している場合にノードの登録が失敗する問題を修正しました（問題 "#1164"）。
- 古いボリュームの公開がまだ解除されている最中にワーカーノードが削除されて再参加した際に、Tridentノードが完全に削除される問題を修正しました。
- Tridentのキャッシュされたエクスポートポリシーが古い場合に、すでに接続されているノードでONTAP-NASボリュームがNFSエクスポートアクセスを失う問題を修正しました。エクスポートポリシーの設定については、"ONTAP NASの設定オプションと例"を参照してください。
- 修正済み `tridentctl`インストール時のクリーンアップ処理でノード修復テンプレートCRDが削除され、強制デタッチが有効になっている状態でのその後のインストールが妨げられる可能性がある問題。インストールの詳細については、"tridentctlを使用したTridentのインストール"を参照してください。
- 再試行遅延を追加することで、スナップショット削除の呼び出しがONTAP APIを過負荷にする問題を修正しました。スナップショット操作については、"スナップショットの管理"を参照してください。
- TridentオペレータのHelmチャートのAutoSupportイメージヘルパーのコメントとcloudProviderの値のクォーティングを修正しました。
- ビジー状態または既存のジョブに対するONTAP ZAPIのボリューム作成処理を修正し、CSIの再試行によってFlexVolsが時期尚早に破棄されたり、作成/破棄ループが発生したりしないようにしました。
- 既に存在しない整合性グループに対してボリュームグループスナップショットを再試行した際に発生するパニックを修正しました。ボリュームグループのスナップショットについては、"ボリュームグループのスナップショットを作成する"を参照してください。
- サポート対象のKubernetesの最小バージョンと最大バージョンに基づくTridentサポートの警告を削除しました。

- 共有名前付きロックにおける競合状態を修正しました。この競合状態により、同一リソースに対する操作が重複する可能性がありました。
- スナップショットが見つからなかった場合にASAドライバの初期化が失敗する問題を修正しました。
- クラスタノードがプライマリHAゾーンにない場合の、リージョンFlex Unifiedプールに対するGCNVプロビジョニングを修正しました。Tridentは、そのHAゾーンを必須のCSIトポロジとして扱わなくなりました。ゾーンプールは変更されていません。Flex Unifiedの設定については、"ブロックストレージ（iSCSI）を設定する"を参照してください。
- 正規のバックエンドIDがTridentボリューム名と異なる場合に、バックエンドの調整中にGCNV Flex Unified NASおよびSANボリュームが誤って孤立したボリュームとしてマークされる問題を修正しました（問題 "#1189"）。GCNVバックエンド構成については、"Google Cloud NetApp Volumes を設定"を参照してください。

- トランスポートがカーネルに組み込まれているホストでのNVMe/TCP検出を修正しました（問題 "#1019"）。ノード要件については、"ワーカーノードを準備します"を参照してください。
- ONTAPボリュームがスナップショットリザーブを含めて要求に対してすでに十分な大きさである場合に、ONTAP-NASおよびONTAP-NAS-FlexGroupの拡張が無限に再試行される問題を修正しました。ボリューム拡張については、"ボリュームを展開します"を参照してください。
- ジャンクションパスの欠落が原因でボリュームマウントが失敗する問題を修正しました。ONTAP NAS構成については、"ONTAP NASドライバーの設定"を参照してください。
- `storagePrefix` にダッシュが含まれている場合に ONTAP-SAN-Economy ボリュームの作成が失敗する問題を修正しました。バックエンド構成オプションについては、"ONTAP のSAN構成オプションと例" を参照してください。
- ソースのFlexVolエクスポートルールが保持されるように、自動エクスポートポリシーを使用した読み取り専用のONTAP-NASクローンの公開を修正しました。ONTAP NASの構成については、"ONTAP NASの設定オプションと例"を参照してください。
- HTTP転送の再試行動作を修正しました（問題 "#1139"）。
- SCSIデバイスの削除をスキップするゴーストデバイス検出を修正しました（問題 "#1161"）。
- 同時実行が有効なプロビジョニングにおいて、選択したノードの優先ゾーンとは異なるトポロジゾーンが選択される問題を修正しました。同時実行とトポロジの動作については、"コントローラのスケーラビリティ"を参照してください。
- 同時実行が有効になっている場合に、同等のバックエンド間でボリュームの配置が偏っていた問題を修正しました。バックエンドは、同時実行ではないプロビジョニングと同様に、ランダムに均等に選択されるようになりました。詳細については、"コントローラのスケーラビリティ"を参照してください。
- Tridentノードポッドの起動中にホストネットワークアドレステーブルが変更された場合に、ノードが欠落したIPアドレスで登録され、NFSエクスポートポリシールールが削除される可能性がある問題を修正しました。
- RESTポーリングを超えてクローン作成が継続し、PVCが分割されていないFlexCloneとしてバインドされる問題を修正しました。Tridentは今後、その後のジョブ作成の失敗を検知するようになります。
- ONTAP-SANボリューム拡張時に、ノード上で誤ったサイズ変更失敗が報告される問題を修正しました。ボリューム拡張については、"ボリュームを展開します"を参照してください。

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】 **Kubernetes**：ONTAP S3からのCOSIベースのオブジェクトストレージバケットの動的プロビジョニングのサポートを追加しました。詳細については、"COSIを使用してオブジェクトストレージをプロビジョニングする"を参照してください。

Tridentプロテクト

拡張機能

- AppVaultのスナップショットおよびバックアップメタデータの暗号化サポートを追加しました。各AppVaultにカスタムのvaultパスワードを使用します。vaultパスワードをローテーションすると、メタデータの暗号化キーが再暗号化され、AppVault内のすべてのKopiaおよびResticリポジトリが更新されます。詳細については、"Trident Protectのスナップショットとバックアップ用のAppVaultメタデータを暗号化する"を参照してください。

- アプリケーションライセンスがMACアドレスに紐付けられている場合に便利な、リストア時にKubeVirt VMのMACアドレスを保持するサポートを追加しました。これを有効にするには、protect.trident.netapp.io/retain-mac-addressアノテーションを「true」に設定します。詳細については、"高度なTrident Protect復元設定を使用する"を参照してください。
- アプリケーションの関連する永続ボリューム要求を同じ時点でキャプチャする、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。Applicationカスタムリソースの `spec.volumeGroups`で、ラベル、KubeVirt仮想マシン、またはその両方を使用してボリュームグループを定義します。詳細については、"Trident Protectでボリュームグループのスナップショットを使用する"を参照してください。
- レプリケートされたアプリケーションの以前のスナップショットへのフェイルオーバーをサポートし、データが破損または侵害される前の時点から復旧できるようにしました。詳細については、"Trident Protectを使用して以前のスナップショットにフェイルオーバーする"を参照してください。

の修正

- `trident-protect-pre-upgrade-hook`ジョブで `trident-protect-controller-manager` ServiceAccountが見つからないために Trident Protect のインストールが失敗する問題を修正しました（"問題 #1154"）。
- 所有者が存在する場合にのみ `ownerReference`を含めるように、PVCのみのリストアを修正しました。

26.06.2の新機能

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

Trident

拡張機能

の修正

- * Kubernetes *：
  - Trident コントローラの再起動時、または `enableConcurrency`が `true`に設定されているときに `TridentBackendConfig`が更新された際に、ONTAP-NAS エクスポートポリシールールがサイレントに削除される問題を修正しました。これにより、すでにマウントされているボリューム（"問題 #1166"、"問題 #1179"）が孤立する可能性がありました。
  - バックエンドの更新時に、認証情報やQoSなどの重要でない変更に対してボリュームごとの孤立スキャンを実行する問題を修正しました。これにより、大規模なバックエンドでCSI操作が数時間ブロックされる可能性がありました。スキャンは、SVMまたはストレージプレフィックスが変更された場合にのみ実行されるようになりました。
  - Tridentのデバッグログ（CSI gRPCリクエストダンプ、起動環境ダンプ、CRDコントローラによって記録された `TridentBackendConfig`仕様など）に認証情報が表示される問題を修正しました。起動環境ダンプは、Trident自身の `TRIDENT_*`変数の値のみをログに記録し、その他の変数は名前でリストするようになりました。

26.06.1の新機能

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

Trident

の修正

- * Kubernetes *：
  - CSIの公開とボリュームの移動の間の競合状態を修正しました。この問題により、古いアクセス情報を使用してボリュームがアタッチされる可能性がありました。
  - NodeGetInfo がノード登録完了前に空のトポロジーラベルを返す可能性があった競合状態を修正し、トポロジーを考慮したプロビジョニングの失敗を防ぎます。
  - 別のポッドがまだ接続中の間に共有サブシステムを切断してしまう可能性があった、まれな NVMe 切断競合を修正しました。
  - 同時実行環境でマウントされたボリュームのクローンを作成する際に発生する、NVMe PVCクローンの公開失敗（および親ネームスペースのマッピングの誤り）を修正しました。
  - nameTemplateを使用したONTAP-SAN-EconomyバックエンドでのPVCのスナップショット作成エラーを修正しました。

拡張機能

- * Kubernetes *：
  - オプション `storageAPITimeout`（秒）をONTAPバックエンド設定に追加し、バックエンドごとにRESTおよびZAPI APIクライアントのタイムアウトを設定できるようにしました。

26.06の新機能

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

Trident

拡張機能

- * Kubernetes *：
  - ONTAP-NAS-EconomyおよびONTAP-SAN-Economyドライバーの同時実行に対する一般提供（GA）サポートを追加し、この機能をテクニカルプレビューから正式版に移行しました。また、Azure NetApp Files（ANF）およびSolidFireドライバーの同時実行に対するGAサポートも追加しました。
  - Kubernetes 1.36 のサポートが追加されました。
  - Google Cloud NetApp Volumes （GCNV）NASおよびSANドライバでのボリュームの自動拡張に対する一般提供（GA）サポートを追加しました。
  - 新しいボリュームのプロビジョニング時に最適なアグリゲートを選択するための統合 ONTAP バランス配置を追加しました（"問題 #64"）。
  - ONTAP-NAS-Flexgroup ドライバに ONTAP AFX サポートを追加しました。
  - Red Hat OpenShift Migration Toolkit for Virtualizationを使用する際の仮想マシンの移行を簡素化するため、NetApp Shift Toolkitとの統合を追加しました。
  - GCNV SANドライバにクラウドIDのサポートを追加しました。
  - Trident Volume Publication （TVP）の伝播およびボリュームの自動拡張ワークフローに対して、従属ボリュームのサポートを追加しました。

- 同時実行が有効な状態で Trident を実行する際のボリューム更新およびボリューム グループ スナップショットのサポートを追加しました。
- GCNV向けに動的なgRPCレート制限機能を追加しました。
- コントローラーの準備状況実行プローブを追加し、コントローラーとサイドカーのデフォルトのKubernetes API QPSおよびバースト設定を調整しました。
- アタッチ、暗号化設定、ファイルシステムフォーマットの各ステップを分離することで、ノード操作のシーケンスを改善しました。
- セキュリティ向上のため、マウントされた Trident コンテナのルートファイルシステムを読み取り専用にしました。
- ONTAP バックエンドでクローンボリュームのソーススナップショットが削除された際に、クローンボリュームスナップショットのベストエフォート型スナップショット削除機能を追加しました。
- Veeam Kasten、Trilio、および Cohesity によって作成された一時的な PVC が ONTAP ボリューム リカバリ キューでスタックしないように、バックアップ クローン PVC の認識を追加しました。
- 特殊リージョンをサポートするために、AWS ARNの処理を強化しました。
- ONTAP-SAN-ECONOMYドライバのサイズ変更操作中にFlexvolが縮小するのを防ぎます。

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。

- **[Tech Preview]**: ONTAP-NAS-Economy qtreeインポートのサポートを追加しました。
- **[Tech Preview]**: ASA-R2 (SAN および NVMe) ドライバーの同時実行のサポートを追加しました。
- **[Tech Preview]**: ONTAP-SANおよびONTAP-NASドライバ向けに、GCNV ONTAPモードのHTTP転送を追加しました。

の修正

- * Kubernetes *：
  - 同時実行される公開/非公開、クローン、キャッシュ、およびバックエンド更新操作における競合状態と状態処理を修正しました。
  - ONTAP-NASおよびONTAP-NAS-Economyドライバにおける、サブボリュームおよび読み取り専用クローンに対するエクスポートポリシーの競合状態と同時パブリッシュ/アンパブリッシュ処理の問題を修正しました。
  - スナップショット ポリシーが異なる場合の、MetroClusterフェイルオーバー後のONTAP-NAS-Economy FlexVolの検出を修正しました（"問題 #1082"）。
  - ONTAP-NAS-Economy ボリューム削除の再試行により、長時間実行中の qtree 削除が中断される問題を修正しました（"問題 #1121"）。
  - ONTAP-SANおよびONTAP-SAN-Economyのインポートおよびサイズ変更の動作を修正しました。これには、fsTypeの検証、ボリュームメタデータの処理、および自動拡張モードの動作が含まれます。
  - 同時ボリューム作成時のNVMe/TCPネームスペース競合状態を修正しました（"問題 #1089"）。
  - 利用可能なパスとポータルが一致しない場合に発生していたiSCSIサイズ変更の失敗を修正し、繰り返しサイズ変更時の動作を改善しました。
  - ボリューム使用状況が原因で発生する、ONTAP-SANボリュームの削除失敗を修正しました。

- ONTAP REST エラー処理を修正し、SVM 状態タイムアウトの動作を改善しました。
- GCNV SAN のゾーンプールとホストグループの処理を修正し、ボリューム識別子の変換を正規化し、最小サイズ未満の作成リクエストを正規化しました。
- ネイティブGCNV容量プールの選択において、ONTAPモードのプールを除外するように修正しました。
- KubeVirt の VM フェイルオーバーを修正しました。Trident 自動ワークロードフェイルオーバーを使用して VM フェイルオーバーを高速化します。
- オペレーター管理インストールのRBACに関する問題を修正しました。これには、ノードのClusterRolesの欠落、およびアップグレード時の不正なClusterRoleとClusterRoleBindingの調整が含まれます。これは、Trident Operatorチャートを使用するOperatorおよびHelmベースのインストールに適用されます。
- デフォルトの優先度値を省略し、YAMLテンプレートのインデントを修正することで、TridentConfiguratorおよび関連するCRDにおける永続的なGitOpsの差分を修正しました（"問題 #915"、"問題 #1120"）。
- Helm アップグレード中の TridentVolumePublication CRD のパッチ適用を修正しました（"問題 #1151"）。
- ノード登録の再試行前にCSI gRPCサービスを開始することで、CSIの起動順序を修正しました。
- 同時実行時のセキュアSMBクローン構成オーバーレイを修正しました。
- アップグレード後に Trident コントローラーポッドに注釈が欠落していた問題を修正しました（"問題 #1004"）。
- 大規模SVM上での低速なqtreeプロビジョニングを改善しました（"問題 #1142"）。
- ControllerPublish が誤ったデフォルトのファイルシステムタイプではなく、ボリューム構成のファイルシステムタイプを使用するように修正しました。
- 孤立した dm デバイスが CSI NodeUnstageVolume の繰り返し失敗を引き起こす可能性がある問題を修正しました。
- ゴーストマルチパスアーティファクトおよび古い TridentNode リソースをソフト削除されたノードのクリーンアップ後に削除することで、古いノードとマルチパスのクリーンアップを改善しました。

Tridentプロテクト

拡張機能

- OpenShift Service CA Operator との統合を追加し、Trident Protect アドミッション Webhook の TLS 証明書管理を自動化します。Helm の値で `useOpenshiftServiceCA`を `true`に設定することで有効にします。詳細については、"Trident Protectのインストールをカスタマイズする"を参照してください。
- OpenShift VirtualizationでのKubeVirt VMのバックアップとリストアのサポートが追加されました。ネームスペース内のすべてのVM、または名前で指定した特定のVMを保護し、バックアップまたはスナップショットから個々のVMを選択的にリストアできます。詳細については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。
- リソース変換のサポートを追加し、復元プロセス中にKubernetesリソース属性を追加、削除、変更できるようにしました。詳細については、"カスタムリソースを使用して復元する際にリソースを変更します"を参照してください。
- AppMirrorRelationship（AMR）での `storageClassMapping`のサポートが追加され、単一の AMR 定義内で個々の PVC のストレージクラスマッピングが可能になりました。詳細については、"Trident Protectを

使用してアプリケーションのレプリケーションを設定および管理する"を参照してください。

の修正

- ソースアプリケーションの名前空間がデスティネーション クラスタに存在しない場合に、クラスター間仮想マシンのリストアがアドミッション Webhook エラーで失敗する問題を修正しました。
- 既存のアプリケーションカスタムリソースとの仕様の不一致が原因で、アプリケーション作成ステップで選択的な仮想マシンの In Place リストアが失敗する問題を修正しました。既存のアプリケーションが再利用されるようになりました。
- 仮想マシンとその PersistentVolumeClaim の両方が削除された場合に、仮想マシンの In Placeリストア ジョブが `Running`状態でハングする問題を修正しました。PersistentVolumeClaim が存在しない場合は、致命的なエラーとして処理されるようになりました。
- 既存の保護スケジュールを編集した際に、誤って即座にバックアップまたはスナップショットが実行されてしまう問題を修正しました。
- スナップショットファイナライザーが、アプリケーションが削除された後にそのカスタムリソースを取得しようとする問題を修正しました。アプリケーションが見つからない場合、ファイナライザーは処理をスキップするようになりました。
- PersistentVolumeClaimのコピー手順でスパースコピーを使用した仮想マシンのIn Placeリストア後に、保護された固定ストレージ容量が大幅に増加する問題を修正しました。
- Trident Protect が ONTAP のバッキング スナップショットの削除を確認する前に Kubernetes スナップショット オブジェクトを削除していた問題を修正しました。ファイナライザーは `VolumeSnapshot`削除を待機するようになりました。
- 仮想マシンのフリーズ操作が失敗した場合にスナップショット処理が進まない問題を修正しました。フリーズおよびフリーズ解除操作は、失敗した場合でも続行されるようになりました。
- AppMirrorRelationshipの反転、反転再同期、および再同期の操作中に、KubeVirt仮想マシン アプリケーションがPersistentVolumeClaimsを誤って再作成する問題を修正しました。

26.02.1の新機能

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

Trident

の修正

- * Kubernetes *：
- Google Cloud NetApp Volumes（GCNV）SANの動作を修正し、ブロック（iSCSI）ワークロード用のFlex Unified ZONALプールをサポートするようにしました。ゾーン構成とリージョン構成の両方がサポートされるようになりました。
- Google Cloud NetApp Volumes（GCNV）NAS UNIFIEDプールの処理を修正しました。 `InternalID`が存在しない場合のボリュームの作成、検証、命名、およびフォールバックボリューム検索操作を改善しました。
- 利用可能なパスとポータルが一致しない場合に、iSCSIデバイスのサイズ変更操作が失敗する可能性がある問題を修正しました。
- `blkid`がLUKSデバイスのファイルシステムタイプを誤って識別し、新しいボリュームの作成ができなかった断続的な問題を修正しました。

- 大規模な LUKS のパブリッシュおよびステージング操作が Trident コントローラをブロックする可能性がある問題を修正しました。ユーザーは、ボリュームに関連付けられた LUKS パスフレーズを追跡することが求められるようになりました。
- iSCSIマルチパスパーティションがゴーストデバイスとして表示され、CSI `NodeUnstage`処理が成功しない問題を修正しました。
- 同時ボリューム作成時のNVMe/TCPネームスペース競合状態を修正しました（問題#1089）。
- 同じ秒内にスナップショットが実行された場合の同時クローンスナップショット操作を修正しました。
- KubeVirt VM が `dataVolumes`を使用して Trident Automated Workload Failover でフェイルオーバーできない問題を修正しました。
- Trident並行コアの使用中に無効な `TridentMirrorRelationship`カスタムリソースを削除する際に発生したエラーを修正しました。
- `tridentactionmirrorupdates` Kubernetes CRD YAML定義のインデントを修正しました（問題#1120）。
- REST qtree非同期削除API呼び出しを修正しました（問題#1121）。
- `ControllerPublish`誤ったデフォルトのファイルシステムタイプではなく、ボリューム構成のファイルシステムタイプを使用するように修正しました。
- コントローラの起動ログを修正し、同時ドライバのステータスを正しく報告するようにしました。
- 並行処理における中断されたバックエンドの処理を修正しましたTrident。
- 特殊リージョンをサポートするために、AWS ARNの処理を強化しました。

Tridentプロテクト

の修正

セキュリティ上の脆弱性を解決するために、Trident Protectの依存関係を更新しました：

- gRPC-Go："CVE-2026-33186を使用したチャンク アップロード署名要求がサポートされるようになりました。"
- Go x/crypto SSH："CVE-2024-45337を使用したチャンク アップロード署名要求がサポートされるようになりました。"
- OpenTelemetry-Go："CVE-2026-24051"
- Go ランタイム："CVE-2025-22871を使用したチャンク アップロード署名要求がサポートされるようになりました。"
- 追加の依存関係："CVE-2025-23109を使用したチャンク アップロード署名要求がサポートされるようになりました。"

26.02の新機能

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

Trident

拡張機能

- * Kubernetes *：
  - ONTAP-NAS（NFSのみ）、ONTAP-SAN（iSCSI、FCP、NVMe）、およびGoogle Cloud NetApp Volumes（GCNV）ドライバの同時実行に対する一般提供（GA）サポートを追加し、この機能をTech Previewから正式版に移行しました。詳細については、"コントローラのスケーラビリティに関するドキュメント"を参照してください。
  - カスタムのユーザー定義Trident AutoGrowポリシーに基づくTridentでのボリュームの自動拡張のサポートを追加しました。詳細については、"自動ボリューム拡張ドキュメント"を参照してください。
  - 強化されたTridentノードの同時実行性により、NVMe ボリュームのノード操作のスケーラビリティが向上します。詳細については、"NVMeボリュームのドキュメント"を参照してください。
  - PVCアノテーション(`tieringPolicy `tieringMinimumCoolingDays`を使用したGoogle Cloud NetApp Volumes NASボリュームの自動階層化のサポートが追加されました。プールの選択とクローンの継承が含まれます。詳細については、"Google Cloud NetApp Volumes ドキュメントの自動階層化を設定する"を参照してください。
  - `google-cloud-netapp-volumes-san`ドライバを使用したGCNV SANブロック（iSCSI）ボリュームのサポートが追加されました。プロビジョニング、LUNアクセス用のノード単位のホストグループマッピング、ボリュームからのクローン操作が含まれます。詳細については、"Google Cloud NetApp Volumes ブロック構成ドキュメント"を参照してください。
  - Amazon FSx for NetApp ONTAP の自動バックエンド構成のサポートを追加しました。必要なパラメータを含む StorageClass を作成すると、Trident は対応するバックエンドと、必要に応じて VolumeSnapshotClass を自動的に作成します。詳細については、"Amazon FSx for NetApp ONTAPバックエンド構成ドキュメント" を参照してください。
  - Azure GovernmentやAzure Chinaなどのさまざまなクラウドのサポートと、Azure NetApp Filesバックエンドのカスタムクラウド構成を追加しました。詳細については、"Azure NetApp Files バックエンド構成ドキュメント"を参照してください。
  - Kubernetes 1.35 のサポートが追加されました。詳細については、"要件ドキュメント"を参照してください。

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。

- **[Tech Preview]**：ONTAP-NAS-Economy および ONTAP-SAN-Economy ドライバーの同時実行のサポートが追加されました。

既知の問題

`csi-snapshotter`サイドカーには既知の問題があります。すべての Kubernetesバージョンにおいて、VolumeGroupSnapshots `v1beta1`がVolumeSnapshotsの `ReadyToUse`状態への到達を防ぎます。

回避策は 2 つあります：

1. VolumeGroupSnapshots CRDを削除してVolumeGroupSnapshotsを無効にし、Tridentを再インストールします。

2. VolumeGroupSnapshots `v1beta2`および `snapshot-controller`バージョン8.4.0以降をインストールしてから、Tridentを再インストールしてください。VolumeGroupSnapshotsはKubernetesバージョンv1.34より前では機能しません。

の修正

- * Kubernetes *：
  - ONTAP-NAS、ONTAP-NAS-Economy、Google Cloud NetApp Volumesドライバで読み取り専用クローンを非公開にすると、ソース ボリュームからエクスポート ポリシー ルールが削除される問題を修正しました"問題 #1086"。
  - Bitnami のパブリック イメージの廃止に伴うプルの失敗を防ぐため、kubectl イメージを軽量の Alpine ベースのバリアントに切り替えました "問題 #1080"。
  - Tridentアップグレード中の既存のデプロイメントの注釈の保持を修正"問題 #1004"。
  - 両方のストレージ クラスが同じバックエンドを指している場合、異なるストレージ クラス間でのクローン作成を許可する"問題 #1104"。
  - ネットワーク遅延のあるクラウド環境でのタイムアウトによって発生するノード準備の失敗を修正しました。クラウドベースのインストールのタイムアウト値を増やしました。
  - プロセスが再試行状態になったときにファイルシステム タイプ属性が設定されないままになる LUN 作成の問題を修正しました。
  - REST APIボリューム ルックアップを修正し、ボリューム状態を無視することで、ボリューム クエリ中の誤検出を防止しました。
  - 大規模に使用する場合の ontap-nas-economy ドライバーの Trident コントローラー効率が向上しました。
  - ontap-san-economy ドライバで LUN インポート時に internalID を設定します。
  - より多くのサブネットを処理できるように、Azure Resource Graph のクエリ制限が引き上げられました。
  - 一部のバックアップアプリケーションとの競合状態を回避するために、CSIとONTAP クローン スプリットのタイムアウトを改善しました"問題 #1098"、"問題 #1100"。
  - LUKSエラーメッセージの抑制を修正"問題 #1069"。
  - iSCSI プロトコルと NVMe プロトコルの両方における古い LUKS マッパーの処理を修正しました。強化されたクリーンアップ ロジックにより、孤立したデバイス マッパーによるマウントの失敗を防止します。
- RWX NVMe ボリュームのスケール制限を修正しました。
- OpenTelemetry-Goパッケージを更新して"CVE-2026-24051"を修正しました。

Tridentプロテクト

拡張機能

- Trident Protectは、In Placeリストアの前に保護スケジュールを自動的に無効にし、進行中の処理をキャンセルして、リストアの完了後に再度有効にします。詳細については、"Trident Protectを使用してアプリケーションを復元する"を参照してください。
- `runImmediately`フィールドをスケジュールCRに追加し、 `--run-immediately` CLIフラグを追加して、スケジュールの作成時に即時バックアップまたはスナップショットをトリガーできるようにしまし

た。詳細については、"データ保護スケジュールを作成"を参照してください。

- リストア CR の `destinationApplicationName`フィールドまたは `--destination-app-name` CLI フラグを使用して、リストアされたアプリケーションのカスタム名を指定するためのサポートが追加されました。詳細については、"Trident Protectを使用してアプリケーションを復元する"を参照してください。

の修正

- 必要なサービス アカウントが使用可能になる前にポッドが作成されたために発生するリストアの失敗を修正しました。
- `Roles`と `RoleBindings`がアプリケーションのリストア中にスキップされる問題を修正しました。
- 正しく設定されているにもかかわらず、 `tridentctl-protect get appvaultcontent`出力に元のクラスタ名が表示されない問題を修正しました。
- 欠落した `pipefail`処理により無視されていたKopiaリストアエラーを修正しました。
- 永続ボリュームを除外するリソース フィルタによって発生するスナップショットおよびバックアップの障害を修正しました。
- ネームスペース全体で同じ名前のPVCを持つマルチネームスペースアプリケーションにおいて、データ損失につながる可能性があった誤ったPVCリストアを修正しました。

25.10の新機能

機能強化、修正、廃止など、 TridentおよびTrident Protect の新機能について説明します。

Trident

拡張機能

- * Kubernetes *：
  - ONTAP -SAN (iSCSI および FC) に加えて、 ONTAP-NAS NFS およびONTAP -SAN-Economy ドライバー用の v1beta1 ボリューム グループ スナップショット Kubernetes API による CSI ボリューム グループ スナップショットのサポートが追加されました。見る"ボリュームグループスナップショットの操作"。
  - ONTAP-NAS およびONTAP-NAS-Economy (両方の NAS ドライバーの SMB を除く)、およびONTAP-SAN およびONTAP-SAN-Economy ドライバーの強制ボリューム デタッチによる自動ワークロード フェイルオーバーのサポートが追加されました。見る"Tridentによるステートフル アプリケーションのフェイルオーバーの自動化"。
  - FCP ボリュームのノード操作のスケーラビリティを向上させるために、 Tridentノードの同時実行性が強化されました。
  - ONTAP NAS ドライバーにONTAP AFX サポートが追加されました。見る"ONTAP NASの設定オプションと例"。
  - TridentOrchestrator CR および Helm チャート値を介してTridentコンテナの CPU およびメモリ リソース要求と制限を構成するためのサポートが追加されました。（"第1000号" 、"第927号" 、"第853号" 、"第592号" 、"第110号" ）。
  - ASAr2 パーソナリティの FC サポートを追加しました。見る"ONTAP のSAN構成オプションと例"。
  - HTTP ではなく HTTPS で Prometheus メトリックを提供するオプションが追加されました。見る"Tridentの監視"。

- オプションを追加しました `--no-rename`ボリュームをインポートするときに元の名前を保持しますが、ボリュームのライフサイクルは Tridentに管理させます。見る"ボリュームをインポート"。
- Tridentデプロイメントは、system-cluster-critical 優先度クラスで実行されるようになりました。

- Tridentコントローラーに、helm、operator、tridentctl 経由でホスト ネットワークを使用するオプションを追加しました ("第858号" ）。
- ANFドライバに手動QoSサポートを追加し、Trident 25.10で本番環境対応になりました。この実験的な機能強化はTrident 25.06で導入されました。

実験的な機能強化

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]**: ONTAP-SAN ドライバー (統合ONTAP 9 の iSCSI および FCP プロトコル) の既存のテクニカル プレビューに加えて、 ONTAP-NAS (NFS のみ) およびONTAP-SAN (統合ONTAP 9 の NVMe) の同時実行のサポートが追加されました。

の修正

- **Kubernetes**:
  - Windows DaemonSet とコンテナ イメージの命名と一致するように Linux DaemonSet を node-driver-registrar に標準化することで、CSI node-driver-registrar コンテナ名の不一致を修正しました。
  - レガシー qtree のエクスポート ポリシーが適切にアップグレードされない問題を修正しました。
- オープンシフト:
  - SCC の allowHostDirVolumePlugin が false に設定されていたために、Openshift の Windows ノードでTridentノード ポッドが起動しない問題を修正しました ("第950号" ）。
- Kubernetes API QPS が Helm 経由で設定されない問題を修正 ("第975号" ）。
- 同じ Kubernetes ノード上の NVMe ベースの XFS ファイルシステム PVC のスナップショットに基づいて永続ボリューム要求 (PVC) をマウントできない問題を修正しました。
- バックエンドごとに一意または共有のサブシステム名 (例: netappdvp_subsystem) を追加することで、NDVP モードでホスト/Docker を再起動した後の UUID 変更の問題を修正しました。
- Trident を23.10 より前のバージョンから 24.10 以降にアップグレードする際に iSCSI ボリュームのマウント エラーが修正され、「無効な SANType」の問題が解決されました。
- Tridentコントローラーを再起動せずにTridentバックエンドの状態がオンライン/オフラインに移行しない問題を修正しました。
- PVC のサイズ変更が遅くなる原因となる断続的な競合状態を修正しました。
- ボリュームクローンの失敗時にスナップショットがクリーンアップされない問題を修正しました。
- カーネルによってデバイス パスが変更されたときにボリュームのステージ解除に失敗する問題を修正しました。
- LUKS デバイスがすでに閉じられているためにボリュームのステージ解除に失敗する問題を修正しました。
- ストレージ操作が遅いために ContextDeadline エラーが発生する問題を修正しました。
- Trident Operator は、設定可能な k8s-timeout を待機してTrident のバージョンを確認します。

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。

拡張機能

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

    見る"サポートされているバックアップとスケジュールの注釈"。
- バックアップ CR によって使用される PVC バインド タイムアウトを構成するための注釈をスケジュール CR に追加しました。 `protect.trident.netapp.io/pvc-bind-timeout-sec` 。見る"サポートされているバックアップとスケジュールの注釈"。
- 改良された `tridentctl-protect`実行フックの失敗を示す新しいフィールドを含むバックアップおよびスナップショットのリスト。

25.06.2 の変更点

Trident

の修正

- **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おける既知の問題があるため、25.06.1にアップグレードしないでください。この問題に対処するため、25.06.2がまもなくリリースされる予定です。

の修正

- * 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

拡張機能

- * 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。を参照して "ボリュームグループスナップショットの操作"

    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"ONTAP のSAN構成オプションと例"。
  - ONTAP-NASおよびONTAP-NAS-Economyボリュームにセキュアな SMB サポートを追加しました。ActiveDirectory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、セキュリティが強化されました。を参照して "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。を参照して "SnapMirrorによるボリュームのレプリケート"

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、コントローラー操作を並列実行できるようになり、高負荷環境や大規模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。

の修正

- * 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ままになる可能性がある 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 `nfsMountOptions`未設定時に NFSv3 にデフォルト設定される問題を修正しました。現在、NFSv3 と NFSv4 の両方のプロトコルがサポートされています。 `nfsMountOptions`が指定されていない場合、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 ("問題#831" ）。
  - スナップショットから作成されたPVC のエクスポート ポリシーが欠落していた問題を修正しました ("問題#1016" ）。
  - ANF ボリューム サイズが 1 GiB 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。
- サイズ変更に失敗したにもかかわらず、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。

拡張機能

- 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。

の修正

- 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るバグを修正しました。現在はすべてのスナップショット注釈が正しく適用されています。
- 定義されていない場合は、デフォルトでデータ ムーバー暗号化 (Kopia / Restic) のシークレットを定義します。
- S3 appvault 作成の検証とエラー メッセージが改善されました。
- AppMirrorRelationship (AMR) は、失敗を回避するために、バインドされた状態の PV のみを複製するようになりました。
- 多数のバックアップがある 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で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

の修正

- * Kubernetes *：
  - デフォルト以外のイメージレジストリ()を使用しているときに、サイドカーイメージの名前とバージョンが誤って入力されるTrident演算子の問題を修正しました"問題#983"。
  - ONTAPフェイルオーバーのギブバック中にマルチパスセッションがリカバリできないという問題が修正されました（"問題#961"）。

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

拡張機能

- * Kubernetes *：
  - ONTAP ASA R2 for iSCSIのサポートが追加されました。
  - ノードの正常でないシャットダウン時のONTAP NASボリュームに対する強制的な接続解除のサポートが追加されました。新しいONTAP - NASボリュームで、Tridentで管理されるボリューム単位のエクスポートポリシーを使用す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、アンパブリッシュ時に既存のボリュームを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が提供されました。
  - cloneFromSnapshotアノテーションが追加されました。
  - ネームスペース間のボリュームクローニングのサポートが追加されました。

- 強化されたiSCSI自己回復スキャンの修正により、ホスト、チャネル、ターゲット、およびLUN IDを指定して再スキャンを開始します。
- Kubernetes 1.32のサポートを追加。

- * OpenShift *：
  - ROSAクラスタでのRHCOSの自動iSCSIノード準備のサポートが追加されました。
  - OpenShift Virtualization for ONTAPドライバのサポートが追加されました。
- ONTAP SANドライバでのファイバチャネルのサポートが追加されました。
- NVMe LUKSのサポートが追加されました。
- すべてのベースイメージのスクラッチイメージに切り替えました。
- iSCSIセッションはログインする必要があるが、ログインしない場合のiSCSI接続状態の検出とロギングが追加されました（"問題#961"）。
- google-cloud-smb-volumesドライバでNetAppボリュームのサポートが追加されました。
- 削除時にONTAPボリュームがリカバリキューをスキップできるようにするための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インストーラにimage-pull-secretsフラグを追加しました。

の修正

- * Kubernetes *：
  - 自動エクスポートポリシーにノードのIPアドレスがない問題を修正しました（"問題#965"）。
  - ONTAP - NAS - Economyでは、ボリュームポリシー単位に早めに切り替わる固定の自動エクスポートポリシー。
  - 使用可能なすべてのAWS ARNパーティションをサポートするように、バックエンドの設定クレデンシャルを修正しました（"問題#913"）。
  - 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が追加されました"問題#924"。
  - CSI-resizerコンテナ()のSecurityContextを追加しました"問題#976"。

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。

拡張機能

- 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が追加されました。

- AutoSupportプロキシ接続の設定のサポートが追加されました。
- Data Mover暗号化のシークレットを定義する機能（Kopia/Restic）が追加されました。
- 実行フックを手動で実行する機能が追加されました。
- 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。
- クラスタを対象としたリソースの除外を有効にする拡張ResourceFilter。
- S3 AppVaultクレデンシャルでのAWSセッショントークンのサポートが追加されました。
- プレスナップショット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。

の修正

- 一時ボリュームの管理が改善され、ONTAPボリュームリカバリキューがスキップされるようになりました。
- SCCのアノテーションが元の値にリストアされました。
- 並列処理のサポートにより、リストア効率が向上します。
- 大規模なアプリケーションの実行フック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

拡張機能

- * Kubernetes *：Kubernetes 1.32のサポートを追加。
- iSCSIセッションはログインする必要があるが、ログインしない場合のiSCSI接続状態の検出とロギングが追加されました（"問題#961"）。

の修正

- 自動エクスポートポリシーにノードのIPアドレスがない問題を修正しました（"問題#965"）。
- ONTAP - NAS - Economyでは、ボリュームポリシー単位に早めに切り替わる固定の自動エクスポートポリシー。
- TridentとTrident ASUPの依存関係を更新し、CVE-2024-45337およびCVE-2024-45310に対応。
- iSCSIの自己修復中に、一時的に正常でない非CHAPポータルのログアウトが削除されました（"問題#961"）。

24.10の変更点

拡張機能

- Google Cloud NetApp VolumesドライバがNFSボリュームに対して一般提供されるようになり、ゾーン対応のプロビジョニングがサポートされるようになりました。
- GCPワークロードIDは、GKEを使用するGoogle Cloud NetApp VolumeのCloud Identityとして使用されま

す。

- LUN-SAN ONTAPドライバおよびLUN-SAN-Economyドライバに設定パラメータが追加され、ユーザがONTAP形式オプションを指定できるようになりました `formatOptions`。
- Azure NetApp Filesの最小ボリュームサイズを50GiBに縮小Azureの新しい最小サイズは、11月に一般提供される予定です。
- ONTAP NAS-EconomyドライバとONTAP SAN-Economyドライバを既存のFlexVolプールに制限する設定パラメータが追加されました `denyNewVolumePools`。
- すべてのONTAPドライバで、SVMでアグリゲートの追加、削除、名前変更が検出されるようになりました。
- 報告された PVC サイズ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 -エコノミーノードステージとアンステージエラー処理が改善され、ステージが失敗した後にアンステージでデバイスを削除できるようになりました。
- カスタムロールジェネレータを追加しました。これにより、お客様はONTAPでTridentの最小限のロールを作成できます。
- トラブルシューティング用のロギングを追加 `lsscsi`（"問題#792"）。

Kubernetes

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

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

実験的な機能強化

- ONTAP SANドライバでのファイバチャネルサポートのテクニカルプレビューを追加。

の修正

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

非推奨

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

製品のブランド変更

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

24.06の変更点

拡張機能

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

- Google Cloud NetApp Volumeのテクニカルプレビューのサポートを追加。

Kubernetes

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

の修正

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

24.02の変更点

拡張機能

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

Kubernetes

- Kubernetes 1.29のサポートを追加。

の修正

- ACPが有効になっていない場合、ACPの警告メッセージが修正されました（"問題#866"）。
- クローンがスナップショットに関連付けられている場合、ONTAPドライバのスナップショット削除中にク

ローンスプリットを実行する前に10秒の遅延が追加されました。

非推奨

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

23.10の変更点

の修正

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

拡張機能

Kubernetes

Kubernetes 1.28のサポートを追加。

Trident

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

23.07.1の変更点

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

23.07の変更点

の修正

Kubernetes

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

Trident

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

拡張機能

Kubernetes

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

Trident

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

実験的強化

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

ドキュメント

組織とフォーマットの多くの改善が行われました。

非推奨

Kubernetes

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

23.04の変更点

ONTAP-SAN-*ボリュームの強制的なボリューム接続解除は、非グレースフルノードシャットダウン機能のゲートが有効になっているKubernetesバージョンでのみサポートされます。[Force

> detach]は、インストール時にを使用して有効にする必要があります `--enable-force-detach` Tridentインストーラのフラグ。

の修正

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

拡張機能

- Kubernetes：
  - Kubernetes 1.27のサポートを追加。
  - LUKSボリュームのインポートのサポートが追加されました。
  - ReadWriteOncePod PVCアクセスモードのサポートが追加されました。
  - ノードの正常でないシャットダウン時にONTAP-SAN-*ボリュームで強制的に接続解除がサポートされるようになりました。
  - すべてのontap-san-*ボリュームでノード単位のigroupを使用するようになりました。LUNはigroupにマッピングされるだけで、それらのノードにアクティブにパブリッシュされるため、セキュリティ体制が強化されます。アクティブなワークロードに影響を与えることなく既存のボリュームを安全であるとTridentが判断した場合、必要に応じて新しいigroupスキームに切り替えます（"問題 #758"）。
  - Tridentで管理されていないigroupをONTAP-SAN-*バックエンドからクリーンアップし、Tridentのセキュリティを強化
- ストレージドライバontap-nas-economyとontap-nas-flexgroupに、Amazon FSxによるSMBボリュームのサポートが追加されました。
- ontap-nas、ontap-nas-economy、ontap-nas-flexgroupストレージドライバでSMB共有のサポートが追加されました。
- arm64ノードのサポートを追加しました"問題 #732"）。
- 最初にAPIサーバを非アクティブ化することで、Tridentが手順 をシャットダウンできるようになりました"問題 #811"）。
- WindowsおよびarmXNUMXホストのクロスプラットフォームビルドサポートをMakefileに追加しました。build.mdを参照してください。

非推奨

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

23.01.1の変更点

の修正

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

23.01の変更点

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

の修正

- Kubernetes：Helm（"問題#783、#794"）。

拡張機能

Kubernetes

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

Trident

- ONTAP NASストレージドライバに、Amazon FSx for NetApp ONTAPを使用したSMBボリュームのサポートが追加されました。
- SMBボリュームの使用時のNTFS権限のサポートが追加されました。
- CVSサービスレベルを使用したGCPボリュームのストレージプールのサポートが追加されました。
- FlexGroupをONTAP-NAS-flexgroupストレージドライバで作成する際のflexgroupAggregateListのオプション使用がサポートされるようになりました。
- 複数のFlexVolボリュームを管理する場合、ONTAP NASエコノミーストレージドライバのパフォーマンスが向上

- すべてのONTAP NASストレージドライバに対してデータLIFの更新を有効にしました。
- Trident DeploymentとDemonSetの命名規則を更新し、ホストノードOSを反映させました。

非推奨

- Kubernetes：サポートされる最小Kubernetes数を1.21に更新
- ドライバまたは `ontap-san-economy`ドライバの設定時にDataLIFを指定しないようにし `ontap-san`ました。

22.10の変更

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

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

の修正

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

拡張機能

- Kubernetes：
  - Kubernetes 1.25のサポートを追加。Kubernetes 1.25にアップグレードする前に、Tridentを22.10にアップグレードする必要があります。
  - Trident Deployment and DemonSet用に別々のServiceAccount、ClusterRole、ClusterRoleBindingを追加して、今後の権限の強化を可能にしました。

- のサポートが追加されました "ネームスペース間ボリューム共有"。

- すべてTrident `ontap-*` ストレージドライバがONTAP REST APIで機能するようになりました。
- 新しい演算子YAMLを追加しました (`bundle_post_1_25.yaml`）を使用しない場合 `PodSecurityPolicy` Kubernetes 1.25をサポートするため。
- を追加しました "LUKS暗号化ボリュームをサポートします" の場合 `ontap-san` および `ontap-san-economy` ストレージドライバ。
- Windows Server 2019ノードのサポートが追加されました。
- を追加しました "WindowsノードでのSMBボリュームのサポート" を使用する `azure-netapp-files` ストレージドライバ。
- ONTAP ドライバの自動MetroCluster スイッチオーバー検出機能が一般提供されるようになりました。

非推奨

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

2007年22月の変更

の修正

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

拡張機能

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

非推奨

- **Kubernetes**：サポートされる最小Kubernetes数が1.19に更新されました。

- バックエンド構成では、単一の構成で複数の認証タイプを使用できなくなりました。

削除します

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

ドキュメント

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

2004年10月22日の変更

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

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

の修正

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

拡張機能

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

削除します

E シリーズドライバ（ 20.07 以降無効）が削除されました。

22.01.1 の変更

の修正

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

22.01.0 の変更

の修正

- * Kubernetes ：大規模なクラスタのノード登録バックオフ再試行時間を延長します。
- azure-NetApp-files ドライバが、同じ名前の複数のリソースによって混乱することがあるという解決済みの問題 。
- 角かっこで指定した場合にONTAP SAN IPv6データLIFが機能するようになりました。
- すでにインポートされているボリュームをインポートしようとすると、 EOF 問題 が返され、 PVC は保留状態になります。（"GitHub 問題 #489"）
- SolidFireボリュームでSnapshotが32個を超える場合にTridentのパフォーマンスが低下する問題が修正されました。
- SSL 証明書の作成時に SHA-1 を SHA-256 に置き換えました。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。
- リソース名の重複を許可し、操作を単一の場所に制限するためのAzure NetApp Filesドライバを修正しました。

拡張機能

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

非推奨

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

21.10.1 の変更点

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

の修正

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

21.10.0 の変更点

の修正

- XFS ボリュームのクローンをソースボリュームと同じノードにマウントできない固定問題（ GitHub 問題 514 ）
- Tridentがシャットダウン時に致命的なエラーを記録する問題を修正(GitHub Issue 597)。
- Kubernetes 関連の修正：
  - スナップショットを作成するときに 'ONTAP-NAS' および 'ONTAP-NAS-flexgroup ドライバ（ GitHub 問題 645 ）を使用して ' ボリュームの使用済み領域を最小リストアサイズとして返します
  - ボリュームのサイズ変更後に 'Failed to expand filesystem エラーがログに記録された問題を修正しました (GitHub 問題 560)
  - POD が「 Terminating 」状態で停止する可能性がある固定問題（ GitHub 問題 572 ）。
  - 「 ONTAP-SAN-エコノミー 」問題がスナップショット FlexVol （ GitHub 533 ）でいっぱいになる場合があるという問題を修正しました。
  - 異なるイメージを持つ固定カスタム YAML インストーラ問題（ GitHub 問題 613 ）。
  - Snapshot サイズの計算方法を固定（ GitHub 問題 611 ）。
  - すべてのTridentインストーラがプレーンなKubernetesをOpenShiftと識別できる問題を修正(GitHub Issue 639)。
  - Kubernetes API サーバにアクセスできない場合に、 Trident オペレータが更新を停止するよう修正しました（ GitHub 問題 599 ）。

拡張機能

- GCP - CVS パフォーマンスボリュームに対する「 unixPermissions 」オプションのサポートが追加されました。
- GCP でのスケール最適化 CVS ボリュームのサポートが 600GiB から 1TiB に追加されました。
- Kubernetes 関連の機能拡張：
  - Kubernetes 1.22 のサポートが追加されました。

- Trident の operator と Helm チャートを Kubernetes 1.22 （ GitHub 問題 628 ）と連携させるように設定
- tridentctl images コマンドに演算子イメージを追加 (GitHub 問題 570)

実験的な機能強化

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

既知の問題

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

- TridentがインストールされているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は `helm upgrade`、クラスタをアップグレードする前に、values.yamlをに `true`設定するかコマンドに追加する `--set excludePodSecurityPolicy=true`ように更新する必要があります。 `excludePodSecurityPolicy`
- StorageClassで指定した(`fsType=""`が含まれていないボリュームには、`Trident`によって空白が適用されるように `fsType`なりました `fsType`。Tridentでは、Kubernetes 1.17以降を使用する場合、NFSボリュームに空のを指定できます `fsType`。iSCSIボリュームの場合、セキュリティコンテキストの使用を適用するときは、StorageClassで `fsGroup`を設定する必要があります `fsType`。
- 複数のTridentインスタンスでバックエンドを使用する場合は、各バックエンド構成ファイルの値がONTAPバックエンドに対して異なるか、SolidFireバックエンドに対して異なる値を使用する `TenantName`必要があります `storagePrefix`。Tridentは、Tridentの他のインスタンスで作成されたボリュームを検出できません。ONTAPまたはSolidFireバックエンドに既存のボリュームを作成しようとすると成功します。これは、Tridentではボリューム作成が優先的な処理として処理されるためです。 `storagePrefix` `TenantName`同じバックエンドに作成されたボリュームで名前の競合が発生する可能性があります。
- Tridentをインストールし（またはTridentオペレータを使用）、を使用して `tridentctl` `Trident`を管理する場合は `tridentctl`、環境変数が設定されていることを確認する必要があります `KUBECONFIG`。これは、対象となるKubernetesクラスタを指定するために必要 `tridentctl`です。複数のKubernetes環境を使用する場合は、ファイルが正確にソースされていることを確認する必要があり `KUBECONFIG`ます。
- iSCSI PVS のオンラインスペース再生を実行するには、作業者ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これはRHEL/Red Hat Enterprise Linux CoreOS（RHCOS）インスタンスに当てはまります `discard` "マウントオプション"。オンラインブロック破棄をサポートするには、discard mountOptionが^]に含まれていることを確認してください。[`StorageClass`
- 各KubernetesクラスタにTridentのインスタンスが複数あると、Tridentは他のインスタンスと通信できず、そのインスタンスが作成した他のボリュームを検出できません。そのため、クラスタ内で複数のインスタンスを実行すると、予期しない誤った動作が発生します。KubernetesクラスタごとにTridentのインスタンスを1つだけ配置する必要があります。
- TridentがオフラインのときにTridentベースのオブジェクトがKubernetesから削除された場合、`StorageClass`Tridentはオンラインに戻っても対応するストレージクラスをデータベースから削除しません。これらのストレージクラスは、またはREST APIを使用して削除して `tridentctl`ください。
- ユーザが、対応するPVCを削除する前にTridentでプロビジョニングされたPVを削除しても、Tridentはバッキングボリュームを自動的に削除しません。またはREST APIを使用してボリュームを削除してください `tridentctl`。

- FlexGroup では、プロビジョニング要求ごとに一意のアグリゲートセットがないかぎり、同時に複数の ONTAP をプロビジョニングすることはできません。
- IPv6経由のTridentを使用する場合は、バックエンド定義でとを `dataLIF`角かっこで指定する必要があります `managementLIF`。たとえば、`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`です。

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

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

詳細については、こちらをご覧ください

- "Trident GitHub"
- "Tridentブログ"

以前のバージョンのドキュメント

Trident 26.10を実行していない場合は、以前のリリースのドキュメントが"Tridentのサポートライフサイクル"に基づいて利用できます。

- "Trident 26.06"
- "Trident 26.02"
- "Trident 25.10"
- "Trident25.06"
- "Trident 25.02"
- "Trident 24.10"
- "Trident 24.06"
- "Trident 24.02"
- "Trident 23.10"
- "Trident 23.07"
- "Trident 23.04"
- "Trident 23.01"

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

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

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

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

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

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

- Trident 25.02以降はiSCSIをサポートします。
- Trident 25.06以降では、iSCSIに加えてNVMe/TCPもサポートされます。
- Trident 25.10以降では、iSCSIおよびNVMe/TCPに加えてFCもサポートされています。

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

サポートされている操作

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

サポートされていない操作

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

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

既知の問題

ここでは、このリリースの製品の正常な使用に支障をきたす可能性がある既知の問題について説明します。

現在のリリースに影響する既知の問題は次のとおりです。

Trident 22.10以前からのアップグレード後もボリュームは接続されたままになる

この問題は、ボリュームがアタッチされている状態で、Trident 22.10以前からTrident 26.10へ直接アップグレードするお客様に影響します。Trident 23.01以降からのアップグレードは影響を受けません。

アップグレード後、影響を受けるボリュームがデタッチされない場合があります。ReadWriteOnce（RWO）ボリュームを使用するPodは別のノードに移動できず、マルチアタッチエラーを報告する可能性があります。PersistentVolume（PV）の削除もハングする可能性があります。

この問題を回避するには、古いボリューム接続が存在するノードで、そのボリュームの `globalmount`ディレクトリから残留している `volumePublishInfo.json`ファイルを削除してください。

VolumeSnapshotsがReadyToUse状態に到達しない

> `csi-snapshotter`サイドカーには既知の問題があります。すべてのKubernetesバージョンにおいて、VolumeGroupSnapshots `v1beta1`がVolumeSnapshotsの `ReadyToUse`状態への到達を防ぎます。

回避策は 2 つあります：

1. VolumeGroupSnapshots CRDを削除してVolumeGroupSnapshotsを無効にし、Tridentを再インストールします。
2. VolumeGroupSnapshots `v1beta2`および `snapshot-controller`バージョン8.4.0以降をインストールしてから、Tridentを再インストールしてください。VolumeGroupSnapshotsはKubernetesバージョンv1.34より前では機能しません。

大容量ファイルのResticバックアップのリストアが失敗することがある

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。

はじめに

Tridentの詳細

Tridentの詳細

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

Tridentとは

NetApp Tridentは、パブリッククラウドまたはオンプレミスにおける、すべての主要なNetAppストレージプラットフォーム全体でのストレージリソースの消費と管理を可能にします。これには、オンプレミスONTAPクラスタ（AFF、FAS、ASA）、ONTAP Select、Cloud Volumes ONTAP、Elementソフトウェア（NetApp HCI、SolidFire）、Azure NetApp Files、Amazon FSx for NetApp ONTAP、Google Cloud NetApp Volumesが含まれます。

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

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

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

サポートされている **Kubernetes** プラットフォーム

Tridentは、さまざまなKubernetesディストリビューションおよびプラットフォームでサポートされています。

サポートされているプラットフォームは以下のとおりです。* アップストリーム Kubernetes * Red Hat OpenShift * SUSE Harvester 1.7.0 (ONTAP iSCSI)

KubernetesとNetApp製品の統合

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

▼ **NetApp ONTAP** 対応の **Amazon FSX**

"NetApp ONTAP 対応の Amazon FSX" は、NetApp ONTAPストレージオペレーティングシステムを基盤とするファイルシステムを起動して実行できる、フルマネージドのAWSサービスです。

▼ **Azure NetApp Files** の特長

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

▼ **Cloud Volumes ONTAP**

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

▼ **Google Cloud NetApp**ボリューム

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

▼ **Element** ソフトウェア

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

▼ **NetApp HCI**

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

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

▼ **NetApp ONTAP**

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

ONTAPシステムは、オールフラッシュ、ハイブリッド、オールHDD構成で構成され、オンプレミスのFAS、AFA、ASAクラスタ、ONTAP Select、Cloud Volumes ONTAPなど、さまざまな導入モデルを提供します。Tridentは、次のONTAP導入モデルをサポートしています。

Tridentのアーキテクチャ

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

コントローラポッドとノードポッドについて

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

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

- コントローラプラグインは、Snapshotやサイズ変更などのボリュームのプロビジョニングと管理を処理します。

- ノードプラグインによって、ノードへのストレージの接続が処理されます。

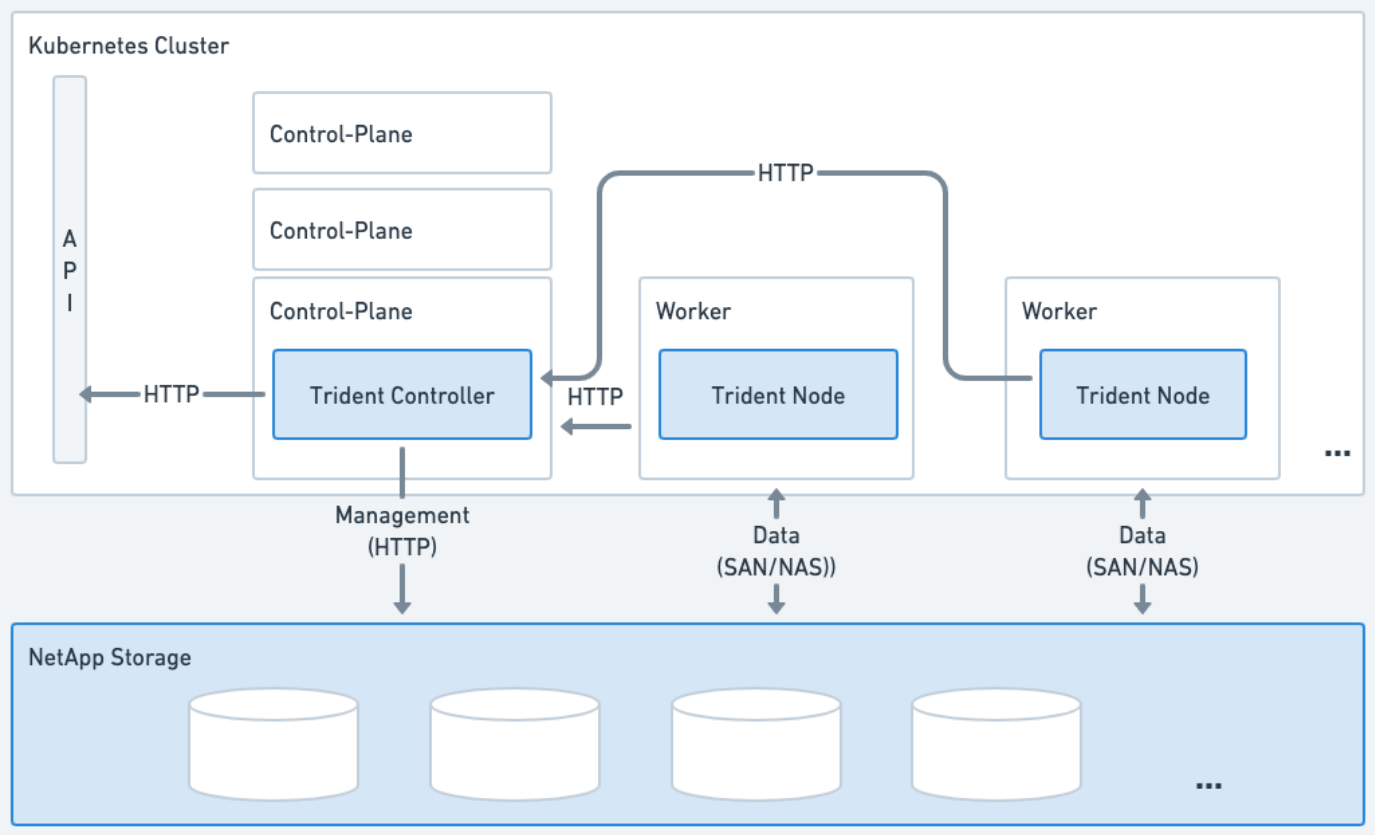


図 **1. Kubernetes**クラスタに導入された**Trident**

Tridentコントローラポッド

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

- NetAppストレージ内のボリュームのプロビジョニングと管理を担当
- Kubernetes環境で管理
- インストールパラメータに応じて、コントロールプレーンノードまたはワーカーノードで実行できます。

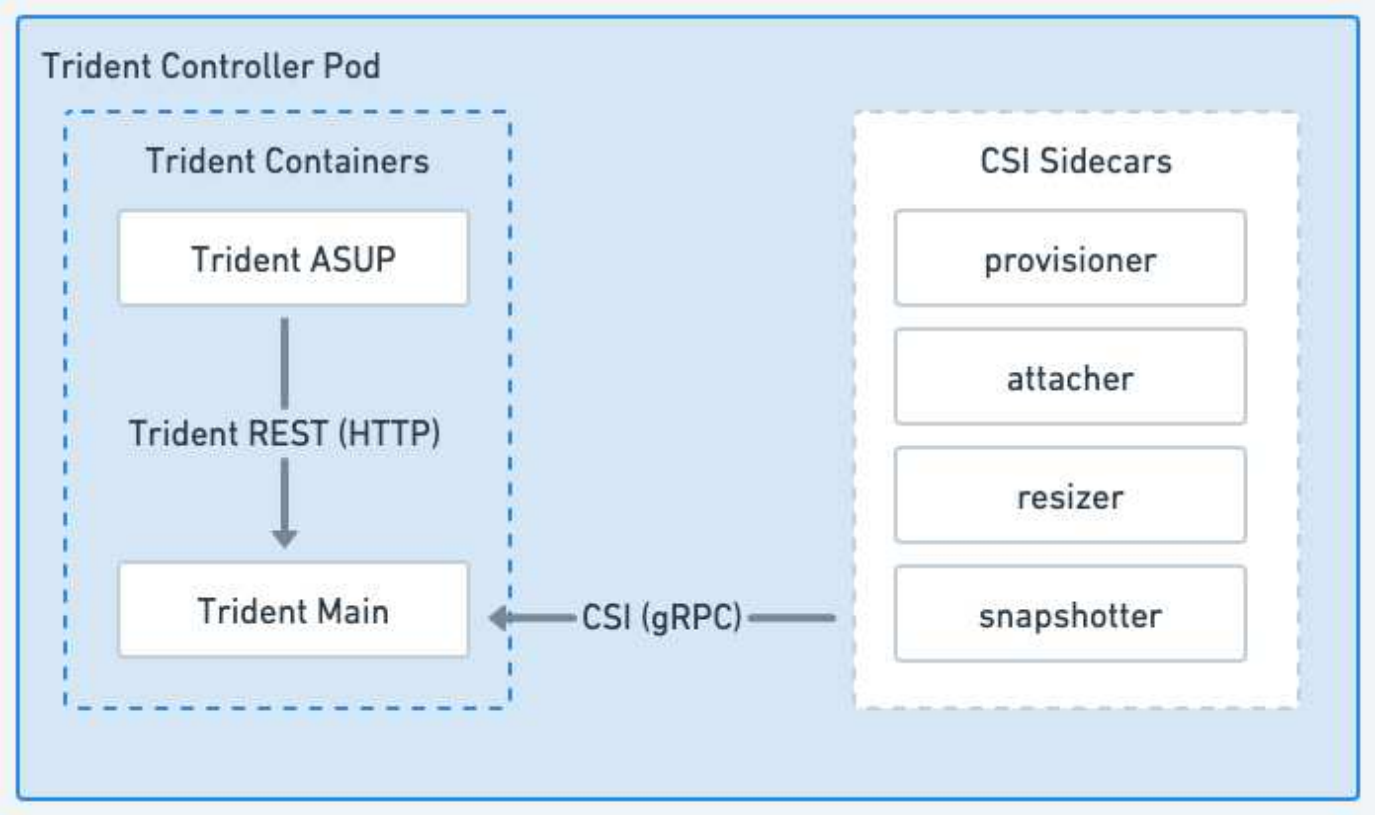


図 **2. Trident**コントローラポッドの図

Tridentノードポッド

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

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

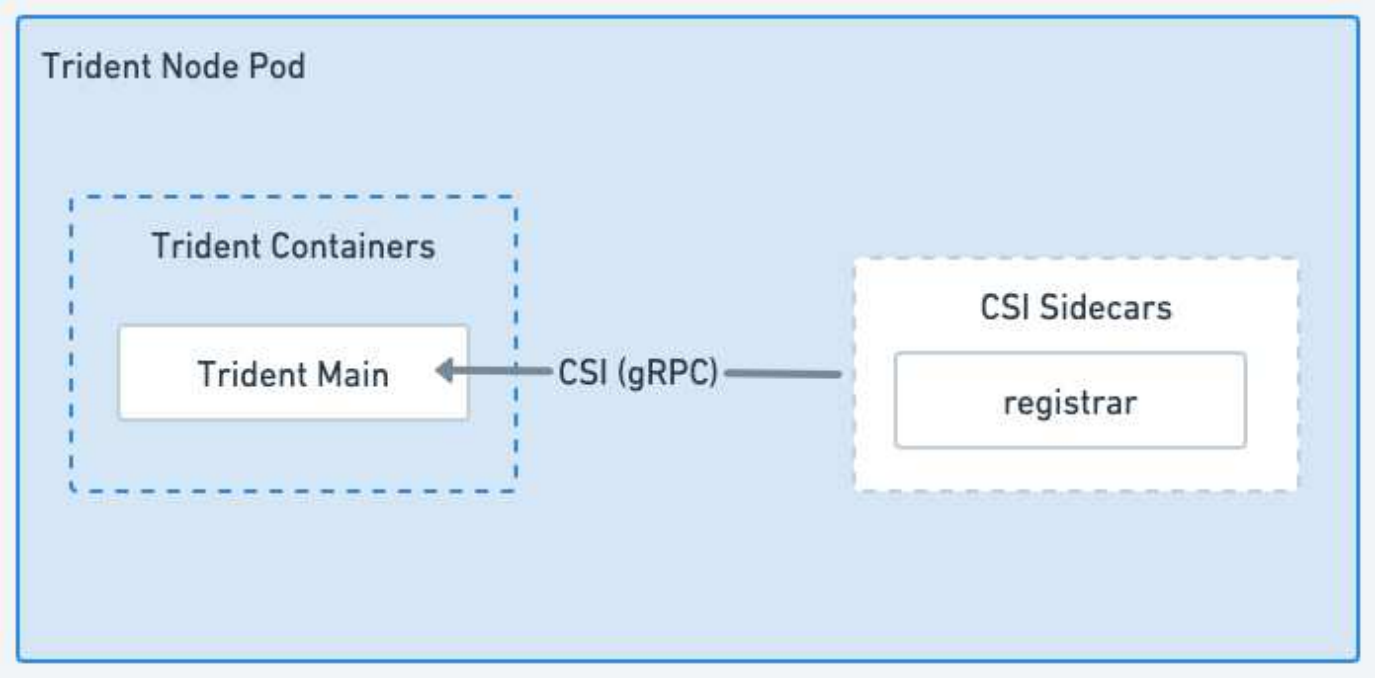


図 **3. Trident**ノードのポッド図

サポートされる **Kubernetes** クラスタアーキテクチャ

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

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

概念

Tridentボリュームプロビジョニングプロセス

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

ストレージクラスの関連付け

バックエンドストレージプールをストレージクラスに関連付けるには、ストレージクラスの要求された属性と、 `additionalStoragePools`の `excludeStoragePools`リストの両方が `storagePools`必要です。ストレージクラスを作成すると、 `Trident` はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。ストレージプールの属性および名前が要求されたす

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

ボリュームの作成

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

Tridentボリュームスナップショットの作成

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

ボリューム**Snapshot**の作成方法について説明します

- のために ontap-nas、 ontap-san 、 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。
- をクリックします ontap-nas-flexgroup ドライバ、各永続ボリューム（PV）はFlexGroup にマッピングされます。その結果、ボリューム Snapshot は NetApp FlexGroup Snapshot として作成されます。NetAppのスナップショット・テクノロジーは'競合するスナップショット・テクノロジーよりも安定性'拡張性'リカバリ性'パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- ドライバの場合 ontap-san-economy、PVSの共有FlexVolボリューム上に作成されたLUNへのPVSマッピングは、関連付けられたLUNのFlexCloneを実行することで実現されます。ONTAP FlexCloneテクノロジを使用すると、大規模なデータセットのコピーをほぼ瞬時に作成できます。コピーと親でデータブロックが共有されるため、メタデータに必要な分しかストレージは消費されません。
- 「 olidfire-SAN' ドライバの場合、各 PV は NetApp Element ソフトウェア / NetApp HCI クラスタ上に作成された LUN にマッピングされます。ボリューム Snapshot は、基盤となる LUN の Element Snapshot で表されます。これらの Snapshot はポイントインタイムコピーであり、消費するシステムリソースとスペースはごくわずかです。
- ドライバと `ontap-san`ドライバを使用する場合、 `ontap-nas`ONTAPスナップショットはFlexVolのポイントインタイムコピーであり、FlexVol自体のスペースを消費します。その結果、ボリューム内の書き込み可能なスペースが、 Snapshot の作成やスケジュール設定にかかる時間を短縮できます。この問題に対処する簡単な方法の 1 つは、 Kubernetes を使用してサイズを変更することでボリュームを拡張することです。もう 1 つの方法は、不要になった Snapshot を削除することです。Kubernetesで作成されたボリュー

ムSnapshotを削除すると、Tridentは関連付けられているONTAP Snapshotを削除します。Kubernetes で作成されていない ONTAP スナップショットも削除できます。

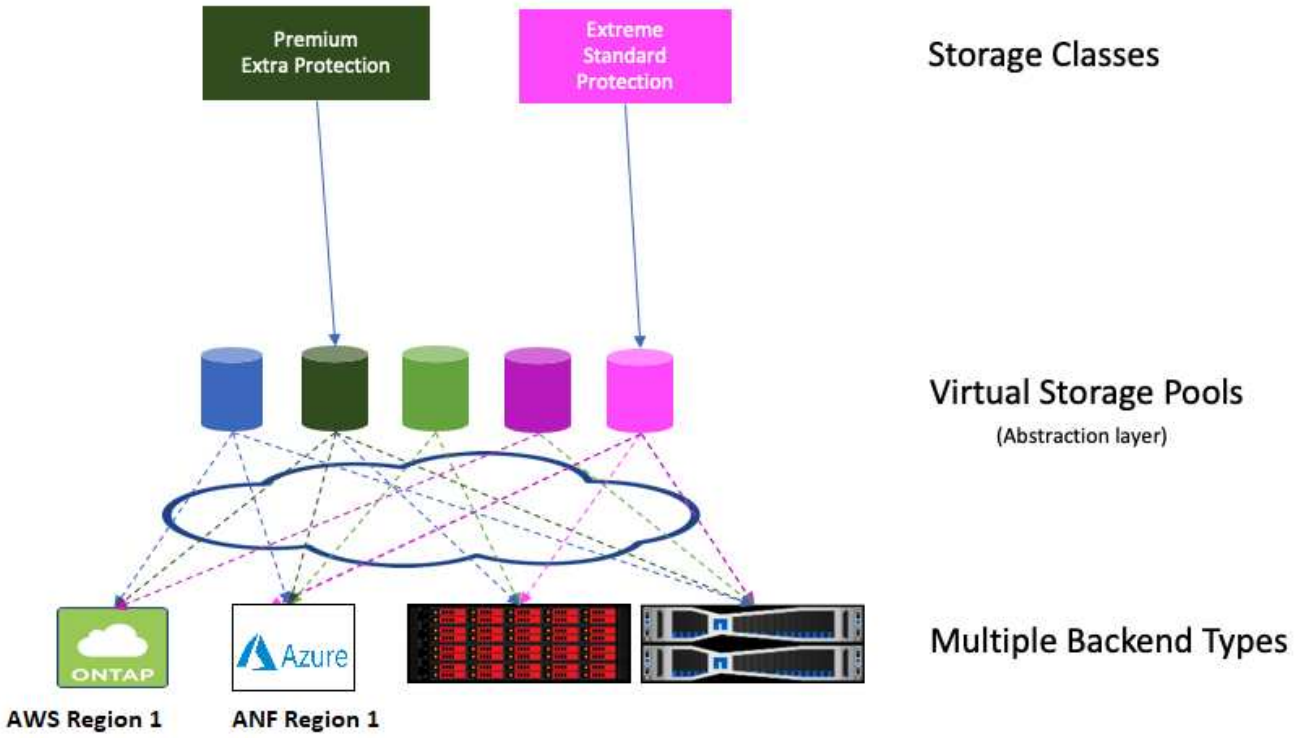
Tridentでは、VolumeSnapshots を使用してそこから新しい PV を作成できます。これらのスナップショットからの PV の作成は、サポートされているONTAPバックエンドのFlexCloneテクノロジーを使用して実行されます。スナップショットから PV を作成する場合、バックアップ ボリュームはスナップショットの親ボリュームのFlexCloneになります。その `solidfire-san`ドライバーは、Element ソフトウェア ボリューム クローンを使用してスナップショットから PV を作成します。ここでは、Element スナップショットからクローンを作成します。

Trident仮想ストレージプールの構成

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

仮想プールについて説明します

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



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

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

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

仮想プールのラベルは次の文字を使用して定義できます。

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

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

演算子	例	プールのラベル値は次のとおりです。
=	パフォーマンス = プレミアム	一致
`!= `	パフォーマンス！ = 非常に優れています	一致しません
「 in`	場所（東部、西部）	値のセットに含まれています
「 notin 」	パフォーマンス記名（シルバー、ブロンズ）	値のセットに含まれていません
`<key>`	保護	任意の値で存在します
`!<key>`	!保護	存在しません

Tridentボリュームアクセスグループ管理

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

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

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

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

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

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

クイックスタートガイド（Trident）

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

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

1 ワーカーノードの準備

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

"ワーカーノードを準備します"

2 Tridentのインストール

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

"Trident をインストール"

3 バックエンドを作成します

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

"バックエンドの設定" ストレージシステム

4 **Kubernetes**ストレージクラスの作成

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

"ストレージクラスを作成する。"

5 ボリュームをプロビジョニングする

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

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

"ボリュームをプロビジョニングする"

次の手順

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

Tridentの要件

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

Tridentに関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ **Trident **に関する****の重要な情報

- Trident 26.10はKubernetes 1.27から1.37までをサポートしています。Kubernetesをアップグレードする前に、以下のバージョン固有のイメージリストを確認してください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

サポートされるフロントエンド（オーケストレーションツール）

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

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.16）
- Kubernetes 1.27 - 1.37
- OpenShift 4.12、4.14 - 4.22（iSCSIノードの準備をOpenShift 4.19で使用する場合、サポートされる最小Tridentバージョンは25.06.1です。）

> Tridentは、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）v1.28.x - 1.37.x
- NFSv4、iSCSI、およびNVMe/TCPに対応したTalos OS。各プロトコルには、以下のTalosシステム拡張機能が必要です。これらの要件については、Sidero Labsにご確認ください。
  - NFSv4： `siderolabs/nfs-utils`
  - iSCSI： `siderolabs/iscsi-tools`、 `siderolabs/multipath-tools`、および `siderolabs/trident-iscsi-tools`
  - NVMe/TCP： `siderolabs/nvme-cli`、 `siderolabs/util-linux-tools`、および `siderolabs/trident-iscsi-tools`

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

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

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

サポートされるバックエンド（ストレージ）

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

- NetApp ONTAP 対応の Amazon FSX
- Azure NetApp Files の特長
- Cloud Volumes ONTAP
- Google Cloud NetAppボリューム
- ネットアップオール SAN アレイ（ ASA ）
- NetApp の完全サポートまたは限定サポートの下でONTAPバージョンを実行しているオンプレミスの FAS、 AFF、またはASA r2 (iSCSI、NVMe/TCP、および FC)。見る "ソフトウェア バージョンのサポート"。

- NetApp HCI / Elementソフトウェア11以降

TridentによるKubeVirtとOpenShiftによる仮想化のサポート

サポートされるストレージドライバ：

Tridentは、KubeVirtおよびOpenShift仮想化用に次のONTAPドライバをサポートしています。

- ONTAP - NAS
- SAN-SAN（iSCSI、FCP、ONTAP over TCP）
- ONTAP SANエコノミー（iSCSIのみ）

考慮すべきポイント：

- OpenShift仮想化環境でストレージクラスを更新し、パラメータ（例： `fsType: "ext4"`）を使用 `fsType`します。必要に応じて、のパラメータを `dataVolumeTemplates`使用してブロックデータボリュームの作成をCDIに通知し、ボリュームモードを明示的にブロックするように設定し `volumeMode=Block`ます。
- ブロックストレージドライバの_rwxアクセスモード_：ONTAP SAN（iSCSI、NVMe/TCP、FC）およびONTAP SANエコノミー（iSCSI）ドライバは、「volumeMode：Block」（rawデバイス）でのみサポートされます。これらのドライバでは `fstype`、ボリュームはrawデバイスモードで提供されるため、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マイグレーションワークフローでは、次の組み合わせがサポートされます。
  - 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）

Kubernetes および OpenShift ホスト型コントロールプレーン上の Kubernetes に対する Trident サポート

Tridentは、Kubernetes on Kubernetesトポロジを完全に検証しています。インフラストラクチャ（ホスト）クラスタ上でノードがKubeVirt仮想マシンとして実行されるKubernetesクラスタ。このトポロジは、OpenShift Virtualization上のOpenShiftホスト型コントロール プレーン（HCP）などのホスト型コントロール プレーン製品の基盤となります。

> この構成は以前のリリースでは動作していましたが、完全な動作確認はバージョン26.06から開始されました。

ゲスト（テナント）クラスターは、KubeVirt CSI ドライバー（(`kubevirt-csi`）をストレージプロバイダーとして使用します。ゲストクラスター内のワークロードが `PersistentVolumeClaim`を要求すると、KubeVirt CSI ドライバーはインフラストラクチャクラスター内に対応するボリュームを作成します。Trident は ONTAP からボリュームをプロビジョニングし、そのボリュームはゲストノードの VM にホットプラグされます。Trident はインフラストラクチャクラスター内でのみ動作します。ゲストクラスターは、KubeVirt CSI ドライバーを介して ONTAP ストレージに間接的にアクセスします。

要件：

- Tridentは、サポートされているONTAPバックエンドを使用したインフラストラクチャクラスターにインストールされます。
- Trident ストレージクラスがインフラストラクチャクラスタに存在し、KubeVirt CSI ドライバがゲストクラスタボリュームをマッピングできるようになっています。
- KubeVirtまたはOpenShift Virtualizationはインフラストラクチャクラスターに展開され、KubeVirt CSIドライバーは各ゲストクラスターにインストールされます。

コンポーネント	バージョン
Trident	26.10
Kubernetes（インフラストラクチャおよびゲストクラスタ）	1.37
KubeVirt	1.9.0
コンテナ化データインポーター（CDI）	1.65

考慮すべきポイント：

- ゲストクラスタでは、ボリュームはゲストノードの仮想マシン上でホットプラグされたブロックデバイスとして表示されます。
- KubeVirt CSI ドライバーのワークフローでは、ゲストクラスター内に Trident は必要ありません。インフラストラクチャクラスター内で ONTAP バックエンドおよびストレージクラスを設定します。

Trident をゲストクラスタに直接インストールする：

Trident をゲスト（テナント）クラスターに直接インストールすることもできます。スナップショット、クローン、ボリュームのサイズ変更、インポートなど、Trident のフル機能セットが必要な場合は、ゲストクラスターに Trident をインストールしてください。これらの機能は `kubevirt-csi`単独では提供されません。

ゲストクラスタでは、NFS、iSCSI、NVMe/TCPといったIPベースのプロトコルを使用します。ゲストノードは直接的なファイバーチャネル（FC）接続を持たない仮想マシンであるため、ゲストクラスタはFCをサポートしていません。

機能の要件

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

フィーチャー（ **Feature** ）	**Kubernetes** のバージョン	フィーチャーゲートが必要ですか？
Trident	1.27 - 1.37	いいえ
ボリューム Snapshot	1.27 - 1.37	いいえ
ボリューム Snapshot からの PVC	1.27 - 1.37	いいえ
iSCSI PV のサイズ変更	1.27 - 1.37	いいえ
ONTAP 双方向 CHAP	1.27 - 1.37	いいえ
動的エクスポートポリシー	1.27 - 1.37	いいえ
Trident のオペレータ	1.27 - 1.37	いいえ
CSI トポロジ	1.27 - 1.37	いいえ

テスト済みのホストオペレーティングシステム

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

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

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

ホストの設定

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

"ワーカーノードを準備します"

ストレージシステムの構成：

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

"バックエンドを設定"

Tridentポート

Tridentでは、通信のために特定のポートにアクセスする必要があります。

"Tridentポート"

コンテナイメージと対応する **Kubernetes** バージョン

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

以下のTrident 26.10イメージリストには、Kubernetes 1.37のイメージが含まれています。

Trident 26.10.0 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、v1.35.0、v1.36.0、v1.37.0	• docker.io/netapp/trident:26.10.0 • docker.io/netapp/trident-autosupport:26.10 • docker.io/netapp/trident-operator:26.10.0（オプション） • registry.k8s.io/sig-storage/csi-attacher:v4.11.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.17.0 • registry.k8s.io/sig-storage/csi-provisioner:v6.3.0 • registry.k8s.io/sig-storage/csi-resizer:v2.1.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.6.0

Trident 26.06.2 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、v1.35.0、v1.36.0	• docker.io/netapp/trident:26.06.2 • docker.io/netapp/trident-autosupport:26.06 • docker.io/netapp/trident-operator:26.06.2（オプション） • registry.k8s.io/sig-storage/csi-attacher:v4.11.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.18.0 • registry.k8s.io/sig-storage/csi-provisioner:v6.3.0 • registry.k8s.io/sig-storage/csi-resizer:v2.1.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.6.0

Trident をインストール

Tridentのインストールについて

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

Trident 26.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ **Trident に関するの重要な情報**

- Kubernetes 1.37 が Trident でサポートされるようになりました。Kubernetes をアップグレードする前に、Trident をアップグレードしてください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

作業を開始する前に

インストールパスに関係なく、次のものが必要です。

- サポートされているバージョンの Kubernetes を実行し、機能要件が有効になっている、サポートされている Kubernetes クラスターに対する完全な権限。詳細については、"要件"を参照してください。
- サポートされているネットアップストレージシステムへのアクセス。
- Kubernetesワーカーノードすべてからボリュームをマウントできます。
- を搭載したLinuxホスト `kubectl` （または `oc`OpenShiftを使用している場合）Kubernetesクラスタを管理するようにインストールおよび設定します。
- 。 `KUBECONFIG` Kubernetesクラスタ構成を参照するように設定された環境変数。
- Kubernetes と Docker Enterprise を併用する場合は、 "CLI へのアクセスを有効にする手順は、ユーザが行ってください"。
- クラスタが特権ワークロードをサポートしている必要があります。

に慣れていない場合は"基本概念"、今すぐそれを行うのに最適な時期です。

インストール方法を選択します

適切なインストール方法を選択してください。決定を下す前に"メソッド間を移動しています"の考慮事項も確認する必要があります。

Trident演算子を使用する

手動でのデプロイでもHelmを使用する場合でも、Tridentオペレーターはインストールを簡素化し、Tridentリソースを動的に管理する優れた方法です。さらに、"Tridentのオペレータ環境をカスタマイズ"は、`TridentOrchestrator`カスタムリソース（CR）の属性を使用して行うこともできます。

Tridentオペレータには次のようなメリットがあります。

▼ ** Tridentオブジェクトの作成**

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

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

▼ ****リソースアカウンタビリティ****

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

▼ **** 自己回復機能****

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

▼ **** は、インストール済みの既存の**** を簡単に更新できます

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

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

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

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

▼ **クリーン再インストール**

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

▼ ** Kubernetesの自動アップグレード処理**

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

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

を使用します **`tridentctl`**

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

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

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

インストールモードを選択します

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

標準インストール

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

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

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

オフラインインストール

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

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

リモートインストール

次に、リモートインストールプロセスの概要を示します。

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

メソッドとモードに基づいてプロセスを選択します

決定が終わったら、適切なプロセスを選択します。

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

インストール方法を切り替える

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

- Tridentのインストールとアンインストールには、常に同じ方法を使用してください。を使用してを展開した場合は `tridentctl`、適切なバージョンのバイナリを使用してTridentをアンインストールする必要があります `tridentctl`。同様に、オペレータを使用して展開する場合は、CRを編集し、Tridentをアンインストールするように設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。
- オペレータベースの導入環境を削除してTridentの導入に使用する場合 `tridentctl`は、まずTridentを編集してからアンインストールするように設定する `spec.uninstall=true`必要があります `TridentOrchestrator`。次に、とオペレータの配置を削除し `TridentOrchestrator`ます。その後、を使用してをインストールできます `tridentctl`。
- 手動のオペレーターベースのデプロイメントがあり、Helm ベースのTridentオペレーター デプロイメントを使用する場合は、最初にオペレーターを手動でアンインストールしてから、Helm インストールを実行する必要があります。これにより、Helm は必要なラベルとアノテーションを使用してTridentオペレーターをデプロイできるようになります。これを行わないと、Helm ベースのTridentオペレーターのデプロイメントは、ラベル検証エラーとアノテーション検証エラーで失敗します。
- もしあなたが `tridentctl`ベースのデプロイメントでは、 Tridentをアンインストールせずに Helm ベースまたは Operator ベースのデプロイメントを実行できます。

その他の既知の設定オプション

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

- --kubbelet-dir` フラグは kubelet ディレクトリの場所に設定する必要があります。デフォルトでは、これは /var/vcap/data/kubelet` です。

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

Tridentオペレータを使用してインストール

Tridentオペレータを手動で導入（標準モード）

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

Trident 26.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ Trident に関するの重要な情報

- Trident 26.10はKubernetes 1.27から1.37までをサポートしています。Kubernetesをアップグレードする前に、"Tridentの要件"でバージョン固有のイメージリストを確認してください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

Tridentオペレータを手動で導入し、Tridentをインストール

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

作業を開始する前に

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

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

▼ Details

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v26.10.0/trident-
installer-26.10.0.tar.gz
tar -xf trident-installer-26.10.0.tar.gz
cd trident-installer
```

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

作成する `TridentOrchestrator`カスタム リソース定義 (CRD)。あなたは `TridentOrchestrator`後でカスタムリソースを作成します。適切なCRD YAMLバージョンを使用する `deploy/crds`を作成する `TridentOrchestrator`CRD。

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

手順**3**：**Trident**のオペレータを導入する

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

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

作業を開始する前に

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

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

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

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

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

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

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

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

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

```
kubectl describe torc trident

Name:         trident
Namespace:
Labels:       <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
  nodePrep:
  - iscsi
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:26.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:26.10.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v26.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します **TridentOrchestrator** ステータス

のステータス `TridentOrchestrator` インストールが正常に完了したかどうかを示し、インストールされているTridentのバージョンが表示されます。インストール中、のステータス `TridentOrchestrator` からの変

更 `Installing` 終了： `Installed`。を確認した場合は `Failed` ステータスとオペレータは単独で回復できません。 "ログをチェックしてください"。

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します **`tridentctl`**

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.0        | 26.10.0        |
+----------------+----------------+
```

Tridentオペレータを手動で導入（オフラインモード）

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

Tridentに関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ **Trident に関するの重要な情報**

- Trident 26.10はKubernetes 1.27から1.37までをサポートしています。Kubernetesをアップグレードする前に、"Tridentの要件"でバージョン固有のイメージリストを確認してください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

Tridentオペレータを手動で導入し、Tridentをインストール

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

作業を開始する前に

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

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

▼ **Details**

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v26.10.0/trident-installer-26.10.0.tar.gz
tar -xf trident-installer-26.10.0.tar.gz
cd trident-installer
```

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

作成する `TridentOrchestrator`カスタム リソース定義 (CRD)。あなたは `TridentOrchestrator`カスタムリソースは後で。適切なCRD YAMLバージョンを使用する `deploy/crds`を作成する `TridentOrchestrator`CRD:

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

手順**3**：オペレータのレジストリの場所を更新します

`/deploy/operator.yaml`で、 `image: docker.io/netapp/trident-operator:26.10.0`を更新して、イメージレジストリの場所を反映します。link:../trident-get-started/requirements.html#container-images-and-corresponding-kubernetes-versions["TridentとCSIの画像"]は1つのレジストリまたは異なるレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。次に例を示します。

- `image: <your-registry>/trident-operator:26.10.0`すべてのイメージが 1 つのレジストリにある場合。
- `image: <your-registry>/netapp/trident-operator:26.10.0` Trident イメージが CSI イメージ

とは異なるレジストリにある場合。

ステップ**4**：**Trident**オペレータを導入

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

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

作業を開始する前に

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

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

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

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

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

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

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

手順

1. リソースを作成し、オペレータを配置します。

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

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

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

Kubernetes クラスタには、オペレータのインスタンスが * 1 つしか存在しないようにしてくだ

さい。Trident のオペレータが複数の環境を構築することは避けてください。

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

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

1つのレジストリ内のイメージ

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.10"
tridentImage: "<your-registry>/trident:26.10.0"
```

異なるレジストリ内の画像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.10"
tridentImage: "<your-registry>/trident:26.10.0"
```

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

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

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

kubectl describe torc trident

Name:         trident
Namespace:
Labels:       <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/trident-autosupport:26.10
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:26.10.0
```

```
Status:
  Current Installation Params:
    IPv6:                        false
    Autosupport Hostname:
    Autosupport Image:           <your-registry>/trident-autosupport:26.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                       true
    Http Request Timeout:        90s
    Image Pull Secrets:
    Image Registry:        <your-registry>
    k8sTimeout:            30
    Kubelet Dir:           /var/lib/kubelet
    Log Format:            text
    Probe Port:            17546
    Silence Autosupport:   false
    Trident Image:         <your-registry>/trident:26.10.0
  Message:                 Trident installed
  Namespace:               trident
  Status:                  Installed
  Version:                 v26.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

インストールを確認します。

インストールを確認するには、いくつかの方法があります。

を使用します `TridentOrchestrator` ステータス

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

ステータス	説明
インストール中です	オペレータはこのCRを使用してTridentをインストールしています `TridentOrchestrator`。
インストール済み	Tridentは正常にインストールされました。
アンインストール中です	オペレータはTridentをアンインストールしています。 `spec.uninstall=true`
アンインストール済み	Tridentがアンインストールされます。

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

を使用します **`tridentctl`**

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.0        | 26.10.0        |
+----------------+----------------+
```

Helm（標準モード）を使用してTridentを導入

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

Trident 26.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ **Trident **に関する****の重要な情報

- Trident 26.10はKubernetes 1.27から1.37までをサポートしています。Kubernetesをアップグレードする前に、"Tridentの要件"でバージョン固有のイメージリストを確認してください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

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

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

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

作業を開始する前に

"導入の前提条件"に加えて"Helm バージョン 3"が必要です。

手順

1. Trident Helmリポジトリを追加します。

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

2. `helm install`を使用して、次の例のようにデプロイメントの名前を指定します。ここで、 `100.2610.0`はインストールするTridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace <trident-namespace>
```

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

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、 `debug`のデフォルト値を変更するには、次のコマンドを実行します。ここで、 `100.2610.0`はインストールするTridentのバージョンです：

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace trident --set tridentDebug=true
```

設定オプション

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

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

オプション	説明	デフォルト

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

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

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

オプション	説明	デフォルト
tridentExcludeAutosupport	true に設定した場合、Tridentはインストール中にAutoSupportコンテナをセットアップしません。	「偽」
tridentAutosupportImageTag	Trident AutoSupportコンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Trident AutoSupportコンテナがHTTPプロキシ経由で自宅に電話できるようにします。	""
tridentLogFormat	Tridentロギング形式を設定し(text`ます。または `json)	"text"
tridentDisableAuditLog	Trident監査ロガーをディセーブルにします。	「真」
tridentLogLevel	Tridentのログレベルを、、 debug info、、 warn、 `error`または `fatal`に設定 `trace`できます。	"info"
tridentDebug	Tridentのログレベルを次のように設定できます。 debug 。ノード ヘルス チェック (NHC) オペレーターとの統合により、強制デタッチ プロセスを自動化できます。詳細については、"Tridentによるステートフル アプリケーションのフェイルオーバーの自動化" 。	「偽」
tridentLogWorkflows	特定のTridentワークフローのトレースロギングまたはログ抑制を有効にできます。	""
tridentLogLayers	トレースロギングまたはログ抑制に対して特定のTridentレイヤをイネーブルにできます。	""
「 tridentImage 」のように入力します	Tridentのイメージを完全に上書きできます。	""
tridentImageTag	Tridentのイメージのタグを上書きできます。	""
tridentProbePort	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
windows	TridentをWindowsワーカーノードにインストールできるようにします。	「偽」
enableForceDetach	強制切り離し機能を有効にできます。	「偽」

オプション	説明	デフォルト
`enableDataLIFRefresh`	LIFが追加、削除、または新しいアドレスが割り当てられた際に、公開ボリュームのNVMeデータLIFを更新します。環境：ユニファイドONTAP上の `ontap-san`（`sanType: nvme`を使用）およびASA r2上のNVMe/TCP。詳細については、"NVMeデータLIFを更新する"を参照してください。	「偽」
`excludePodSecurityPolicy`	オペレータポッドのセキュリティポリシーを作成から除外します。	「偽」
`cloudProvider`	AKSクラスターでマネージドIDまたはクラウドIDを使用する場合は`"Azure"`に、EKSクラスターでクラウドIDを使用する場合は `"AWS"`に、GKEでクラウドIDを使用する場合やOpenShiftでWorkload Identity Federationを使用する場合は `"GCP"`に設定します。	""
`cloudIdentity`	AKS クラスターでクラウド ID を使用する場合は、ワークロード ID（"azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"）に設定してください。EKS クラスターでクラウド ID を使用する場合は、AWS IAM ロール（"'eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role'"）に設定してください。GKE でクラウド ID を使用する場合は、GKEクラウド ID アノテーションに設定してください。OpenShift Workload Identity Federation には設定しないでください。	""
`gcpWIFAudience`	OpenShift 上の Google Cloud Workload Identity Federation で使用される投影されたサービスアカウントトークンのオーディエンス。プロバイダの正規リソース名を使用してください。詳細については、"Workload Identity フェデレーションを使用して Google Cloud NetApp Volumes を設定する" を参照してください。	""
`iscsiSelfHealingInterval`	iSCSIの自己修復が実行される間隔。	`5m0s`

オプション	説明	デフォルト
`iscsiSelfHealingWaitTime`	iSCSIの自己修復が、ログアウトとその後のログインを実行して古いセッションの解決を開始するまでの時間。	`7m0s`
`nodePrep`	指定されたデータ 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、 Tridentが 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。*注意: 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。	
`enableConcurrency`	スループット向上のために Trident コントローラの同時操作を有効にします。サポートされているドライバーの詳細については、リンク： Controller Scalability documentation を参照してください。	いいえ
`k8sAPIQPS`	Kubernetes API サーバーと通信するときにコントローラーが使用する 1 秒あたりのクエリ数 (QPS) の制限。バースト値は QPS 値に基づいて自動的に設定されます。	`100`; オプション

オプション	説明	デフォルト

`resources`	Tridentコントローラー、ノード、オペレーター ポッドの Kubernetes リソース制限と要求を設定します。Kubernetes でのリソース割り当てを管理するために、コンテナとサイドカーごとに CPU とメモリを設定できます。
	
	リソース要求と制限の設定の詳細については、以下を参照してください。"ポッドとコンテナのリソース管理" 。
	
	• コンテナまたはフィールドの名前を変更しないでください。
	• インデントを変更しないでください。YAML インデントは適切な解析に重要です。
	
	• デフォルトでは制限は適用されません。リクエストのみにデフォルト値があり、指定されていない場合は自動的に適用されます。
	• コンテナ名は、ポッド仕様に表示されるとおりにリストされます。
	• サイドカーは各メイン コンテナーの下にリストされます。
	• TORCを確認する `status.CurrentInstallationParams `現在適用されている値を表示するフィールド。
		`resources:`
		`  controller:`
		`  trident-main:`
		`  requests:`
		`  cpu: 10m`
		`  memory: 80Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-provisioner:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-attacher:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-resizer:`
		`  requests:`
		`  cpu: 3m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-snapshotter:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  trident-autosupport:`
		`  requests:`
		`  cpu: 1m`
		`  memory: 30Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  node:`

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

コントローラポッドとノードポッドについて

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

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

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

Helm（オフラインモード）を使用したTridentのオペレータの導入

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

Trident 26.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ **Trident **に関する****の重要な情報

- Trident 26.10はKubernetes 1.27から1.37までをサポートしています。Kubernetesをアップグレードする前に、"Tridentの要件"でバージョン固有のイメージリストを確認してください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

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

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

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

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

作業を開始する前に

"導入の前提条件"に加えて"Helm バージョン 3"が必要です。

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

手順

1. Trident Helmリポジトリを追加します。

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

2. `helm install`を使用して、デプロイメントの名前とイメージレジストリの場所を指定します。"TridentとCSIの画像"は1つのレジストリまたは異なるレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例では、 `100.2610.0`はインストールするTridentのバージョンです。

1つのレジストリ内のイメージ

```
helm install <name> netapp-trident/trident-operator --version
100.2610.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

異なるレジストリ内の画像

```
helm install <name> netapp-trident/trident-operator --version
100.2610.0 --set imageRegistry=<your-registry> --set operatorImage
=<your-registry>/trident-operator:26.10.0 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:26.10
--set tridentImage=<your-registry>/trident:26.10.0 --create
-namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```

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

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、 `debug`のデフォルト値を変更するには、次のコマンドを実行します。ここで、 `100.2610.0`はインストールするTridentのバージョンです：

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace trident --set tridentDebug=true
```

nodePrep値を追加するには、次のコマンドを実行します。

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

設定オプション

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

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

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

オプション	説明	デフォルト

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

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

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

オプション	説明	デフォルト
`tridentK8sTimeout`	ほとんどの Kubernetes API 操作のデフォルトの 180 秒のタイムアウトをオーバーライドします (ゼロ以外の場合は秒単位)。 その `tridentK8sTimeout` パラメータはTridentのインストールにのみ適用されます。	`180`
`tridentHttpRequestTimeout`	HTTP要求のデフォルトの90秒タイムアウトをで上書きします `0s` タイムアウトの期間は無限です。負の値は使用できません。	`"90s"`
`tridentSilenceAutosupport`	Trident定期AutoSupportレポートをディセーブルにできます。	「偽」
`tridentExcludeAutosupport`	`true` に設定した場合、Tridentはインストール中にAutoSupportコンテナをセットアップしません。	「偽」
`tridentAutosupportImageTag`	Trident AutoSupportコンテナのイメージのタグを上書きできます。	`<version>`
`tridentAutosupportProxy`	Trident AutoSupportコンテナがHTTPプロキシ経由で自宅に電話できるようにします。	""
`tridentLogFormat`	Tridentロギング形式を設定し(`text`ます。または `json`)	`"text"`
`tridentDisableAuditLog`	Trident監査ロガーをディセーブルにします。	「真」
`tridentLogLevel`	Tridentのログレベルを、、 `debug` `info`、、 `warn`、 `error`または `fatal`に設定 `trace`できます。	`"info"`
`tridentDebug`	Tridentのログレベルをに設定できます `debug`。	「偽」
`tridentLogWorkflows`	特定のTridentワークフローのトレースロギングまたはログ抑制を有効にできます。	""
`tridentLogLayers`	トレースロギングまたはログ抑制に対して特定のTridentレイヤをイネーブルにできます。	""
「 tridentImage 」のように入力します	Tridentのイメージを完全に上書きできます。	""
`tridentImageTag`	Tridentのイメージのタグを上書きできます。	""

オプション	説明	デフォルト
`tridentProbePort`	Kubernetesの活性/準備プローブに使用されるデフォルトポートを上書きできます。	""
`windows`	TridentをWindowsワーカーノードにインストールできるようにします。	「偽」
`enableForceDetach`	強制デタッチ機能を有効にできます。ノード ヘルス チェック (NHC) オペレーターとの統合により、強制デタッチ プロセスを自動化できます。詳細については、"Tridentによるステートフル アプリケーションのフェイルオーバーの自動化" 。	「偽」
`enableDataLIFRefresh`	LIFが追加、削除、または新しいアドレスが割り当てられた際に、公開ボリュームのNVMeデータLIFを更新します。環境：ユニファイドONTAP上の `ontap-san`（`sanType: nvme`を使用）およびASA r2上のNVMe/TCP。詳細については、"NVMeデータLIFを更新する"を参照してください。	「偽」
`excludePodSecurityPolicy`	オペレータポッドのセキュリティポリシーを作成から除外します。	「偽」
`nodePrep`	指定したデータストレージプロトコルを使用してボリュームを管理できるように、TridentでKubernetesクラスタのノードを準備できるようにします。現在`**iscsi**`サポートされている値は、のみです。 OpenShift 4.19 以降、この機能でサポートされるTridentの最小バージョンは 25.06.1 です。	

オプション	説明	デフォルト

resources	Tridentコントローラー、ノード、オペレーター ポッドの Kubernetes リソース制限と要求を設定します。Kubernetes でのリソース割り当てを管理するために、コンテナとサイドカーごとに CPU とメモリを設定できます。
	
	リソース要求と制限の設定の詳細については、以下を参照してください。"ポッドとコンテナのリソース管理" 。
	
	• コンテナまたはフィールドの名前を変更しないでください。
	• インデントを変更しないでください。YAML インデントは適切な解析に重要です。
	
	• デフォルトでは制限は適用されません。リクエストのみにデフォルト値が設定されます。
	• コンテナ名は、ポッド仕様に表示されるとおりにリストされます。
	• サイドカーは各メイン コンテナーの下にリストされます。
	• TORCを確認する `status.CurrentInstallationParams `現在適用されている値を表示するフィールド。
		resources:
		  controller:
		  trident-main:
		  requests:
		  cpu: 10m
		  memory: 80Mi
		  limits:
		  cpu:
		  memory:
		  csi-provisioner:
		  requests:
		  cpu: 2m
		  memory: 20Mi
		  limits:
		  cpu:
		  memory:
		  csi-attacher:
		  requests:
		  cpu: 2m
		  memory: 20Mi
		  limits:
		  cpu:
		  memory:
		  csi-resizer:
		  requests:
		  cpu: 3m
		  memory: 20Mi
		  limits:
		  cpu:
		  memory:
		  csi-snapshotter:
		  requests:
		  cpu: 2m
		  memory: 20Mi
		  limits:
		  cpu:
		  memory:
		  trident-autosupport:
		  requests:
		  cpu: 1m
		  memory: 30Mi
		  limits:
		  cpu:
		  memory:

Tridentオペレータのインストールをカスタマイズ

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

コントローラポッドとノードポッドについて

Trident は、クラスター内の各ワーカー ノード上で単一のコントローラー ポッドとノード ポッドとして実行されます。ノード ポッドは、 Tridentボリュームをマウントする可能性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。

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

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

設定オプション

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

このテーブルの詳細 `TridentOrchestrator` 属性。

パラメータ	説明	デフォルト
`namespace`	Tridentをインストールするネームスペース	`"default"`
「バグ」	Tridentのデバッグを有効にする	「偽」
`enableForceDetach`	`ontap-san`、 `ontap-san-economy` 、 `ontap-nas` 、 そして `ontap-nas-economy`のみ。Kubernetes Non-Graceful Node Shutdown (NGNS) と連携して、ノードが正常でなくなった場合に、マウントされたボリュームを持つワークロードを新しいノードに安全に移行する機能をクラスター管理者に付与します。詳細については、"Tridentによるステートフル アプリケーションのフェイルオーバーの自動化" 。	「偽」
`enableDataLIFRefresh`	LIFが追加、削除、または新しいアドレスが割り当てられた際に、公開ボリュームのNVMeデータLIFを更新できるようにします。環境：ユニファイドONTAP上の `ontap-san`（ `sanType: nvme`を使用）およびASA r2上のNVMe/TCP。詳細については、"NVMeデータLIFを更新する"を参照してください。	「偽」

パラメータ	説明	デフォルト
`windows`	をに設定します `true` Windowsワーカーノードへのインストールを有効にします。	「偽」
`cloudProvider`	AKSクラスタで管理IDまたはクラウドIDを使用する場合はに設定します `"Azure"`。EKSクラスタでクラウドIDを使用する場合は、に設定され `"AWS"`ます。`GKE`クラスタでクラウド`ID`を使用する場合は、に設定します `"GCP"`。	`""`
`cloudIdentity`	AKSクラスタでクラウドIDを使用する場合は、ワークロードID（「azure.workload.identity/client-id：xxxxxxxxxx-xxxx-xxxxxxx」）に設定します。EKSクラスタでクラウドIDを使用する場合は、AWS IAMロール（「eks.amazonaws.com/role-arn: arn：aws：iam：：123456：role / Trident -role」）に設定されます。GKEクラスタでクラウドIDを使用する場合は、クラウドID（「IAM.GKE.IO/gcp-service-account: xxxx@mygcpproject.iam.gserviceaccount.com」）に設定します。	`""`
`gcpWIFAudience`	OpenShift 上の Google Cloud Workload Identity Federation で使用される投影されたサービスアカウントトークンのオーディエンス。プロバイダの正規リソース名を使用してください。詳細については、"Workload Identity フェデレーションを使用して Google Cloud NetApp Volumes を設定する" を参照してください。	`""`
`IPv6`	IPv6経由のTridentのインストール	いいえ
`k8sTimeout`	Kubernetes 操作のタイムアウト。 その `k8sTimeout`パラメータはTridentのインストールにのみ適用されます。	`180sec`
`silenceAutosupport`	AutoSupportバンドルをNetAppに送信しない自動	「偽」
「 autosupportImage 」を参照してください	AutoSupport テレメトリのコンテナイメージ	`"netapp/trident-autosupport:26.10"`
「 autosupportProxy 」と入力します	AutoSupportを送信するためのプロキシのアドレス/ポートテレメータ	`"http://proxy.example.com:8888"`
uninstall	Tridentのアンインストールに使用するフラグ	「偽」
`logFormat`	使用するTridentログ形式[text、json]	`"text"`
「 tridentImage 」のように入力します	インストールするTridentイメージ	`"netapp/trident:26.10.0"`
「 imageRegistry 」と入力します	形式の内部レジストリへのパス `<registry FQDN>[:port][/subpath]`	`"registry.k8s.io"`

パラメータ	説明	デフォルト
「 kubeletDir 」を参照してください	ホスト上の kubelet ディレクトリへのパス	`"/var/lib/kubelet"`
`wipeout`	Tridentを完全に削除するために削除するリソースのリスト	
「 imagePullSecrets 」	内部レジストリからイメージをプルするシークレット	
`imagePullPolicy`	Tridentオペレータのイメージプルポリシーを設定します。有効な値は次のとおりです。 `Always` 常にイメージをプルする。 `IfNotPresent` ノード上にイメージが存在しない場合にのみ取得します。 `Never` 画像を絶対に引き出さないでください。	`IfNotPresent`
controller'PluginNodeSelector	ポッド用の追加のノードセレクタ。 の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです
controllerPluginTolerations	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです
「 nodePluginNodeSelector 」	ポッド用の追加のノードセレクタ。の形式はと同じです `pod.spec.nodeSelector`。	デフォルトはありません。オプションです
「 nodePluginTolerations` 」	ポッドに対するKubernetesの許容範囲を上書きします。はと同じ形式です `pod.spec.Tolerations`。	デフォルトはありません。オプションです
`nodePrep`	指定したデータストレージプロトコルを使用してボリュームを管理できるように、TridentでKubernetesクラスタのノードを準備できるようにします。現在 **`iscsi`**`サポートされている値は、のみです。 OpenShift 4.19 以降、この機能でサポートされるTrident の最小バージョンは 25.06.1 です。	
`k8sAPIQPS`	Kubernetes API サーバーと通信するときにコントローラーが使用する 1 秒あたりのクエリ数 (QPS) の制限。バースト値は QPS 値に基づいて自動的に設定されます。	`100`; オプション
`enableConcurrency`	スループット向上のために Trident コントローラの同時操作を有効にします。サポートされているドライバーの詳細については、リンク： Controller Scalability documentation を参照してください。	`enableCOSI`

パラメータ	説明	デフォルト
Trident Container Object Storage Interface（COSI）フロントエンドを有効にします。`enableConcurrency`を`true`に設定する必要があります。Trident COSIは、コミュニティCOSIコントローラと並行して実行することはできません。詳細については、"COSIを使用してオブジェクトストレージをプロビジョニングする"を参照してください。	「偽」	`resources`

パラメータ	説明	デフォルト

Tridentコントローラーとノード ポッドのKubernetes リソース制限と要求を設定します。Kubernetes でのリソース割り当てを管理するために、コンテナとサイドカーごとに CPU とメモリを設定できます。
 	
リソース要求と制限の設定の詳細については、以下を参照してください。"ポッドとコンテナのリソース管理"。
 	
• コンテナまたはフィールドの名前を変更しないでください。
• インデントを変更しないでください。YAML インデントは適切な解析に重要です。
 	
• デフォルトでは制限は適用されません。リクエストのみにデフォルト値があり、指定さ
	`resources:`
	`  controller:`
	`  trident-main:`
	`  requests:`
	`  cpu: 10m`
	`  memory: 80Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-provisioner:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-attacher:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-resizer:`
	`  requests:`
	`  cpu: 3m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-snapshotter:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  trident-autosupport:`
	`  requests:`
	`  cpu: 1m`
	`  memory: 30Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  node:`
		`httpsMetrics`

パラメータ	説明	デフォルト
Prometheus メトリックエンドポイントに対して HTTPS を有効にします。	いいえ	「ホストネットワーク」

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

構成例

属性は[設定オプション]定義するとき `TridentOrchestrator`インストールをカスタマイズします。

▼ 基本的なカスタム構成

この例は、 `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml`コマンドは、基本的なカスタムインストールを表します。

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

▼ ノードセレクター

この例では、ノード セレクターを使用してTrident をインストールします。

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

▼ **Windows**ワーカーノード

この例は、 `cat deploy/crds/tridentorchestrator_cr.yaml`コマンドは、Windows ワーカー ノードにTrident をインストールします。

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

▼ **AKS** クラスター上のマネージド **ID**

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

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

▼ **AKS** クラスター上のクラウド **ID**

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

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

▼ **EKS** クラスター上のクラウド **ID**

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

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
```

```
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/trident-role'"
```

▼ **GKEのクラウドID**

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

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

▼ **Tridentコントローラーと Trident Linux** ノード ポッドの **Kubernetes** リソース要求と制限の構成

この例では、 TridentコントローラーとTrident Linux ノード ポッドの Kubernetes リソース要求と制限を構成します。

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

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
```

```
          memory: 80Mi
        limits:
          cpu: 200m
          memory: 256Mi
      # sidecars
      csi-provisioner:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-attacher:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-resizer:
        requests:
          cpu: 3m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-snapshotter:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      trident-autosupport:
        requests:
          cpu: 1m
          memory: 30Mi
        limits:
          cpu: 50m
          memory: 128Mi
    node:
      linux:
        trident-main:
          requests:
            cpu: 10m
            memory: 60Mi
```

```
          limits:
            cpu: 200m
            memory: 256Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 1m
            memory: 10Mi
          limits:
            cpu: 50m
            memory: 32Mi
```

▼ **Trident**コントローラーと**Trident Windows** および **Linux** ノード ポッドの **Kubernetes** リソース要求と制限の構成

この例では、 TridentコントローラーとTrident Windows および Linux ノード ポッドの Kubernetes リソース要求と制限を構成します。

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

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  windows: true
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
        limits:
          cpu: 200m
          memory: 256Mi
        # sidecars
      csi-provisioner:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
```

```
          cpu: 100m
          memory: 64Mi
      csi-attacher:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-resizer:
        requests:
          cpu: 3m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-snapshotter:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      trident-autosupport:
        requests:
          cpu: 1m
          memory: 30Mi
        limits:
          cpu: 50m
          memory: 128Mi
    node:
      linux:
        trident-main:
          requests:
            cpu: 10m
            memory: 60Mi
          limits:
            cpu: 200m
            memory: 256Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 1m
            memory: 10Mi
          limits:
            cpu: 50m
```

```
            memory: 32Mi
      windows:
        trident-main:
          requests:
            cpu: 6m
            memory: 40Mi
          limits:
            cpu: 200m
            memory: 128Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 6m
            memory: 40Mi
          limits:
            cpu: 100m
            memory: 128Mi
        liveness-probe:
          requests:
            cpu: 2m
            memory: 40Mi
          limits:
            cpu: 50m
            memory: 64Mi
```

tridentctlを使用してインストールします

tridentctlを使用したTridentのインストール

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

Trident 26.10に関する重要な情報

- Tridentに関する次の重要な情報をお読みください。*

▼ Trident に関するの重要な情報

- Kubernetes 1.37 は Trident 26.10 でサポートされています。Kubernetes をアップグレードする前に Trident をアップグレードしてください。
- Tridentでは、SAN環境でのマルチパス構成の使用が厳密に適用されます。multipath.confファイルの推奨値はです `find_multipaths: no`。

  非マルチパス構成またはを使用 `find_multipaths: yes` または `find_multipaths: smart`

multipath.confファイルの値が原因でマウントが失敗します。Tridentはの使用を推奨しています `find_multipaths: no` 21.07リリース以降

を使用して**Trident**をインストール `tridentctl`

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

作業を開始する前に

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

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

▼ **Details**

1. Kubernetesのバージョンを確認します。

```
kubectl version
```

2. クラスタ管理者の権限を確認します。

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

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v26.10.0/trident-
installer-26.10.0.tar.gz
tar -xf trident-installer-26.10.0.tar.gz
cd trident-installer
```

手順2：Tridentをインストールする

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

標準モード

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

1つのレジストリ内のイメージ

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:26.10 --trident
-image <your-registry>/trident:26.10.0
```

異なるレジストリ内の画像

```
./tridentctl install -n trident --image-registry <csi-registry>
--autosupport-image <autosupport-registry>/trident-autosupport:26.10
--trident-image <trident-registry>/trident:26.10.0
```

インストールステータスは次のようになります。

```
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=26.10.0
INFO Trident installation succeeded.
....
```

インストールを確認します。

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

ポッドの作成ステータスを使用する

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

```
kubectl get pods -n trident

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

インストーラが正常に完了しない場合、または `trident-controller-<generated id>` (`trident-csi-<generated id>`（23.01より前のバージョン）が*Running*ステータスでない場合、プラットフォームはインストールされていません。 `-d`を使用して"デバッグモードをオンにします"問題のトラブルシューティングを行います。

を使用します `tridentctl`

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

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.0        | 26.10.0        |
+----------------+----------------+
```

構成例

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

▼ **Windows**ノオト

WindowsノードでTridentを実行するには、次の手順を実行します。

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

▼ 強制的に切り離し

詳細については、"Tridentによるステートフル アプリケーションのフェイルオーバーの自動化" 。

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

▼ **Trident**コントローラーの同時操作を有効にする

スループットを向上させるために同時 Trident コントローラー操作を有効にするには、この例に示すように、インストール時に `--enable-concurrency`オプションを追加します。サポートされているドライバーの詳細については、リンク先を参照してください：https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability[コントローラーのスケーラビリティに関するドキュメント]

```
tridentctl install --enable-concurrency -n trident
```

▼ **NVMe**データ**LIF**を更新する

LIF構成が変更されたときに公開ボリュームのNVMeデータLIFを更新するには、 `--enable-datalif-refresh`を使用してTridentをインストールします。サポートされているドライバと構成の詳細については、"NVMeデータLIFを更新する"を参照してください。

```
tridentctl install --enable-datalif-refresh -n trident
```

Trident tridentctl インストールのカスタマイズ

Tridentインストーラを使用して、インストールをカスタマイズできます。

インストーラの詳細を確認してください

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

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

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

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

- trident-clusterrolebinding.yaml
- trident-deployment.yaml
- trident-CRDs .YAML

- trident-clusterrolment.yaml
- trident-demimonimon.yamml`
- trident-service.yaml
- trident-namespac.yaml
- trident-ServiceAccount.yaml
- `trident-resourcequota.yaml*`

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

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

OpenShift認定オペレータを使用してインストール

OpenShift OperatorHubを使用したTridentのインストール

Red Hat OpenShiftを使用する場合は、Red Hat認定オペレータを使用してNetApp Tridentをインストールできます。Red Hat OpenShift Container PlatformからTridentをインストールするには、次の手順を実行します。

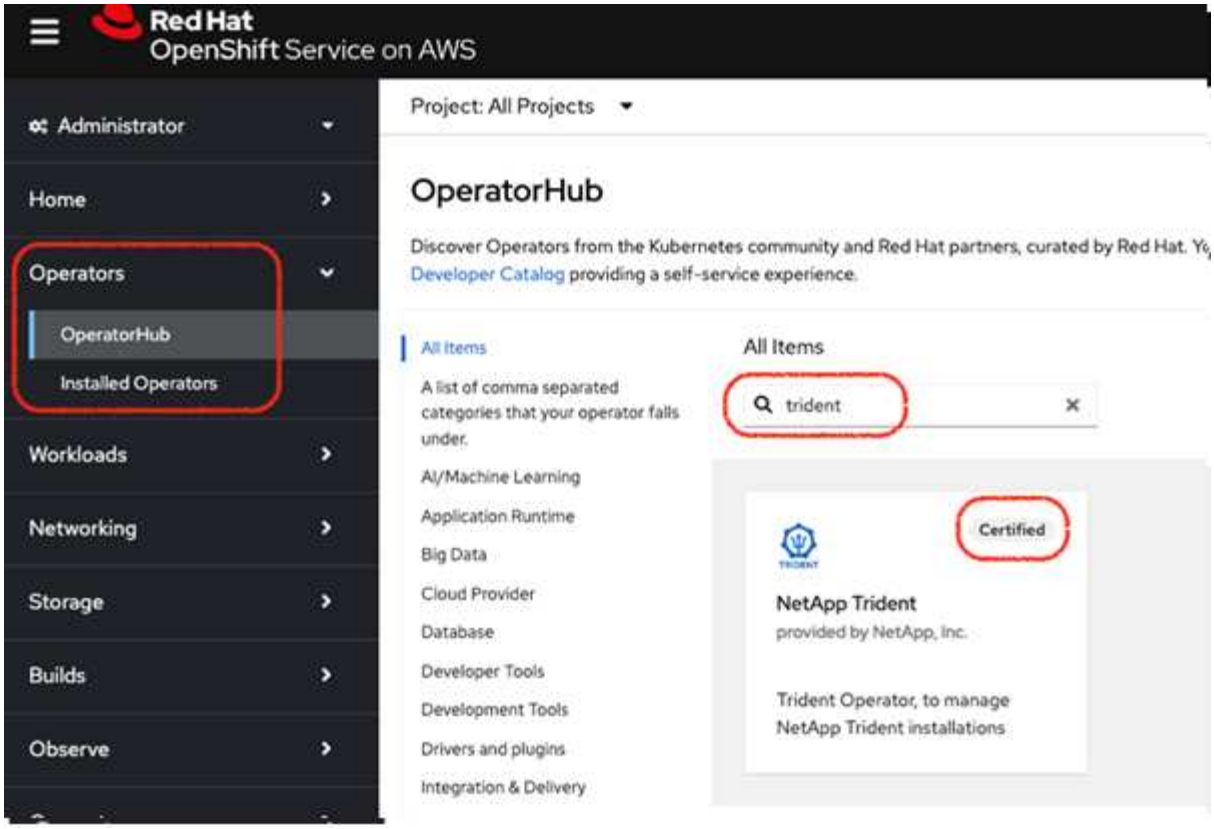
作業を開始する前に

インストールを開始する前に、を"Tridentをインストールするための環境の準備"参照してください。

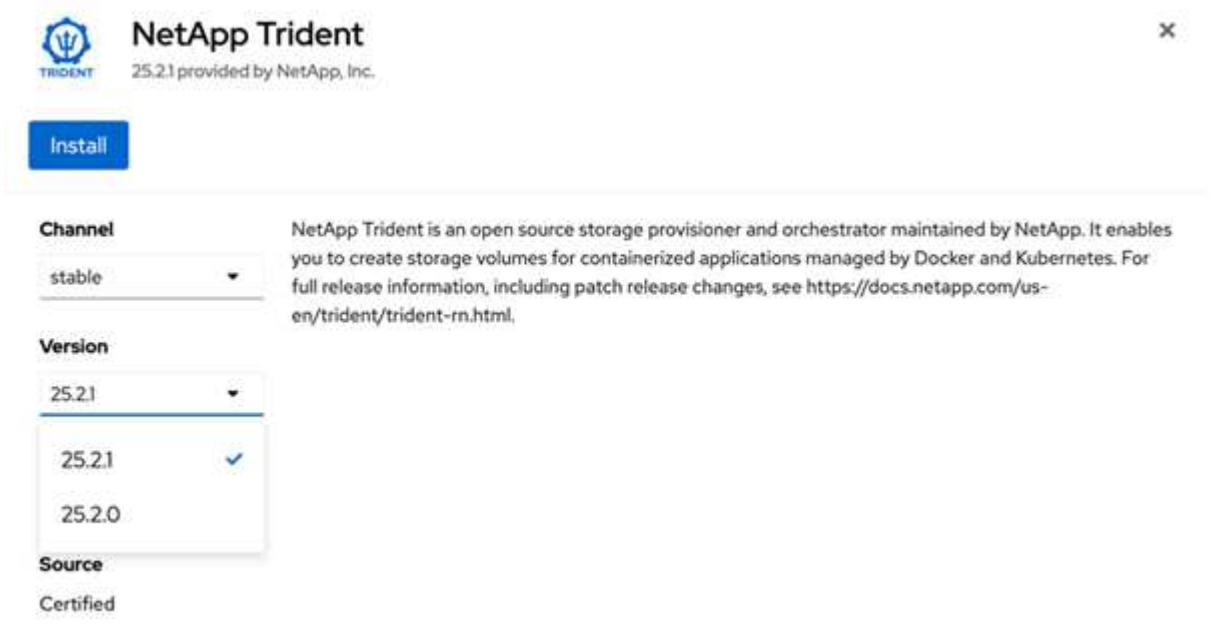
Tridentオペレータの検索とインストール

手順

1. OpenShift OperatorHubに移動し、NetApp Tridentを検索します。

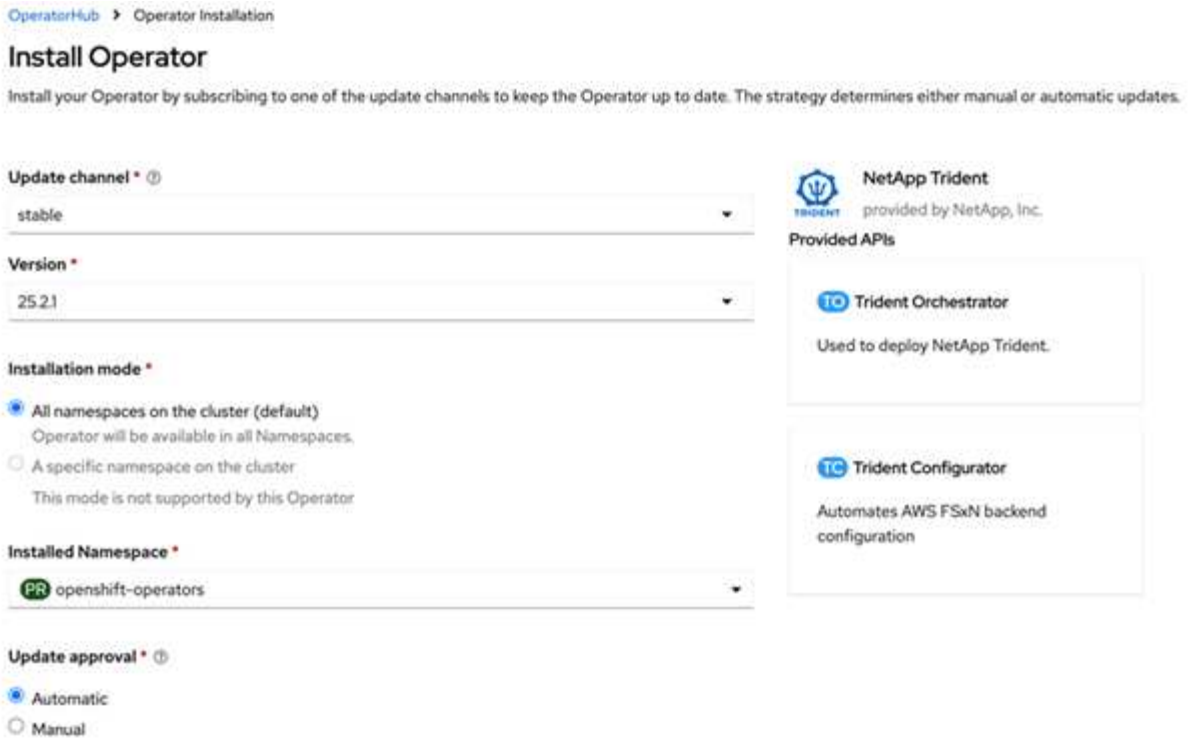


2. * NetApp Trident* をクリックしてインストール設定を開きます。
3. 必要なオプションを選択し、「インストール」をクリックしてオペレーター構成を開きます。

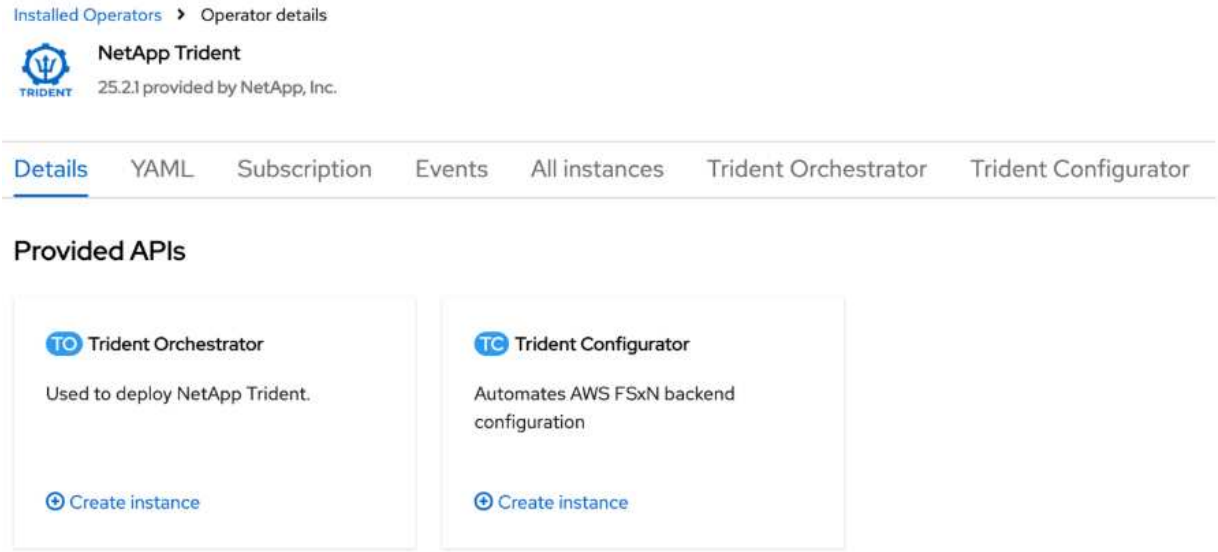


必ず最新の Operator バージョンを選択してください。

4. すべてのパラメータをそのままにして、[インストール] をクリックします。



5. オペレーターの詳細を表示するには、「オペレーターの表示」をクリックします。



6. **YAML** ビュー をクリックし、次の内容をフォームに貼り付けます。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
  namespace: openshift-operators
spec:
  IPv6: false
```

```
  debug: false
  nodePrep:
  - iscsi
  imageRegistry: ''
  k8sTimeout: 180s
  namespace: trident
  silenceAutosupport: false
```

UIにはデフォルトの例が表示されます。設定全体をコピーする代わりに、直接編集することもできます。

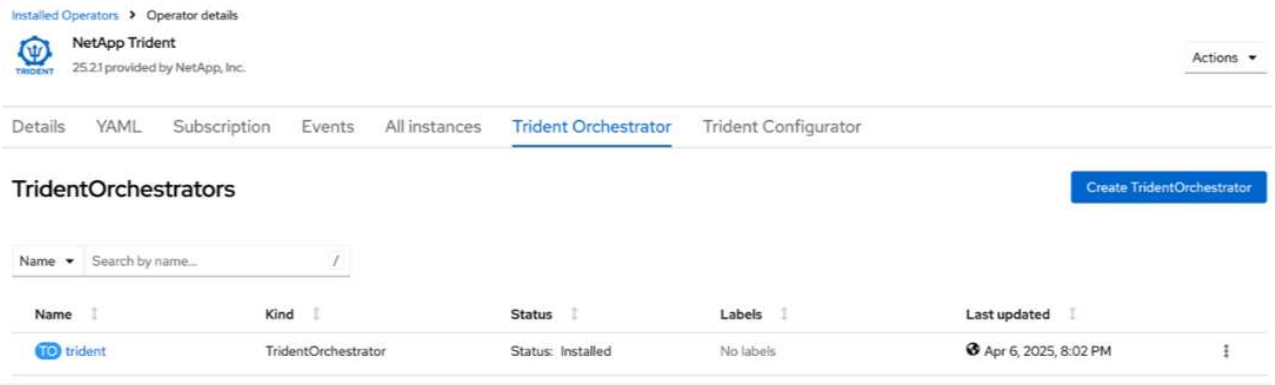
オプション：並行処理を有効にする

同時実行を有効にするには、次のフィールドを spec に追加します：

```
enableConcurrency: true
```

- Red Hat Enterprise Linux CoreOS（RHCOS）では iSCSI が有効化および構成されていません。
- 追加できるのは `nodePrep`すべての OpenShift ワーカー ノードで iSCSI サービスとマルチパス サービスの両方を設定および有効にするパラメーター。
- 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。

1. *作成*をクリックすると、 Trident Orchestrator が完全にインストールされます。



Tridentオペレーターをアンインストールする

手順

1. インストールされている演算子のリストからTrident演算子を選択します。
2. 演算子からすべてのオペランドインスタンスを削除する場合に選択します。

[この演算子からすべてのオペランドインスタンスを削除する]チェックボックスをオンにし

ないと、Tridentはアンインストールされません。

3. [アンインストール] をクリックします。

OpenShift認定Tridentオペレーターに切り替える

コミュニティオペレーター、Helmベースのインストール、または手動でデプロイされたオペレーターから、Red Hat OpenShift認定Tridentオペレーターに切り替えることができます。各方法のプロセスでは、既存のオペレーターをアンインストールしてから、OperatorHubを使用して認定オペレーターをインストールします。

作業を開始する前に

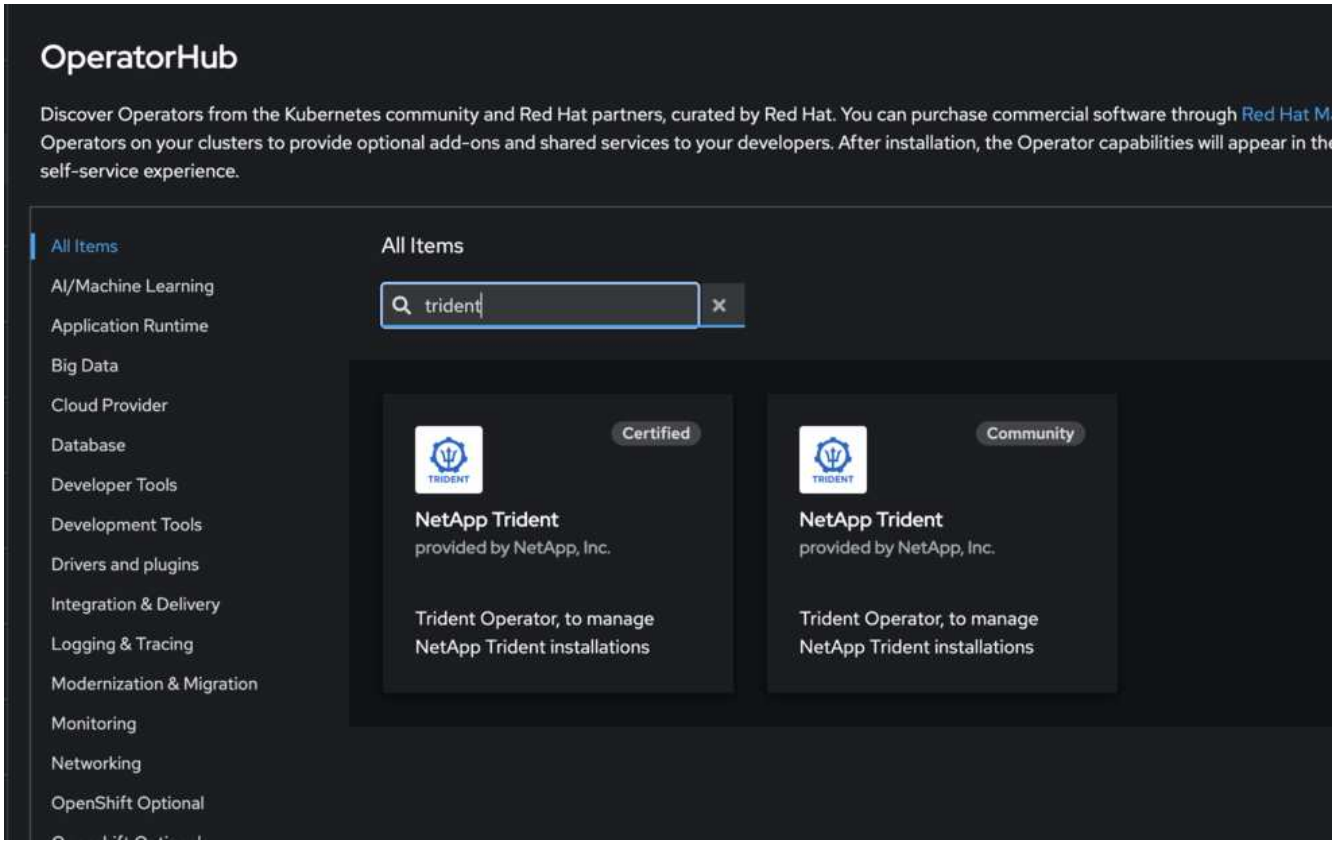
インストールを開始する前に、を"Tridentをインストールするための環境の準備"参照してください。

アンインストールプロセス中に `TridentOrchestrator`カスタムリソース（CR）を削除しないでください。 `TridentOrchestrator`CRは、認定オペレーターをインストールした後に必要なバックエンドとストレージクラスの設定を保持します。

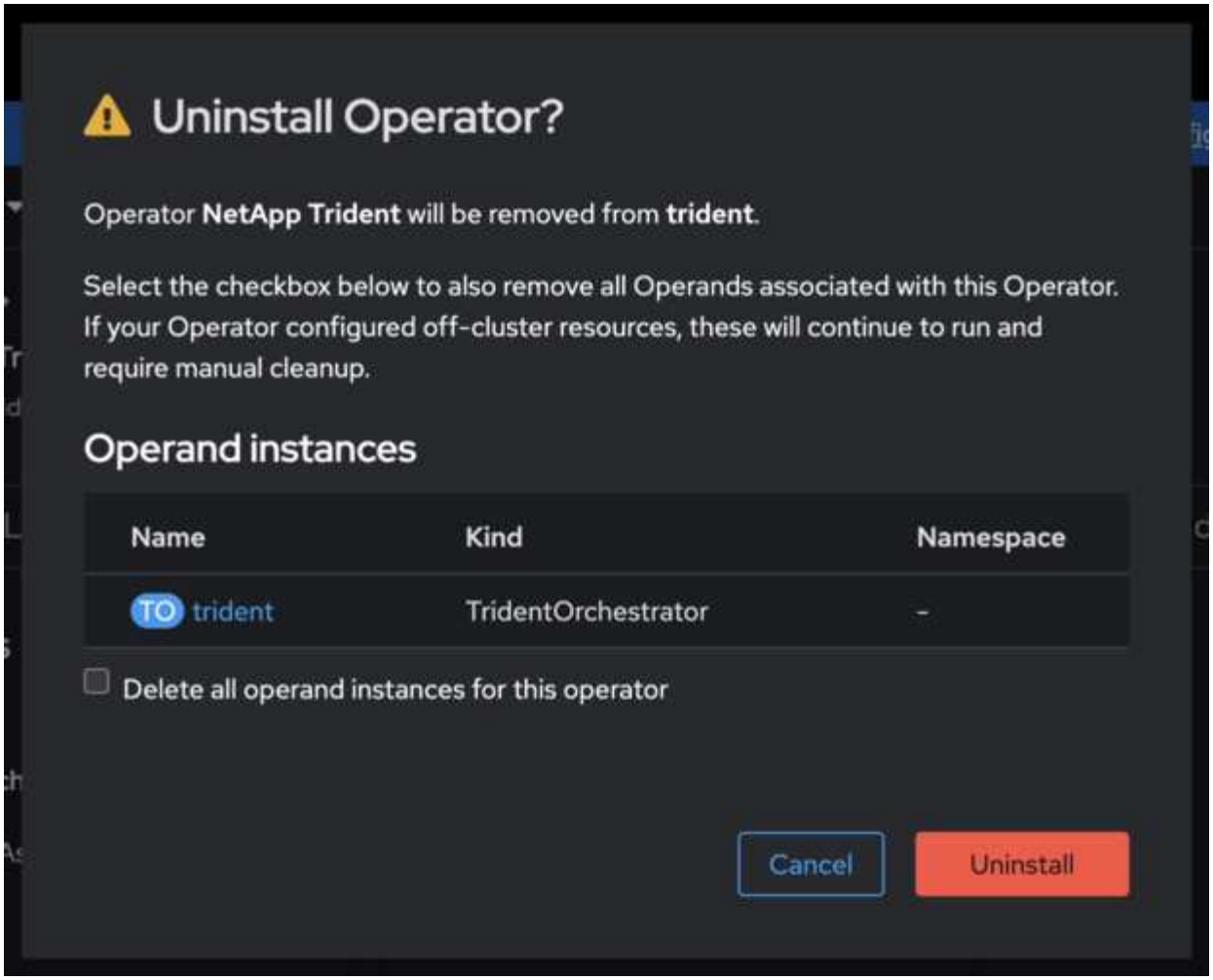
コミュニティオペレーターから切り替える

手順

1. OpenShiftコンソールを使用して、OperatorHubに移動します。



2. NetApp Tridentコミュニティオペレータを検索します。



[Delete all operand instances from this operator]*を選択しないでください。

3. [アンインストール] をクリックします。
4. アンインストールが完了したら、OpenShift 認定オペレーターをインストールするに進みます。

Helmベースのoperatorインストールから切り替える

手順

1. Trident インストールの Helm リリースを一覧表示します：

```
helm ls -n trident
```

2. Helm リリースをアンインストールします：

```
helm uninstall <release-name> -n trident
```

3. アンインストールが完了したら、OpenShift 認定オペレーターをインストールするに進みます。

手動で展開されたオペレーターから切り替える

インストーラー パッケージの `bundle.yaml`を使用してオペレーターを手動で展開することでTridentをインストールした場合は、同じマニフェストを削除して削除します。

手順

1. バンドル マニフェストを使用してオペレーターのデプロイメントを削除します：

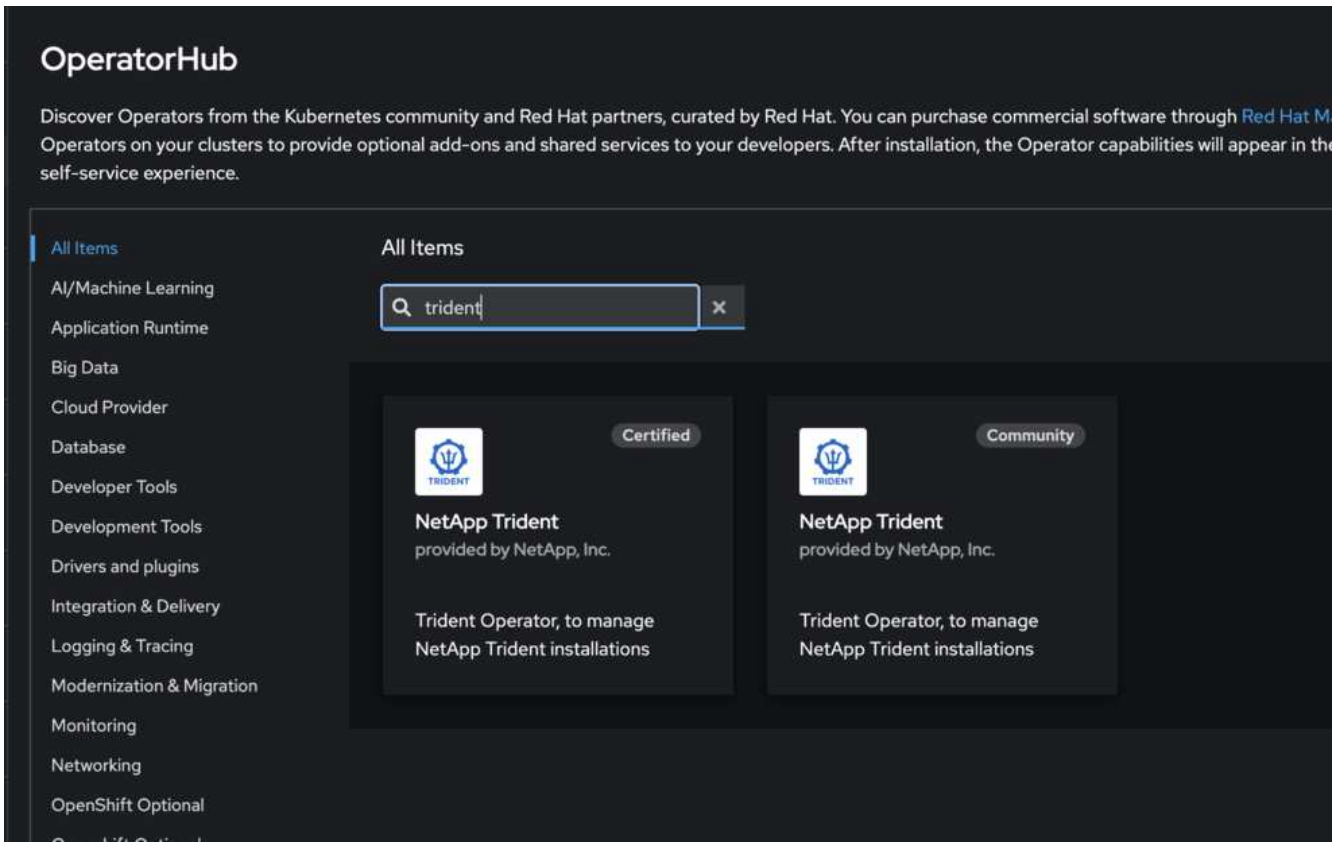
```
kubectl delete -f deploy/bundle.yaml -n trident
```

2. アンインストールが完了したら、OpenShift 認定オペレーターをインストールするに進みます。

OpenShift 認定オペレーターをインストールする

手順

1. Red Hat OperatorHubに移動します。
2. NetApp Trident演算子を検索して選択します。



3. 画面の指示に従って、オペレータをインストールします。

検証

- コンソールのOperatorHubをチェックして、新しい認定オペレータが正常にインストールされていることを確認します。

Tridentを使用

Tridentワーカーノードを準備する

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

適切なツールを選択する

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

NFSツール

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

iSCSIツール

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

NVMeツール

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

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

SCSI over FCツール

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

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

考慮事項：* 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の自己回復機能は、SCSI over FCには適用されません。

SMBツール

"SMBボリュームをプロビジョニングする準備をします" 使用している場合: `ontap-nas` SMB ボリュームをプロビジョニングします。

ノードサービスの検出

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

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

イベントを確認します

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

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

検出されたサービスを確認

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

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

NFS ボリューム

オペレーティングシステム用のコマンドを使用して、NFSツールをインストールします。ブート時にNFSサービスが開始されていることを確認します。

RHEL 8以降

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

Ubuntu

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

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

iSCSI ボリューム

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

iSCSIの自己回復機能

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

1. *希望するiSCSIセッションの状態と現在のiSCSIセッションの状態を識別します

2. *希望する状態と現在の状態を比較して、必要な修理を特定します。Tridentは、修理の優先順位と、修理をいつプリエンプトするかを決定します。
3. *現在のiSCSIセッションの状態を希望するiSCSIセッションの状態に戻すために必要な修復*を実行します。

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

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

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

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

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

iSCSIツールをインストール

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

作業を開始する前に

- Kubernetes クラスタ内の各ノードには一意の IQN を割り当てる必要があります。* これは必須の前提条件です * 。
- RHCOSバージョン4.5以降またはRHEL互換のその他のLinuxディストリビューションをで使用している場合は、を使用します `solidfire-san` DriverおよびElement OS 12.5以前。CHAP認証アルゴリズムがMD5 inに設定されていることを確認します `/etc/iscsi/iscsid.conf`。Element 12.7では、FIPS準拠のセキュアなCHAPアルゴリズムSHA1、SHA-256、およびSHA3-256が提供されています。

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

- iSCSI PVS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、StorageClassでmountOptionを指定してインラインのスペース再生を実行します `discard`。を参照してください "Red Hat のドキュメント"。

- 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：

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

4. マルチパスを有効化：

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

> の下に `defaults`含むを `find_multipaths no`確認します `/etc/multipath.conf`。

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

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

6. 'iSCSI' を有効にして開始します

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

```
scsitools
```

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

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

3. スキャンを手動に設定：

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

4. マルチパスを有効化：

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

> の下に `defaults`含むを `find_multipaths no`確認します `/etc/multipath.conf`。

5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されていることを確認します。

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

> Ubuntu 18.04 では 'iSCSI デーモンを起動するために 'open-iscsi' を起動する前に 'iscsiadm を持つターゲット・ポートを検出する必要がありますまたは 'iscsid' サービスを 'iscsid' を自動的に開始するように変更することもできます

iSCSI自己回復の設定または無効化

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

- * iSCSIの自己修復間隔*：iSCSIの自己修復を実行する頻度を指定します（デフォルト：5分）。小さい数

値を設定することで実行頻度を高めるか、大きい数値を設定することで実行頻度を下げることができます。

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

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

Helm

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

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

```
helm install trident trident-operator-100.2610.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

Tridentctl

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

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

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

NVMe/TCPホリユウム

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

> - NVMeにはRHEL 9以降が必要です。
> - Kubernetesノードのカーネルバージョンが古すぎる場合や、使用しているカーネルバージョンに対応するNVMeパッケージがない場合は、ノードのカーネルバージョンをNVMeパッケージで更新しなければならないことがあります。

RHEL 9

```
sudo yum install nvme-cli
```

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

Ubuntu

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

インストールを確認します

インストールが完了したら、次のコマンドを使用して、Kubernetesクラスタ内の各ノードに一意のNQNが割り当てられていることを確認します。

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

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

SCSI over FCボリューム

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

前提条件

FCに必要なネットワークとノードを設定します。

ネットワーク設定

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

   対応するホストオペレーティングシステムユーティリティを参照してください。

3. ホストとターゲットのWWPNを使用してFCスイッチにゾーニングを設定します。

   詳細については、各スイッチベンダーのドキュメントを参照してください。

   詳細については、次のONTAPドキュメントを参照してください。

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

FCツールのインストール

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

- FC PVS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、StorageClassでmountOptionを指定してインラインのスペース再生を実行します `discard`。を参照してください "Red Hat のドキュメント"。

RHEL 8以降

1. 次のシステムパッケージをインストールします。

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

2. マルチパスを有効化：

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

> の下に `defaults`含むを `find_multipaths no`確認します `/etc/multipath.conf`。

3. が実行中であることを確認し `multipathd`ます。

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

2. マルチパスを有効化：

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

> の下に `defaults`含むを `find_multipaths no`確認します

`/etc/multipath.conf。

3. が有効で実行中であることを確認し `multipath-tools`ます。

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

SMBボリュームをプロビジョニングする準備をします

SMBボリュームをプロビジョニングするには、 `ontap-nas` ドライバー。

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

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

作業を開始する前に

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

- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2022を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Active Directoryクレデンシャルを含む少なくとも1つのTridentシークレット。シークレットを生成するには `smbcreds`：

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

- Windowsサービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub: CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

手順

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

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

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

   a. 必要に応じて、共有のディレクトリパス構造を作成します。

      。 `vserver cifs share create` コマンドは、共有の作成時に-pathオプションで指定されている

パスを確認します。指定したパスが存在しない場合、コマンドは失敗します。

b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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

を参照してください "SMB 共有を作成" 詳細については、

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

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

COSIを使用してオブジェクトストレージをプロビジョニングする

Container Object Storage Interface（COSI）を使用して、Tridentが管理するKubernetesリソースを介してKubernetesワークロード用のS3バケットをプロビジョニングします。

概要

COSIは、オブジェクトストレージプロバイダーをKubernetesワークロードに接続します。Tridentは以下のリソースを提供します：

- `TridentObjectStore`：Tridentをオブジェクトストレージシステムおよびその認証情報に接続します。このリソースはTridentネームスペースに作成します。
- `BucketClass`：オブジェクトストアの選択およびバケットの削除ポリシーを定義します。
- `BucketClaim`：バケットを要求します。
- `BucketAccessClass`：バケットへのアクセスのプロビジョニング方法を定義します。
- `BucketAccess`：1つ以上のバケットの認証情報を要求します。

Tridentは、以下のオブジェクトストレージドライバでCOSIをサポートしています：

- StorageGRID：一般提供されています。
- ONTAP S3：テクニカルプレビュー。

TridentはS3プロトコルをサポートしています。初期リリースでは、パス形式のS3アドレス指定のみがサポートされています。

要件および制限事項

- `TridentOrchestrator`カスタムリソース（TORC）で `enableConcurrency`と `enableCOSI`の両方を有効にします。COSI はデフォルトでは有効になっていません。
- コントローラの同時実行を有効にする前に、設定されたすべてのTridentバックエンドがサポートされているドライバを使用していることを確認してください。サポートされていないバックエンドドライバを使用すると、Tridentが起動しなくなります。"コントローラーのスケーラビリティを有効にする前に"を参照してください。
- Trident COSIをコミュニティCOSIコントローラと並行して実行しないでください。
- 最初のリリースでは、 `objectstorage.k8s.io/v1alpha2` COSI APIを使用します。
- StorageGRID `TridentObjectStore`リソースは、長期間有効なテナントのルート認証情報を使用します。各`TridentObjectStore`は1つのStorageGRIDテナントに接続します。複数のリソースを同じテナントに接続できます。認証情報はTridentネームスペースのKubernetes Secretに保存し、セキュリティ要件に従って保護してください。
- StorageGRID `BucketAccess`は、最大8つのバケットをサポートします。

並行コントローラ操作に関する一般的な情報については、"コントローラのスケーラビリティ"を参照してください。

COSI を有効にする

両方のオプションを `TridentOrchestrator`仕様で設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
```

```
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
  enableCOSI: true
```

TORCを適用するか、通常のオペレータワークフローを使用して既存のリソースを更新します。TORCの設定の詳細については、"Tridentオペレータのインストールをカスタマイズ"を参照してください。

StorageGRIDオブジェクトストアを設定する

TridentネームスペースにStorageGRIDテナントの認証情報を含むKubernetes Secretを作成します：

```
apiVersion: v1
kind: Secret
metadata:
  name: tenant3
  namespace: trident
type: Opaque
stringData:
  username: <tenant-root-username>
  password: <tenant-root-password>
```

Secretを参照する `TridentObjectStore`を作成します。サンプルのエンドポイントとアカウントIDを、ご使用のStorageGRID環境の値に置き換えます：

```
apiVersion: trident.netapp.io/v1
kind: TridentObjectStore
metadata:
  name: trident-tenant3
  namespace: trident
spec:
  accountId: "<account-id>"
  credentials:
    secretName: tenant3
  labels:
    tenantInstance: three
  managementEndpoint: https://<storagegrid-management-endpoint>
  objectStoreDriver: storage-grid
  s3Endpoint: https://<storagegrid-s3-endpoint>
```

両方のリソースを適用します：

```
kubectl apply -f tenant-secret.yaml
kubectl apply -f trident-object-store.yaml
```

ONTAP S3オブジェクトストアを構成する

COSIを介したONTAP S3のサポートは、Trident 26.10ではテクニカルプレビューであり、本番環境での使用はサポートされていません。設定項目や動作は、一般提供開始前に変更される場合があります。

TridentネームスペースにONTAP S3ユーザー認証情報を含むKubernetes Secretを作成し、そのSecretを参照する `TridentObjectStore`を作成します。プレースホルダーをONTAP S3環境の値に置き換えます：

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-s3-credentials
  namespace: trident
type: Opaque
stringData:
  username: "<user>"
  password: "<password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentObjectStore
metadata:
  name: ontap-s3
  namespace: trident
spec:
  objectStoreDriver: ontap-s3
  objectStoreName: ontap-object-store
  managementEndpoint: "<management-endpoint>"
  svmName: "<svm-name>"
  storagePrefix: trident
  credentials:
    secretName: ontap-s3-credentials
```

`storagePrefix`はオプションであり、省略した場合はデフォルトで `trident`になります。Secretと `TridentObjectStore`を適用します：

```
kubectl apply -f ontap-s3-object-store.yaml
```

BucketClassを作成してバケットをリクエストする

`BucketClass`は、ラベルセレクタ、オブジェクトストアタイプ、またはその両方を使用してオブジェクトストアを選択できます。選択パラメータがない場合、Tridentは設定済みのすべてのオブジェクトストアから選択できます。

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketClass
metadata:
  name: sg-delete
spec:
  driverName: cosi.trident.netapp.io
  deletionPolicy: Delete
  parameters:
    selector: tenantInstance=three
    objectStoreType: storage-grid
```

クラスを使用する`BucketClaim`を作成します：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketClaim
metadata:
  name: claim-a1
  namespace: default
spec:
  bucketClassName: sg-delete
  protocols:
    - S3
```

Tridentがクレームを処理する際、一致するオブジェクトストアを選択します。複数のストアが一致する場合、Tridentは一致するストア間でバケットを分散させるために、ランダムに1つを選択します。

ワークロードにバケットへのアクセス権限を付与する

アクセスプロビジョニングを定義する`BucketAccessClass`を作成します：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketAccessClass
metadata:
  name: sg-access
spec:
  authenticationType: Key
  driverName: cosi.trident.netapp.io
```

```
  multiBucketAccess: MultipleBuckets
```

バケットクレームを参照する `BucketAccess`を作成します：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketAccess
metadata:
  name: sg-claim-a1
  namespace: default
spec:
  bucketAccessClassName: sg-access
  bucketClaims:
    - accessMode: ReadWrite
      accessSecretName: sg-claim-a1
      bucketClaimName: claim-a1
  protocol: S3
```

Tridentは、バケットへのアクセス情報を含むKubernetes Secretを作成します。ワークロードは、標準のKubernetes Secretメカニズムを使用してこのSecretを利用できます。生成されたSecretには、バケット識別子、エンドポイント、リージョン、アクセスキーなどのS3接続の詳細が含まれます。生成されたSecretを、許可されていないユーザーやワークロードに公開しないでください。

生成されたシークレットには、 `COSI_S3_ACCESS_KEY_ID`、 `COSI_S3_ACCESS_SECRET_KEY`、 `COSI_S3_ADDRESSING_STYLE`、 `COSI_S3_BUCKET_ID`、 `COSI_S3_ENDPOINT`、 `COSI_S3_REGION`などのキーが含まれます。

バックエンドの構成と管理

Tridentバックエンドの設定

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

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

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

Azure NetApp Files の特長

Azure NetApp Files バックエンドを設定します

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

サポートされている Azure クラウド環境

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

サポートされている Azure クラウドは次のとおりです：

- Azure コマーシャル
- Azure Government（Azure Government / MAG）
- Azure China

Trident を導入する場合、または Azure NetApp Files バックエンドを設定する場合は、Azure Resource Manager と認証エンドポイントが Azure クラウド環境と一致していることを確認してください。

Azure NetApp Files ドライバのサポートを確認

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

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

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

考慮事項の確認

- Azure NetApp Filesは、50 GiB未満のボリュームをサポートしていません。Tridentは、より小さいボリュームが要求された場合、50 GiBのボリュームを作成します。
- Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- 非商用 Azure クラウドでの Azure NetApp Files の導入には、クラウド固有の Azure Resource Manager と認証エンドポイントが必要です。Trident とすべてのバックエンド構成で、Azure クラウド環境に適したエンドポイントを使用していることを確認してください。

AKS のマネージド ID を使用する

Tridentは、AKSクラスタ用の"管理対象ID"をサポートしています。

`tridentctl`を使用して `Azure NetApp Files`

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

マネージド ID を使用するには、次のものが必要です：

- AKSを使用して導入されるKubernetesクラスタ
- AKS Kubernetes クラスタで設定された管理対象 ID
- Tridentがインストールされ、 `cloudProvider`が `"Azure"`に設定されている

Trident オペレータ

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

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

Helm

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

```
helm install trident trident-operator-100.2610.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

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

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

AKS のワークロード ID を使用する

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

`tridentctl`を使用して `Azure NetApp Files` バックエンドを作成または管理する場合は、正しい `Azure`

クラウド環境用に構成されていることを確認してください。

ワークロード ID を使用するには、次のものが必要です：

- AKSを使用して導入されるKubernetesクラスタ
- AKS KubernetesクラスタIに設定されたワークロードIDとoidc-issuer
- Tridentがインストールされ、 `cloudProvider`が `"Azure"`に設定され、 `cloudIdentity`がワークロードIDの値に設定されている

Trident オペレータ

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

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

Helm

次の環境変数を使用して、*クラウド プロバイダ（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`を `$CP`を使用して設定し、 `cloudIdentity`を `$CI`を使用して設定します：

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

<code>tridentctl</code>

次の環境変数を使用して、*クラウド プロバイダ*および*cloud identity*フラグの値を設定します：

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

```
xxxxxxxxxxx"
```

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

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

Azure NetApp Files バックエンドを設定する準備をします

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

サポートされている **Azure** クラウド環境

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

サポートされている Azure クラウドは次のとおりです：

- Azure コマーシャル
- Azure Government（Azure Government / MAG）
- Azure China

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

NFSボリュームと**SMB**ボリュームの前提条件

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

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

- `subscriptionID`、 `tenantID`、 `clientID`、 `location`および `clientSecret` AKS クラスタで管理対象IDを使用する場合はオプションです。
- `tenantID`、 `clientID`および `clientSecret` は、AKSクラスタでクラウドIDを使用する場合はオプションです。
- 非商用 Azure クラウドでの Azure NetApp Files の導入には、クラウド固有の Azure Resource Manager と認証エンドポイントが必要です。Trident とすべてのバックエンド構成で、Azure クラウド環境に適したエンドポイントを使用していることを確認してください。

- 容量プール。を参照してください "Microsoft：Azure NetApp Files 用の容量プールを作成します"。

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

    ▼ カスタムコントリビュータロール

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

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

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

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

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/re
ad",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/wr
ite",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/de
lete",

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

```
            "Microsoft.Network/virtualNetworks/read",
            "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/delete",
            "Microsoft.Features/features/read",
            "Microsoft.Features/operations/read",
            "Microsoft.Features/providers/features/read",
            "Microsoft.Features/providers/features/register/action",

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

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

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

SMBボリュームに関するその他の要件

SMBボリュームを作成するには、以下が必要です。

- Active Directoryが設定され、Azure NetApp Files に接続されています。を参照してください "Microsoft：Azure NetApp Files のActive Directory接続を作成および管理します"。
- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2022を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Azure NetApp FilesがActive Directoryに対して認証できるように、Active Directoryクレデンシャルを含む少なくとも1つのTridentシークレット。シークレットを生成するには `smbcreds`：

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

- Windowsサービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub: CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

Azure NetApp Files バックエンド構成のオプションと例

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

バックエンド構成オプション

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

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

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

パラメータ	説明	デフォルト
「 clientSecret 」を入力します。	アプリケーション登録からのクライアントシークレット AKSクラスタで管理IDまたはクラウドIDを使用する場合はオプションです。	
「サービスレベル」	「標準」、「プレミアム」、「ウルトラ」のいずれかです	"" （ランダム）
「ロケーション」	新しいボリュームを作成する Azure の場所の名前 AKSクラスタで管理IDが有効になっている場合はオプションです。	
`cloudConfiguration.cloudName`	バックエンドのAzureクラウド環境を識別します。Azure Government を使用する場合は、`AzureGovernment`に設定します。	
「 resourceGroups 」	検出されたリソースをフィルタリングするためのリソースグループのリスト	"[]" （フィルタなし）
「 netappAccounts 」のように入力します	検出されたリソースをフィルタリングするためのネットアップアカウントのリスト	"[]" （フィルタなし）
「 capacityPools	検出されたリソースをフィルタリングする容量プールのリスト	"[]" （フィルタなし、ランダム）
「 virtualNetwork 」	委任されたサブネットを持つ仮想ネットワークの名前	""
「サブネット」	「 icrosoft.Netapp/volumes` 」に委任されたサブネットの名前	""
「ネットワーク機能」	ボリュームのVNet機能のセット。`Basic`または `Standard`の場合があります。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。 `networkFeatures`を指定すると、この機能が有効になっていない場合、ボリュームのプロビジョニングが失敗します。	""

パラメータ	説明	デフォルト
「 nfsvMountOptions 」のように入力します	NFSマウントオプションのきめ細かな制御。SMBボリュームの場合は無視されます。NFSバージョン4.1を使用してボリュームをマウントするには、カンマ区切りのマウントオプションリストに`nfsvers=4`を含めて、NFS v4.1を選択します。ストレージクラス定義で設定されたマウントオプションは、バックエンド構成で設定されたマウントオプションをオーバーライドします。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例 : `\{"API":false," メソッド ":true," 検出 ":true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
`supportedTopologies`	このバックエンドでサポートされているリージョンとゾーンのリストを表します。詳細については、を参照してください "CSI トポロジを使用します"。	
`qosType`	QoSタイプ（自動または手動）を表します。	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。	`4 MiB/sec`

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

Azureクラウド環境について検討する（**26.06**）

26.06リリース以降、Tridentは複数のAzureクラウド環境でAzure NetApp Filesバックエンドの作成と管理をサポートします。

サポートされている Azure クラウドは次のとおりです：

- Azure コマーシャル
- Azure Government（Azure Government / MAG）
- Azure China

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

Azure Government バックエンドの場合は、バックエンド設定で `cloudConfiguration.cloudName`を設定します：

```
cloudConfiguration:
  cloudName: AzureGovernment
```

必要な権限とリソース

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

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

vNet がAzure NetApp Files (ANF) ストレージ アカウントとは異なるリソース グループにある場合は、バックエンドの resourceGroups リストを構成するときに、仮想ネットワークのリソース グループを指定します。

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

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

を入力します	の形式で入力し
サブネット	<resource group>/< 仮想ネットワーク >/< サブネット >

ボリュームのプロビジョニング

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

パラメータ	説明	デフォルト
「 exportRule 」	新しいボリュームに対するエクスポートルール `exportRule` CIDR表記のIPv4アドレスまたはIPv4サブネットの任意の組み合わせをカンマで区切って指定する必要があります。SMBボリュームでは無視されます。	"0.0.0.0/0 "
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスします	`true`、 `false` （明示的に設定）。
「 size 」	新しいボリュームのデフォルトサイズ	" 100G "
「 unixPermissions 」	新しいボリュームのUNIX権限（8進数の4桁）。SMBボリュームでは無視されます。	"" （プレビュー機能、サブスクリプションでホワイトリスト登録が必要）

構成例

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

▼ 最小限の構成

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

この構成は、Azure NetApp Filesの使用を開始して試している段階で、実際にはプロビジョニングするボリュームに対して追加の範囲を設定することが必要な場合に適しています。

```
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
```

```
  storageDriverName: azure-netapp-files
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
  tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
  clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
  clientSecret: SECRET
  location: eastus
```

▼ AKSの管理対象ID

このバックエンド構成では、 `subscriptionID`、 `tenantID`、 `clientID`および `clientSecret`は、管理対象IDを使用する場合はオプションです。

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

▼ AKSのクラウドID

このバックエンド構成では、 `tenantID`、 `clientID`および `clientSecret`は、クラウドIDを使用する場合はオプションです。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
```

```
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
  location: eastus
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
```

▼ 容量プールフィルタを使用した特定のサービスレベル構成

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

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

▼ 手動 **QoS** 容量プールを使用したバックエンドの例

このバックエンド構成では、Azureの `eastus`手動 QoS 容量プールのある場所。

```
---
version: 1
storageDriverName: azure-netapp-files
backendName: anf1
location: eastus
labels:
  clusterName: test-cluster-1
  cloud: anf
  nasType: nfs
defaults:
  qosType: Manual
storage:
  - serviceLevel: Ultra
    labels:
      performance: gold
    defaults:
```

```
      maxThroughput: 10
  - serviceLevel: Premium
    labels:
      performance: silver
    defaults:
      maxThroughput: 5
  - serviceLevel: Standard
    labels:
      performance: bronze
    defaults:
      maxThroughput: 3
```

▼ 高度な設定

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

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

▼ 仮想プール構成

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

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

▼ サポートされるトポロジ構成

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

```
---
```

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

ストレージクラスの定義

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

を使用した定義の例 **`parameter.selector`** フィールド

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

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

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

```
  selector: performance=silver
allowVolumeExpansion: true

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

SMBボリュームの定義例

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

▼ デフォルトネームスペースの基本設定

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

▼ ネームスペースごとに異なるシークレットを使用する

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

```
    csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

▼ ボリュームごとに異なるシークレットを使用する

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

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

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

Google Cloud NetAppボリューム

Google Cloud NetApp Volumes を設定

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

概要

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

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

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

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

これは次の環境に適用されます：

- Trident 26.02以降
- Google Kubernetes Engine（GKE）、またはGCP上で稼働するRed Hat OpenShiftクラスタ
- OpenShiftワークロードIDフェデレーション（WIF）にはTrident 26.10以降が必要です
- Google Cloud NetApp Volumes ストレージプール

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

設定を準備

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

ワークロードIDを設定する前に、必要なIDリソースを作成する権限を持つアカウントを使用してください。GKEの場合、Google Cloudサービスアカウントを作成および構成するためのGoogle Cloudの権限が必要です。OpenShiftの場合、これにはクラスタのWIF構成を作成または選択し、Google Cloud Workload Identity Poolとプロバイダを構成するための権限が必要です。

GKEワークロード**ID**を使用する

Google Cloud NetApp VolumesでGKEワークロードIDを使用するには、以下が必要です：

- Google Kubernetes Engine（GKE）を使用してデプロイされた Kubernetes クラスタ
- GKE クラスタでワークロード ID が有効になっており、ノード プールでメタデータ サーバーが有効になっている
- Google Cloud NetApp Volumes Admin ロールを持つ Google Cloud サービスアカウント(`roles/netapp.admin`または同等のカスタムロール
- Tridentがインストールされ、クラウド プロバイダが `GCP`に設定され、クラウドIDアノテーションが設定されている

Trident オペレータ

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

```
apiVersion: trident.netapp.io/v1
```

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

Helm

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

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

Tridentctl

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

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

OpenShiftでワークロード**ID**フェデレーションを使用する

Trident 26.10以降では、Google Cloud Workload Identity Federation（WIF）を使用して、GCP上で動作するOpenShiftクラスター上のGoogle Cloud NetApp Volumesドライバーを認証できます。これはセルフマネージドのOpenShift Container Platform（OCP、IPIデプロイメントを含む）およびOpenShift Dedicated（OSD）に適用され、NAS (`google-cloud-netapp-volumes`）とSAN (`google-cloud-netapp-volumes-san`）の両方のドライバーをサポートします。

WIFは、プロジェクションされたKubernetesサービスアカウントトークンを、有効期限の短いGoogle Cloud認証情報と交換します。Tridentはこれらの認証情報を使用してGoogle Cloudサービスアカウントの権限を借用します。そのサービスアカウントのIAMロールによって、Google Cloud NetApp Volumes API呼び出しが承認されます。これにより、バックエンドの設定に長期間有効なサービスアカウントキーを保存する必要がなくなります。

OpenShiftクラスタの準備

OSDの場合、クラスター作成時にGCP WIF構成を作成または選択します。例えば、OpenShift Cluster Manager CLI で作成できます：

```
ocm gcp create wif-config --name <wif-name> --project <gcp-project-id>
```

クラスタ認証タイプとしてWorkload Identity Federationを選択し、WIF構成をクラスタに関連付けます。セルフマネージドOCPの場合は、クラスタのサービスアカウントトークン発行者を信頼するGoogle Cloud OIDCプロバイダを設定します。クラスタのプロビジョニングとトークンプロジェクションの手順は、OSDとセルフマネージドOCPで異なります。該当するRed Hatのガイダンスに従い、プロジェクションされたトークンを`/var/run/secrets/wif-token/token`のTridentコントローラで利用できるようにします。

OSDのWIF認証は、クラスターを作成する際に設定されます。TridentのWIFオーディエンスは別のインストール設定です。TridentでWIFを有効にするためにクラスターを再作成する必要はありません。

Tridentをインストールまたは設定する前に、インストールに使用するOpenShift IDに、選択したインストール方法で必要となるクラスタ権限があることを確認してください。インストール後、コントローラとノードのポッドが正常であることを確認します：

```
oc whoami
oc auth can-i create namespaces
oc auth can-i create csidrivers.storage.k8s.io
oc auth can-i create customresourcedefinitions.apiextensions.k8s.io
oc auth can-i create securitycontextconstraints.security.openshift.io

oc -n trident get pods
oc -n trident rollout status deploy/trident-controller --timeout=300s
oc -n trident rollout status ds/trident-node-linux --timeout=300s
```

WIFオーディエンスを使用して**Trident**をインストールする

Tridentは、インストール時にオーディエンスを設定した場合にのみ、WIF用のトークンを投影します。Google Cloudプロバイダとバックエンドの`wipCredentialConfig.audience`で同じオーディエンスを使用してください：

```
//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>
```

`cloudProvider`を `GCP`に設定し、選択したインストール方法を使用して`gcpWIFAudience`を設定します：

Tridentctl

```
tridentctl install \
  --cloud-provider GCP \
  --gcp-wif-audience "<audience>" \
```

```
  --namespace trident
```

Trident オペレータ

`TridentOrchestrator`仕様にこれらのフィールドを設定します：

```
spec:
  cloudProvider: GCP
  gcpWIFAudience: "<audience>"
```

Helm

チャートをインストールする際に、以下の値を設定してください：

```
helm install trident netapp-trident/trident-operator \
  --set cloudProvider=GCP \
  --set-string gcpWIFAudience="<audience>"
```

OpenShift WIFに対して、GKE `cloudIdentity` / `--cloud-identity`オプションも設定しないでください。インストール後にTridentコントローラのDeploymentにパッチを適用しないでください。オペレータが手動の変更を元に戻します。投影されたトークンは、バックエンドの `/var/run/secrets/wif-token/token`で利用できます。`trident-main`コンテナにはシェルが含まれていないため、ファイルを確認するには `oc exec`を使用する代わりに、コントローラDeploymentのボリューム構成を調べてください：

```
oc -n trident get deploy trident-controller -o yaml
```

Google Cloudの信頼とアクセス許可を設定する

クラスター用に、Google CloudワークロードIDプールとOIDCプロバイダーを設定します。プロバイダーの発行者はトークンの `iss`クレームと一致する必要があり、その属性マッピングは `assertion.sub`を`google.subject`にマッピングする必要があります。プロバイダーの許可されたオーディエンスは、投影されたトークンの `aud`クレームと一致する必要があります。

> OSDの場合、共有プロバイダー上のオーディエンスリストを置き換えたり、制限したりしないでください。 `allowedAudiences`が空の場合、Googleは、 `https://`プレフィックスの有無にかかわらず、プロバイダの正規リソース名を受け入れます。 `--allowed-audiences`を設定すると、そのデフォルトのリストが置き換えられ、他のOSDコンポーネントがトークンを交換できなくなる可能性があります。プロバイダーが既に明示的なリストを持っている場合にのみオーディエンスを追加し、既存の値はすべて保持します。OSDの共有プロバイダーに属性条件を追加しないでください。Google Cloudの"OIDCプロバイダーリファレンス"を参照してください。

パブリック `.well-known/openid-configuration`およびJWKSエンドポイントは、OIDCディスカバリを使用する場合にのみ必要です。Googleは、 `jwksJson`または `--jwk-json-path`を使用してJWKSを直接アップロードすることもサポートしています。

Tridentが権限を借用するGoogle Cloudサービスアカウントの `roles/iam.workloadIdentityUser`ロールをWIFプリンシパルに付与します。トークンサブジェクトが `system:serviceaccount:trident:trident-controller`の場合、プリンシパルは次の形式になります：

```
principal://iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/subject/system:serviceaccount:trident:trident-controller
```

Google Cloud NetApp Volumes リソースを含むプロジェクトで、なりすましサービスアカウントに `roles/netapp.admin`、または必要な権限を持つ同等のカスタムロールを付与します。これらは別々の権限付与です：1つ目はなりすましを許可し、2つ目はストレージAPI操作を承認します。

トークンのオーディエンスとプロバイダの設定が一致しない場合、Google Security Token Serviceは `invalid_grant`オーディエンスエラーで交換を拒否する場合があります。プロバイダの設定を次で確認します：

```
gcloud iam workload-identity-pools providers describe <provider-id> \
  --project=<wif-pool-project-id> \
  --location=global \
  --workload-identity-pool=<pool-id> \
  --format="yaml(oidc.allowedAudiences,oidc.issuerUri,attributeMapping)"
```

トークン構成を診断する際は、トークンの `iss`、 `aud`、および `sub`クレームをプロバイダおよびIAM構成と比較してください。投影されたトークン全体を共有したり、ログに記録したりしないでください。

Google Cloud NetApp Volumes ストレージを準備する

バックエンドが使用するリージョンにストレージプールを作成または特定します。NASバックエンドにはFILEプールが必要です。SANバックエンドは、"ブロックストレージ（iSCSI）を設定する"で説明されているFlex Unifiedプールを使用します。プールのVPCおよびプライベートサービスアクセス（PSA）の設定が、クラスタネットワークに適していることを確認してください。

例えば、プールのPSA範囲を選択する前に、ネットワークの既存のグローバルVPCピアリング範囲を一覧表示します：

```
gcloud compute addresses list \
  --project <gcnv-project-id> \
  --global \
  --filter="purpose=VPC_PEERING AND network:<network-name>" \
  --format="table(name,address,prefixLength,status,network,purpose)"
```

`location`でプールのリージョンを使用し、必要に応じて `storagePools`で完全なリソースパスを使用してプールを明示的に選択します。リージョンとプールの場所の不一致、誤ったネットワーク、またはプールが利用できない場合、認証が成功しても検出やプロビジョニングが妨げられる可能性があります。

Tridentバックエンドを設定する

プロバイダー構成と `wipCredentialConfig.audience`の両方で、予測されるトークンオーディエンスを使用します。 `projectNumber`フィールドは、Google Cloud NetApp Volumesプロジェクトを識別します。オーディエンスは、Workload Identityプールを含むプロジェクト番号を使用します。この例のプレースホルダーを、お客様の環境の値に置き換えてください：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-nas-wif
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
    network: "<vpc-network>"
    serviceLevel: Premium
  wipCredentialConfig:
    type: external_account
    audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
    subject_token_type: urn:ietf:params:oauth:token-type:jwt
    token_url: https://sts.googleapis.com/v1/token
    credential_source:
      file: /var/run/secrets/wif-token/token
    service_account_impersonation_url:
https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken
```

SANバックエンドの場合は、 `google-cloud-netapp-volumes-san`を `storageDriverName`および該当するFlex Unifiedプールの設定として使用します。WIF認証情報の設定は、それ以外は両方のドライバで同じです。

バックエンドを適用し、その監視状態を確認します：

```
oc apply -f <backend-file>.yaml
oc -n trident get tridentbackendconfig <backend-name> -o yaml
```

プロビジョニングをテストする前に、バックエンドが `Bound`であり、最後の操作が成功したことを確認します。ライブの `spec`および `status`を検査します。以前に適用された構成の注釈は、その後の変更を反映して

いない場合があります。WIFを使用する場合は、静的な `apiKey`認証情報を含めないでください。

エンドツーエンドのプロビジョニングを確認する

バックエンドをターゲットとするStorageClassを作成し、次に小さなテスト用PVCを作成します。たとえば、NAS StorageClassはバックエンドラベルを選択できます：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-wif-smoke
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  selector: "auth=wif"
allowVolumeExpansion: true
```

一致する `auth: wif`ラベルをNASバックエンドのストレージエントリに追加し、 `gcnv-wif-smoke`を使用してPVCを作成し、 `Bound`に到達することを確認します。SANスモークテストの場合は、 `google-cloud-netapp-volumes-san`、SANドライバの設定、および適切なFlex Unifiedプールを使用します。完了したら、テストPVCとStorageClassを削除します。

よくある障害のトラブルシューティング

現象	チェックする項目
`invalid_grant`またはオーディエンスのミスマッチ	投影されたトークンの `aud`を、プロバイダーが許可したオーディエンスおよび `wipCredentialConfig.audience`と比較します。
`iam.serviceAccounts.getAccessToken`アクセスが拒否されました	Tridentコントローラの `sub`の正確なWIFプリンシパルが、なりすましサービスアカウントに対する `roles/iam.workloadIdentityUser`を持っていることを確認します。
`netapp.storagePools.list`アクセスが拒否されました	なりすましサービスアカウントが、正しいプロジェクトに対して、`roles/netapp.admin`などのGoogle Cloud NetApp Volumesの権限を持っていることを確認してください。
プールが見つからない、またはプールが見つかりません	バックエンド `location`、プールのリージョンとタイプ、ネットワーク、PSA構成、および明示的に構成されたストレージプールのリソースパスを確認してください。
バックエンドはバインドされていません	基盤となる認証、権限、またはプール検出のエラーについて、Tridentコントローラのログと`TridentBackendConfig`のステータスを確認します。

バックエンドの調整エラーについてコントローラーのログを検査するには：

```
oc -n trident logs deploy/trident-controller -c trident-main --tail=400
```

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

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

ドライバの詳細

Trident は、Google Cloud NetApp Volumes から NAS ストレージをプロビジョニングするための `google-cloud-netapp-volumes`ドライバを提供します。

ドライバーは次のアクセス モードをサポートしています：

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

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

Trident NAS バックエンドを設定する

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

NAS ボリュームのプロビジョニング

NASボリュームは、 `google-cloud-netapp-volumes`バックエンドを使用してプロビジョニングされ、NFSお

よびSMBプロトコルをサポートします。

NFS ボリューム用の StorageClass

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

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

SMBボリューム用のStorageClass

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

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

PersistentVolumeClaim の例（RWX）

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

```
  storageClassName: gcnv-nfs
```

PersistentVolumeClaim の例（**RWO**）

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

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

SAN ワークロード用に **Google Cloud NetApp Volumes** を設定

`google-cloud-netapp-volumes-san` ドライバを使用して、Google Cloud NetApp Volumes Flex Unifiedストレージプールから iSCSIブロックボリュームをプロビジョニングします。

SANドライバはブロックワークロードのみをサポートします。NASボリュームをプロビジョニングするには、 `google-cloud-netapp-volumes`ドライバを使用します。

概要

このドライバは、Flex Unifiedストレージプールからボリュームをプロビジョニングし、それらをiSCSIブロックデバイスとしてKubernetesノードに提示します。

このドライバーは以下の環境をサポートしています：

- Trident 26.02以降
- Google Kubernetes Engine（GKE）、またはGCP上で稼働するRed Hat OpenShiftクラスタ
- Google Cloud NetApp Volumes Flex統合ストレージプール
- iSCSIベースのワークロード

Trident 26.10以降、SANドライバは、GCP上で実行される自己管理型のOpenShift Container Platform（OCP、IPIデプロイを含む）およびOpenShift Dedicated（OSD）クラスタでのGoogle Cloud Workload Identity Federation（WIF）をサポートします。WIFの前提条件とバックエンド構成については、"OpenShiftでワークロードIDフェデレーションを使用する"を参照してください。このページのSANドライバ名とFlex Unifiedストレージ設定を使用してください。

SAN用にWIFを構成する

WIFは、プロジェクションされたKubernetesサービスアカウントトークンを、有効期限の短いGoogle Cloud認証情報と交換します。Tridentは、Google Cloud NetApp Volumes APIを呼び出す際に、これらの認証情報を使用してGoogle Cloudサービスアカウントになりすまします。

OpenShiftクラスタ用にGoogle Cloud Workload Identity PoolとOIDCプロバイダを設定します。プロバイダの発行者はトークンの `iss`クレームと一致し、その属性マッピングは `assertion.sub`を `google.subject`にマッピングし、許可されたオーディエンスはトークンの `aud`クレームと一致する必要があります。

Tridentが権限を借用するGoogle Cloudサービスアカウントの `roles/iam.workloadIdentityUser`ロールを、`Trident`コントローラサービスアカウントの`WIF`プリンシパルに付与します。デフォルトの`Trident`コントローラサービスアカウントの場合、トークンのサブジェクトは `system:serviceaccount:trident:trident-controller`です。インストール環境に設定されたサブジェクトを使用してください。`Google Cloud NetApp Volumes`リソースを含むプロジェクトで、権限を借用するサービスアカウントに `roles/netapp.admin`、または同等の権限を付与します。

投影されたトークンは、 `/var/run/secrets/wif-token/token`でTridentコントローラが利用できる必要があります。 `wipCredentialConfig.audience`を、トークンおよびプロバイダの許可されたオーディエンスと同じオーディエンスに設定します。プレースホルダをお客様の環境の値に置き換えて、以下のSANバックエンドの `spec`に次のスタンザを追加します：

```
wipCredentialConfig:
  type: external_account
  audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
  subject_token_type: urn:ietf:params:oauth:token-type:jwt
  token_url: https://sts.googleapis.com/v1/token
  credential_source:
    file: /var/run/secrets/wif-token/token
  service_account_impersonation_url:
https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken
```

WIFの設定は、どちらのGCNVドライバでも同じです。SANバックエンドは引き続き `google-cloud-netapp-volumes-san`とサポートされているFlex Unifiedストレージプールを使用する必要があります。

Flex Unified ストレージプール

Flex Unifiedストレージプールは、iSCSIプロトコルを使用してブロックストレージを提供し、SANプロビジョニングに必要です：

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

バックエンド `location`は、Flex Unifiedプールのリージョンと一致している必要があります。Tridentがプールを見つけられない場合は、リージョン、ネットワーク、およびプールのリソースパスを確認してください。

Trident SANバックエンドを設定する

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

StorageClassを作成します

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

バックエンドではなく、StorageClassでファイルシステムの種類を指定します。

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

このドライバは、以下のファイルシステムタイプをサポートしています：

- `ext4`（デフォルト）
- `ext3`
- `xfs`

SANドライバはFlexサービスレベルのみをサポートし、 `exportRule`、 `unixPermissions`、 `nasType`、 `snapshotDir`、 `nfsMountOptions`などのNAS固有のバックエンドパラメータや

> 階層化関連の設定は使用しません。

WIFプロビジョニングを確認する

バックエンドを適用し、TridentBackendConfigがバインドされ、最後の操作が成功したことを確認します：

```
oc apply -f <backend-file>.yaml
oc -n trident get tridentbackendconfig <backend-name> -o yaml
```

このページの `gcnv-san` StorageClass を使用してPVCを作成し、 `Bound` に到達することを確認します。これにより、WIF交換、サービスアカウントの権限借用、Google Cloud NetApp Volumesの権限、およびSANプールの選択がまとめて検証されます。完了したら、テストPVCを削除してください。

バックエンドの調整が失敗した場合は、Tridentコントローラのログを確認してください：

```
oc -n trident logs deploy/trident-controller -c trident-main --tail=400
```

よくあるエラーについては、以下を確認してください：

- `invalid_grant`オーディエンスエラーの場合は、トークンの `aud`クレームをOIDCプロバイダの許可されたオーディエンスおよび `wipCredentialConfig.audience`と比較します。
- `iam.serviceAccounts.getAccessToken`権限エラーが発生した場合は、TridentコントローラのWIFプリンシパルが、権限を借用したサービスアカウントに対して `roles/iam.workloadIdentityUser`を持っていることを確認してください。
- `netapp.storagePools.list`権限エラーが発生した場合は、偽装されたサービスアカウントに、適切なプロジェクトに対する必要なGoogle Cloud NetApp Volumes権限があることを確認してください。
- プールが見つからない場合は、その場所と種類がサポートされているか、バックエンドのリージョンがプールのリージョンと一致しているか、バックエンドが正しいネットワークとプールを参照しているかを確認してください。

ブロックボリュームのプロビジョニング

ReadWriteOnce（RWO）

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

```
  storageClassName: gcnv-san
```

ReadWriteOncePod（RWOP）

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

ReadOnlyMany（ROX）

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

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

ReadWriteMany（RWX）— rawブロックのみ

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

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: gcnv-san-raw-rwx
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ブロックボリュームの動作

ブロック ボリュームは iSCSI LUN としてプロビジョニングされ、Kubernetes ノードにブロック デバイスとして提示されます。

ブロックボリューム：

- iSCSIプロトコルを使用する
- ファイルシステムとrawブロックのプレゼンテーションをサポート
- Tridentによって接続され、管理されます
- 複数の Kubernetes アクセスモードをサポート

アクセスモード

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

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

volumeModeの動作

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

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

サポートされている操作

`google-cloud-netapp-volumes-san`ドライバーを使用してプロビジョニングされたブロックボリュームは、次の機能をサポートします：

- 作成
- 削除
- クローン
- スナップショット
- サイズ変更
- インポート

ボリュームサイズのオーバーヘッド

Google Cloud NetApp Volumes ブロックボリュームには、内部メタデータのオーバーヘッドが含まれます。その結果、カーネルから見えるデバイスのサイズは、プロビジョニングされた容量よりも小さくなります。

観測されたオーバーヘッド：

- ボリューム作成時に約300 KiB
- サイズ変更後、最大約107MiB

Google Cloud NetApp Volumesは、GiB単位での容量割り当てを受け付けます。使用可能なデバイスのサイズがPVCの要求を満たすか、それを超えることを保証するために、Trident：

- 要求されたサイズを次の整数GiBに切り上げます。
- 1 GiBのバッファを追加します。

例：

- PVC要求量：100 GiB。
- プロビジョニング済み容量：101 GiB。
- アプリケーションから見える使用可能な容量：最低100 GiB。

Podの例

ファイルシステムにマウントされたブロックボリューム（**RWO**）

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

```
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-rwo
```

Raw ブロックデバイス (RWX)

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

アタッチとマウントの動作

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

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

> SAN ブロックボリュームでは、分散ロックや書き込み調整は提供されません。ブロックボリュームが複数のノードからアクセスされる場合（ReadWriteMany with `volumeMode: Block`）、アプリケーションまたはファイルシステムは同時実行性を管理する必要があります。

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

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

NFSまたはSMBボリュームの前提条件

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

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

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

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

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

Google Cloud NetApp Volumeのバックエンド構成オプションについて説明し、構成例を確認します。

バックエンド構成オプション

各バックエンドは、 1 つの Google Cloud リージョンにボリュームをプロビジョニングします。他のリージョンにボリュームを作成する場合は、バックエンドを追加で定義します。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	の値は `storageDriverName` 「google-cloud-netapp-volumes」と指定する必要があります。
backendName`	（オプション）ストレージバックエンドのカスタム名	ドライバ名 + "_" + API キーの一部
`storagePools`	ボリューム作成用のストレージプールを指定するオプションのパラメータ。	

パラメータ	説明	デフォルト
「 ProjectNumber 」	Google Cloud アカウントのプロジェクト番号。この値は、Google Cloudポータルのホームページにあります。	
「ロケーション」	TridentがGCNVボリュームを作成するGoogle Cloudの場所。リージョン間Kubernetesクラスタを作成する場合、で作成したボリュームは `location` 、複数のGoogle Cloudリージョンのノードでスケジュールされているワークロードで使用できます。リージョン間トラフィックは追加コストを発生させます。	
「 apiKey 」と入力します	ロールが割り当てられたGoogle Cloudサービスアカウントの`APIキー` `netapp.admin` 。このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。には `apiKey` 、、、、、の各キーのキーと値のペアを含める必要があります。 `type` `project_id` `client_email` `client_id` `auth_uri` `token_uri` `auth_provider_x509_cert_url`、、および `client_x509_cert_url`。	
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）
「サービスレベル」	ストレージプールとそのボリュームのサービスレベル。値は `flex`、 `standard`、 `premium`、または `extreme`です。	
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「ネットワーク」	GCNVボリュームに使用されるGoogle Cloudネットワーク。	
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： `{"api":false, "method":true}`トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`

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

ボリュームのプロビジョニングオプション

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

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

構成例

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

▼ 最小限の構成

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

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

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

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

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

▼ **SMBホリユウムノセツテイ**

```
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**フイルタヲシヨウシタセツテイ

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

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

```
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

▼ 仮想プール構成

このバックエンド構成では、1つのファイルに複数の仮想プールが定義されます。仮想プールは、セクションで定義し `storage` ます。さまざまなサービスレベルをサポートする複数のストレージプールがあり、それらを表すストレージクラスをKubernetesで作成する場合に役立ちます。仮想プールラベルは、プールを区別するために使用されます。たとえば、次の例では `performance` 、仮想プールを区別するためにラベルと `serviceLevel` タイプが使用されています。

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

詳細については、を参照してください "仮想プール"。

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

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  apiKey:
```

```
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

▼ **GKEのクラウドID**

GKEクラウドIDを使用してTridentをインストールすると、バックエンドはワークロードIDとして認証され、 `apiKey`ブロックは省略されます。クラウドIDは、両方のGoogle Cloud NetApp Volumesドライバーで同じように設定されます。唯一の違いは `storageDriverName`の値です。NAS（NFSまたはSMB）ボリュームには `google-cloud-netapp-volumes`を使用し、SAN（iSCSIブロック）ボリュームには `google-cloud-netapp-volumes-san`を使用します。

> ⓘ `google-cloud-netapp-volumes-san`ドライバーのクラウドID サポートには、Trident 26.06以降が必要です。SANボリュームはFlex Unifiedストレージプールからプロビジョニングされるため、 `serviceLevel`を `flex`に設定してください。

NASドライバー（<code>google-cloud-netapp-volumes</code>）

```
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
```

SANドライバ（<code>google-cloud-netapp-volumes-san</code>）

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcp-gcnv-san
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: '012345678901'
  network: gcnv-network
  location: us-west2
  serviceLevel: flex
  storagePool: pool-flex1
```

▼ サポートされるトポロジ構成

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

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

```
    topology.kubernetes.io/zone: asia-east1-a
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-b
```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
kubectl get tridentbackendconfig

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

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

```
tridentctl logs
```

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

ストレージクラスの定義

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

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

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

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

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

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

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

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

SMBボリュームの定義例

`node-stage-secret-name`、およびを使用する `nasType` `node-stage-secret-namespace`と、SMBボリュームを指定し、必要なActive Directoryクレデンシャルを指定できます。権限の有無にかかわらず、すべてのActive Directoryユーザ/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。

▼ デフォルトネームスペースの基本設定

```
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"
```

▼ ネームスペースごとに異なるシークレットを使用する

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

▼ ボリュームごとに異なるシークレットを使用する

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

`nasType: smb` SMBボリュームをサポートするプールでフィルタリングします。 `nasType: nfs` または `nasType: null` NFSプールに対してフィルタを適用します。

PVC定義の例PVCテイギノレイ

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: gcnv-nfs-pvc
spec:
  accessModes:
```

```
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs-sc
```

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

```
kubectl get pvc gcnv-nfs-pvc

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

Google Cloud NetApp Volumes の自動階層化を構成する

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

概要

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

Trident は、ボリューム作成時にのみ自動階層化設定を適用します。プロビジョニング後の変更は Trident 26.10 ではサポートされていません。

概念

自動階層化

自動階層化では、アクセス パターンに基づいて、アクセス頻度の低いデータをパフォーマンス層から容量層に移動します。データの移動は非同期的に行われ、即時には行われません。

階層化ポリシー

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

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

冷房日数

冷却日数は、データ ブロックが階層化の対象となる前に非アクティブのままにしておく必要がある最小日数を指定します。冷却日数は、階層化ポリシーが `auto`に設定されている場合にのみ適用されます。

構成モデル

構成スコープ

自動階層化は複数のスコープで設定できます：

- ストレージプールのスコープ 環境プールからプロビジョニングされたすべてのボリューム。
- ボリュームスコープ 環境単一のボリュームに適用されます（PersistentVolumeClaim アノテーションを使用）。

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

設定の優先順位

複数のスコープで同じ設定が定義されている場合、Tridentは次の優先順位を適用します：

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

より高い優先順位で定義された設定は、より低いレベルの値を上書きします。

Trident 26.10 でサポートされている機能

Trident 26.10は、Google Cloud NetApp Volumesの次の自動階層化機能をサポートしています：

- ボリュームのプロビジョニング時の自動階層化の有効化または無効化
- Tridentバックエンド構成での階層化ポリシーの定義
- PVC アノテーションを使用したボリュームごとの階層化ポリシーと冷却日数の上書き
- 自動階層化が有効になっているボリュームのクーリング日数の設定

Trident 26.10 でサポートされていない機能

次の処理はサポートされていません。

- ボリューム作成後の自動階層化設定の変更
- Kubernetes アップデートを使用して既存のボリュームの階層化ポリシーを変更する
- Trident 管理のプロビジョニング ワークフロー外での自動階層化設定の適用

バックエンド構成パラメータ

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

パラメータ	必須	説明
tieringPolicy	いいえ	ボリュームの階層化ポリシー（(`auto`または `none`）

パラメータ	必須	説明
tieringMinimumCoolingDays	いいえ	データが階層化されるまでの非アクティブ日数（範囲：2～183、デフォルト：31）

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

サポートされている注釈

PersistentVolumeClaim 注釈を使用すると、ボリュームごとに自動階層化設定をオーバーライドできます。

アノテーション	説明
`trident.netapp.io/tieringPolicy`	ボリュームの階層化ポリシーを上書きします
`trident.netapp.io/tieringMinimumCoolingDays`	ボリュームの冷却日数の値を上書きします

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

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

動作と制限

プロビジョニング動作

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

プラットフォームの制限

- 自動階層化は NAS ボリューム（NFS および SMB）でのみサポートされます。
- ブロックボリューム（iSCSI）は自動階層化をサポートしていません。

- Google Cloud NetApp Volumes ストレージ プールでは、Google Cloud で自動階層化が有効になっている必要があります。

サポートされている値

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

Google Cloud NetApp Volumes ONTAP モードストレージプールの設定

Trident 26.06以降、既存の `ontap-san`および `ontap-nas`バックエンドを設定して、Google Cloud NetApp Volumes（GCNV）のONTAPモードストレージプールにストレージをプロビジョニングできます。この機能はテクニカルプレビューです。

メモ	この機能は Trident 26.10 のテクニカルプレビューです。テクニカルプレビュー機能は、本番環境での使用はサポートされていません。機能、設定項目、およびサポートされるマトリックスは、一般提供開始（GA）前に変更される場合があります。

仕組み

Trident は既存の `ontap-san`および `ontap-nas`ドライバーロジックを再利用し、GCNV プロキシエンドポイントを介して ONTAP REST 呼び出しをルーティングします。このアプローチにより、直接 ONTAP バックエンドに使用するのと同じ運用ワークフローを維持しながら、GCNV ホスト上の ONTAP クラスターのストレージをプロビジョニングおよび管理できます。ONTAP モードでは、新しいストレージドライバーは導入されません。バックエンドごとに有効にするには、 `gcnv`構成ブロックを `ontap-san`または `ontap-nas`バックエンドに追加します。

プロビジョニングのパスは次のとおりです：

`PersistentVolumeClaim` → Trident (`ontap-san` または `ontap-nas`) → ONTAP REST クライアント → GCNV プロキシ → GCNV プール内の ONTAP クラスター

サポート対象範囲とサポート対象外範囲

このテクニカルプレビューでは、以下をサポートしています。

- ドライバー： `ontap-san` （iSCSI）および `ontap-nas` （NFSまたはSMB）。
- プロキシを介したライフサイクル操作：バックエンドの初期化、ボリュームの作成と削除、および関連する ONTAP REST フロー。
- 認証モデル：Workload Identity Pool （WIP）、サービスアカウントキー、およびApplication Default Credentials （ADC）フォールバック。

このテクニカルプレビューでは、以下の項目は対象外となります。

- `ontap-san-economy`、 `ontap-nas-economy`、および `ontap-nas-flexgroup`ドライバー。
- ASA r2 パーソナリティパス。
- ZAPIのフォールバック。ONTAPモードはONTAP RESTのみを使用します。

前提条件

ONTAPモードのバックエンドを設定する前に、以下のものを用意してください：

- Trident 26.06以降。
- 対象プロジェクトおよび対象場所に、GCNV ONTAP モードのストレージプールが必要です。
- プロキシ操作に必要な GCP IAM 権限を、お客様の環境に合わせて適切にスコープ設定します。
- `proxyURL`、 `projectNumber`、 `location`、および `poolID`を含むバックエンド構成。
- iSCSI SANワークフローの場合、プラットフォームのノード側iSCSIおよびマルチパスの前提条件が必要です。"ワーカーノードを準備します"を参照してください。

バックエンドの設定

A `TridentBackendConfig` には、Kubernetes Secret を参照する `credentials`が必要です。`ONTAP`モードの場合、シークレットは`GCP`プロキシアクセスの認証情報を提供しますが、`ONTAP` `managementLIF` / `username` / `password`の認証情報ではありません。認証方法によっては、サービスアカウントキーのフィールドが含まれる場合もあれば、空になる場合もあります。 `svm`はオプションです：各GCNV ONTAPモードプールには現在1つのSVMがあり、省略された場合、Tridentはプールからそれを取得します。

サービスアカウントキー認証には、ネイティブGCNVと同じパターンを使用します：非機密フィールドは`gcnv.apiKey`に、 `private_key`および `private_key_id`はシークレットに配置します。"Google Cloud NetApp Volumes の構成例"を参照してください。Workload Identity PoolまたはGKEクラウドID認証については、認証の例を参照してください。

ontap-san バックエンド

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-san
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  backendName: gcnv-ontap-san
  credentials:
    name: gcnv-sa-secret
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
  svm: "<svm-name>"
```

ontap-nas backend

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas
  credentials:
    name: gcnv-sa-secret
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
    apiKey:
      type: service_account
      project_id: "<project-id>"
      client_email: "<service-account-email>"
      client_id: "<client-id>"
      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: "<client-x509-cert-url>"
  svm: "<svm-name>"
```

秘密

```
apiVersion: v1
kind: Secret
metadata:
  name: gcnv-sa-secret
  namespace: trident
type: Opaque
stringData:
  private_key_id: "<private-key-id>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <private-key>
    -----END PRIVATE KEY-----
```

メモ

`TridentBackendConfig`仕様には `private_key`や`private_key_id`を指定しないでください。CRDによって拒否されます。ブロックストレージには、同じ `gcnv`とクレデンシャル パターンを持つ `ontap-san`を使用してください。

認証

Tridentは、プロキシアクセス用の認証情報を以下の順序で解決します：

1. Workload Identity プール（WIP）
2. サービスアカウントキー
3. アプリケーションのデフォルト認証情報（ADC）

このテクニカルプレビューでは、以下の手順に従ってください：

- Workload Identity Poolが利用可能な場合は、それを優先してください。
- バージョン管理されたバックエンドファイルに、生の秘密鍵を埋め込まないでください。認証情報をKubernetesシークレットに保存します。
- サービスアカウントに最小権限のIAMスコープを適用します。

`gcnv`ブロック内の`location`は、プールのロケーションと一致している必要があります。ゾーンプールにはゾーンを、リージョンプールにはリージョンを使用してください。

Workload Identity プール

`gcnv.wipCredential`は、`TridentBackendConfig`で設定された明示的な外部アカウントフェデレーションに使用します。OpenShiftでは、Tridentはトークンを`Å /var/run/secrets/wif-token/token`にプロジェクトします。これは TridentがÅOpenShift WIF用にプロジェクトする唯一のトークンパスであるため、`credential_source.file`に使用してください。link:gcnv.html#openshift-wif["Workload Identity フェデレーションを使用して Google Cloud NetApp Volumes を設定する"]の説明に従って、WIF オーディエンスを使用して Trident をインストールしてください。 `credentials`フィールドは引き続き Secret を参照しますが、Secret には `private_key`を含めてはなりません。

```
apiVersion: v1
kind: Secret
metadata:
  name: gcnv-ontap-empty
```

```
  namespace: trident
type: Opaque
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas-wip
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas-wip
  useREST: true
  nasType: nfs
  nfsMountOptions: nfsvers=4.1
  credentials:
    name: gcnv-ontap-empty
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
    wipCredential:
      type: external_account
      audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<wif-pool-
id>/providers/<provider-id>"
      subject_token_type: "urn:ietf:params:oauth:token-type:jwt"
      token_url: "https://sts.googleapis.com/v1/token"
      service_account_impersonation_url:
"https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken"
      credential_source:
        file: "/var/run/secrets/wif-token/token"
```

GKEクラウドアイデンティティ

GKEの場合は、"GKEワークロードIDを使用する"で説明されているように、インストール時にTridentのクラウドIDを設定します。バックエンドはApplication Default Credentials（ADC）を使用します。 `gcnv.apiKey`と`gcnv.wipCredential`の両方を省略してください。 `credentials`フィールドは引き続きSecretを参照しますが、そこに `private_key`を含めないでください。

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: gcnv-ontap-empty
  namespace: trident
type: Opaque
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas-wi
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas-wi
  useREST: true
  nasType: nfs
  nfsMountOptions: nfsvers=4.1
  credentials:
    name: gcnv-ontap-empty
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
```

検証および初期化動作

Tridentは初期化中にONTAPモードのバックエンドを検証し、部分的に初期化するのではなく、すぐにエラーで終了します。

必須の `gcnv`フィールドが欠落しているか、ドライバがONTAPモードでサポートされていないか、または認証情報の解決に失敗した場合、バックエンドの初期化は失敗します。

既知の制限事項

- この機能はテクニカルプレビュー版です。動作およびサポートマトリックスはGA前に変更される可能性があります。
- ONTAPモードではZAPIフォールバックは使用されません。
- プロトコルおよびノードの準備状況に関する要件は引き続き適用されます。たとえば、iSCSIワークフローでは、"ワーカーノードを準備します"で説明されているノード側の前提条件が必要です。
- 既存のONTAPおよびGCNV環境の制約は引き続き適用されます。

アップグレードと互換性

ONTAPモードは、 `gcnv`設定ブロックを介してバックエンドごとにオプトインします：

- 既存の非GCNV ONTAPバックエンドには影響はありません。

- 直接ONTAPバックエンドとONTAPモードのプロキシバックエンドを組み合わせた混在環境での導入は、バックエンドレベルの設定を通じてサポートされます。各バックエンドは個別に設定されます。
- ONTAPモードの使用を停止するには、影響を受けるバックエンドを削除または交換してください。グローバルな切り替えは不要です。

次の手順

TBCの前にシークレットを適用します。例に合わせるには、 `kubectl apply`と `-n trident`を使用します。

NetApp HCI または SolidFire バックエンドを設定します

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

Elementドライバの詳細

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

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

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

作業を開始する前に

Elementバックエンドを作成する前に、次の情報が必要になります。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	常に「SolidFire - SAN」
backendName`	カスタム名またはストレージバックエンド	「SolidFire _」+ストレージ（iSCSI）IPアドレス
「エンドポイント」	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP	
「 VIP 」	ストレージ（ iSCSI ）の IP アドレスとポート	
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット。	""
「 tenantname 」	使用するテナント名（見つからない場合に作成）	
「 InitiatorIFCace 」	iSCSI トラフィックを特定のホストインターフェイスに制限します	デフォルト
UseCHAP'	CHAPを使用してiSCSIを認証します。TridentはCHAPを使用します。	正しいです
「アクセスグループ」	使用するアクセスグループ ID のリスト	「Trident」という名前のアクセスグループのIDを検索します。
「タイプ」	QoS の仕様	
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します	"" （デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true}	null

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

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

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

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
labels:
```

```
    k8scluster: dev1
    backend: dev1-element-cluster
  UseCHAP: true
  Types:
    - Type: Bronze
      Qos:
        minIOPS: 1000
        maxIOPS: 2000
        burstIOPS: 4000
    - Type: Silver
      Qos:
        minIOPS: 4000
        maxIOPS: 6000
        burstIOPS: 8000
    - Type: Gold
      Qos:
        minIOPS: 6000
        maxIOPS: 8000
        burstIOPS: 10000
```

例**2**：のバックエンドとストレージクラスの設定 **`solidfire-san`** 仮想プールを備えたドライバ

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

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

以下に示すバックエンド定義ファイルの例では、すべてのストレージプールに対して特定のデフォルトが設定されています。これにより、が設定されます `type` シルバー。仮想プールは、で定義されます `storage` セクション。この例では、一部のストレージプールが独自のタイプを設定し、一部のプールが上記のデフォルト値を上書きします。

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
UseCHAP: true
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
```

```
    - Type: Silver
      Qos:
        minIOPS: 4000
        maxIOPS: 6000
        burstIOPS: 8000
    - Type: Gold
      Qos:
        minIOPS: 6000
        maxIOPS: 8000
        burstIOPS: 10000
  type: Silver
  labels:
    store: solidfire
    k8scluster: dev-1-cluster
  region: us-east-1
  storage:
    - labels:
        performance: gold
        cost: "4"
      zone: us-east-1a
      type: Gold
    - labels:
        performance: silver
        cost: "3"
      zone: us-east-1b
      type: Silver
    - labels:
        performance: bronze
        cost: "2"
      zone: us-east-1c
      type: Bronze
    - labels:
        performance: silver
        cost: "1"
      zone: us-east-1d
```

次のStorageClass定義は、上記の仮想プールを参照しています。を使用する `parameters.selector` 各ストレージクラスは、ボリュームのホストに使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プール内で定義された要素があります。

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

```
apiVersion: storage.k8s.io/v1
```

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

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

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

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

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

```
  fsType: ext4
```

詳細については、こちらをご覧ください

- "ボリュームアクセスグループ"

ONTAP SAN ドライバ

ONTAP SANドライバの概要

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

ONTAP SANドライバの詳細

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

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

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

警告

- 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。
- 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。
- ONTAPボリュームの自動サイズ調整は、 `ontap-san`および `ontap-san-economy`でサポートされています。ONTAP NASドライバではサポートされていません。詳細および制限事項については、"ONTAPボリュームの自動拡張を設定"を参照してください。

ユーザ権限

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

メモ

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

ONTAP内でTridentドライバが使用できる、より制限の厳しいロールを作成することは可能ですが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

NVMe/TCPに関するその他の考慮事項

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

- IPv6
- NVMeボリュームのSnapshotとクローン
- NVMeボリュームのサイズ変更
- Tridentの外部で作成されたNVMeボリュームをインポートして、そのライフサイクルをTridentで管理できるようにする

- NVMeネイティブマルチパス
- Kubernetesノードのグレースフルシャットダウンまたはグレースフルシャットダウン（24.06）

Tridentは以下をサポートしていません。

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

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

バックエンドに**ONTAP SAN**ドライバを設定する準備をします

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

要件

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

メモ	"ASA r2システム"ストレージ層の実装において他のONTAPシステム (ASA、 AFF、 FAS) と異なります。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"これ"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。

複数のドライバを実行し、 1 つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば 'ONTAP-SAN' ドライバを使用する「 -dev 」クラスと 'ONTAP-SAN-エコノミー 'one を使用する「デフォルト」クラスを設定できます

すべてのKubernetesワーカーノードに適切なiSCSIツールをインストールしておく必要があります。を参照してください "ワーカーノードを準備します" を参照してください。

ONTAPバックエンドの認証

Tridentには、ONTAPバックエンドの認証に2つのモードがあります。

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

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式を切り替えることができます。ただし、一度にサポートされる認証方法は1つだけです。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

警告	クレデンシャルと証明書の両方を*指定しようとすると、バックエンドの作成が失敗し、構成フ

ァイルに複数の認証方法が指定されているというエラーが表示されます。

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成または更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証の有効化

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

- clientCertificate ： Base64 でエンコードされたクライアント証明書の値。

- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された CA を使用する場合は、このパラメータを指定する必要があります。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

手順

1. クライアント証明書とキーを生成します。生成時に、 ONTAP ユーザとして認証するように Common Name （ CN ；共通名）を設定します。

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

2. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っている可能性があります。信頼できる CA が使用されていない場合は無視します。

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

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

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

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

4. ONTAP セキュリティ・ログイン・ロールが 'cert' 認証方式をサポートしていることを確認します

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

5. 生成された証明書を使用して認証をテストONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

既存のバックエンドを更新して、別の認証方法を使用したり、クレデンシャルをローテーションしたりできます。これはどちらの方法でも機能します。ユーザ名とパスワードを使用するバックエンドは証明書を使用する

ように更新できますが、証明書を使用するバックエンドはユーザ名とパスワードに基づいて更新できます。これを行うには、既存の認証方法を削除して、新しい認証方法を追加する必要があります。次に'必要なパラメータを含む更新されたbackend.jsonファイルを使用して'tridentctl backend updateを実行します

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

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

メモ	パスワードのローテーションを実行する際には、ストレージ管理者が最初に ONTAP でユーザのパスワードを更新する必要があります。この後にバックエンドアップデートが続きます。証明書のローテーションを実行する際に、複数の証明書をユーザに追加することができます。その後、バックエンドが更新されて新しい証明書が使用されるようになります。この証明書に続く古い証明書は、 ONTAP クラスタから削除できます。

バックエンドを更新しても、すでに作成されているボリュームへのアクセスは中断されず、その後のボリューム接続にも影響しません。バックエンドの更新が成功すると、TridentがONTAPバックエンドと通信し、以降のボリューム処理を処理できるようになります。

Trident用のカスタムONTAPロールの作成

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

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

ONTAP CLIノシヨウ

1. 次のコマンドを使用して新しいロールを作成します。

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

2. Tridentユーザのユーザ名を作成します。

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

3. ユーザにロールをマッピングします。

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

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

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

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

   b. の横にある矢印アイコン（→*）を選択します。
   c. [Roles]*で[+Add]*を選択します。
   d. ロールのルールを定義し、*[保存]*をクリックします。
2. ロールを**Trident**ユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。
   a. で**[**アイコンの追加**]**+*を選択します。
   b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。
   c. [保存（ Save ）] をクリックします。

詳細については、次のページを参照してください。

- "ONTAPの管理用のカスタムロール"または"カスタムロールの定義"
- "ロールとユーザを使用する"

双方向**CHAP**による接続の認証

Tridentでは、ドライバと `ontap-san-economy`ドライバの双方向CHAPを使用してiSCSIセッションを認証できます `ontap-san`。これには、バックエンド定義でオプションを有効にする必要があり `useCHAP`ます。に設定する `true`と、TridentはSVMのデフォルトのイニシエータセキュリティを双方向CHAPに設定し、ユーザ名とシークレットをバックエンドファイルに設定します。接続の認証には双方向 CHAP を使用するこ

とを推奨します。次の設定例を参照してください。

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

警告	「 useCHAP 」パラメータは、 1 回だけ設定できるブール型のオプションです。デフォルトでは false に設定されています。true に設定したあとで、 false に設定することはできません。

「 useCHAP=true' に加えて、「 chapInitiatorSecret 」、「 chapTargetInitiatorSecret 」、「 chapTargetUsername 」、および「 chapUsername 」フィールドもバックエンド定義に含める必要があります。シークレットは 'tridentctl update を実行してバックエンドを作成した後に変更できます

仕組み

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

- SVM で CHAP をセットアップします。
  - SVMのデフォルトのイニシエータセキュリティタイプがnone（デフォルトで設定）*で、*ボリュームに既存のLUNがない場合、Tridentはデフォルトのセキュリティタイプをに設定し `CHAP`、CHAPイニシエータとターゲットのユーザ名とシークレットの設定に進みます。
  - SVMにLUNが含まれている場合、TridentはSVMでCHAPを有効にしません。これにより、SVMにすでに存在するLUNへのアクセスが制限されなくなります。
- CHAP イニシエータとターゲットのユーザ名とシークレットを設定します。これらのオプションは、バックエンド構成で指定する必要があります（上記を参照）。

バックエンドが作成されると、Tridentは対応するCRDを作成し `tridentbackend`、CHAPシークレットとユーザ名をKubernetesシークレットとして格納します。このバックエンドでTridentによって作成されたすべてのPVSがマウントされ、CHAP経由で接続されます。

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 証明書を更新するには 'backend.json ファイルの CHAP パラメータを更新しますこれには 'CHAP シークレットを更新し 'tridentctl update コマンドを使用してこれらの変更を反映する必要があります

警告	バックエンドのCHAPシークレットを更新する場合は、を使用してバックエンドを更新する必

> 要があります `tridentctl`。ONTAP CLIまたはONTAPシステムマネージャを使用してストレージクラスタのクレデンシャルを更新しないでください。Tridentではこれらの変更を反映できません。

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

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

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

ONTAP のSAN構成オプションと例

ONTAP SAN ドライバーを作成してTridentインストールで使用します。このセクションでは、バックエンド構成の例と、バックエンドをStorageClassesにマッピングするための詳細について説明します。"ASA r2システム"は、ストレージ レイヤの実装において、他のONTAPシステム（ASA、AFF、およびFAS）とは異なります。これらの違いは、記載されている特定のパラメータの使用に影響します。"ASA r2 システムと他の ONTAP システムの違いについて詳しくは、こちらをご覧ください。"。Tridentバックエンド構成では、システムがASA r2であることを指定する必要はありません。 `ontap-san`を

`storageDriverName`として選択すると、TridentはASA r2またはその他のONTAPシステムを自動的に検出します。以下の表に示すように、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。

メモ	のみ `ontap-san`ドライバー (iSCSI、NVMe/TCP、および FC 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。

バックエンド構成オプション

26.06 リリース以降、ONTAP バランスの取れた配置を有効にすることで、ONTAP が新しいボリュームのアグリゲートを選択できるようになります。要件と例については、"ONTAPバックエンドでバランスの取れた配置を使用する" を参照してください。

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前。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。	`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	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタ管理LIFまたはSVM管理LIFのIPアドレス。 Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。 IPv6フラグを使用してTridentがインストールされている場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。 シームレスなMetroClusterスイッチオーバーについては、を参照してMetroClusterの例ください。 メモ 「vsadmin」のクレデンシャルを使用する場合はSVMのクレデン `managementLIF`シャル、「admin」のクレデンシャルを使用する場合はクラスタのクレデンシャル `managementLIF`を使用する必要があります。	"10.0.0.1 ","[2001：1234：abcd：：fe]"

パラメータ	説明	デフォルト
「重複排除	プロトコル LIF の IP アドレス。IPv6フラグを使用してTridentがインストールされている場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。* iSCSIの場合は指定しないでください。**Trident**は、を使用して**"ONTAP の選択的LUNマップ"**、マルチパスセッションの確立に必要な**iSCSI LIF**を検出します。が明示的に定義されている場合は、警告が生成され **`dataLIF`**ます。 MetroClusterの場合は省略してください。*を参照してくださいMetroClusterの例。	SVMの派生物です
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* MetroClusterの例。	SVM 「管理 LIF 」が指定されている場合に生成されます
「 useCHAP 」	CHAPを使用してONTAP SANドライバのiSCSIを認証します（ブーリアン）。バックエンドで指定されたSVMのデフォルト認証として双方向CHAPを設定して使用する場合は、Tridentのをに設定し `true`ます。詳細については、を参照してください "バックエンドにONTAP SANドライバを設定する準備をします" 。**FCP** または **NVMe/TCP** ではサポートされません。	「偽」
「 chapInitiatorSecret 」	CHAP イニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	""
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「 chapTargetInitiatorSecret 」	CHAP ターゲットイニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	""
「 chapUsername 」	インバウンドユーザ名。「 useCHAP = TRUE 」の場合は必須	""
「 chapTargetUsername 」	ターゲットユーザ名。「 useCHAP = TRUE 」の場合は必須	""
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""

パラメータ	説明	デフォルト
「ユーザ名」	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"。	""
「 password 」と入力します	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"。	""
'VM'	使用する Storage Virtual Machine	SVM 「管理 LIF 」が指定されている場合に生成されます
'toragePrefix'	SVMで新しいボリュームをプロビジョニングする際に使用されるプレフィックス。後から変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`
「集約」	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVMに割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバーの場合、このオプションは無視されます。割り当てられていない場合は、利用可能なアグリゲートのいずれかを使用してFlexGroupボリュームをプロビジョニングできます。 メモ SVM でアグリゲートが更新されると、Trident コントローラを再起動することなく、SVM をポーリングすることで Trident に自動的に反映されます。Trident でボリュームをプロビジョニングするために特定のアグリゲートを設定している場合、そのアグリゲートの名前が変更されたり SVM から移動されたりすると、SVM アグリゲートのポーリング中にバックエンドが Trident で失敗状態に移行します。バックエンドをオンラインに戻すには、アグリゲートを SVM 上に存在するものに変更するか、完全に削除する必要があります。 **ASA r2** システムには指定しないでください。 • `useBalancedPlacement`が `true`に設定されている場合は指定しないでください。*	""

パラメータ	説明	デフォルト
`useBalancedPlacement`	新しいボリュームのアグリゲート選択を ONTAP に委任します [Boolean]。ONTAP 9.19.1 以降が必要で、`useREST` を `true` に設定する必要があります。`aggregate` と組み合わせることはできません。 要件と例については、"ONTAPバックエンドでバランスの取れた配置を使用する"を参照してください。	「偽」
「AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。Amazon FSx for NetApp ONTAP バックエンドを使用している場合は、を指定しないで `limitAggregateUsage`ください。指定されたと `vsadmin`には `fsxadmin`、アグリゲートの使用量を取得してTridentを使用して制限するために必要な権限が含まれていません。**ASA r2** システムには指定しないでください。	""（デフォルトでは適用されません）
「limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、LUNで管理するボリュームの最大サイズも制限します。	""（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、200 です	`100`
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} トラブルシューティングを行い、詳細なログダンプが必要な場合を除き、は使用しないでください。	`null`

パラメータ	説明	デフォルト
「 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。 メモ: NVMe はONTAP REST API でのみサポートされ、ONTAPI (ZAPI) ではサポートされません。 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以降の場合は、それ以外の場合は false。
sanType	iSCSI、 nvme`NVMe/TCP、または `fcp`SCSI over Fibre Channel（FC；SCSI over Fibre Channel）に対してを選択します `iscsi。	iscsi 空白の場合

パラメータ	説明	デフォルト
formatOptions	`formatOptions`を使用して、ボリュームがフォーマットされるたびに適用される`mkfs`コマンドのコマンドライン引数を指定します。これにより、ボリュームをお客様の好みに合わせてフォーマットできます。デバイスパスを除き、mkfsコマンドのオプションと同様にformatOptionsを指定してください。例：「-E nodiscard」 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。**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エコノミーバックエンドでLUNを使用する場合の、要求可能な最大FlexVolサイズ。	""（デフォルトでは適用されません）
denyNewVolumePools	バックエンドがLUNを格納するために新しいFlexVolボリュームを作成することを制限します ontap-san-economy。新しいPVのプロビジョニングには、既存のFlexVolのみが使用されます。	

formatOptionsの使用に関する推奨事項

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

- **-E nodiscard (ext3、ext4):** mkfs 時にブロックを破棄しません (最初にブロックを破棄することは、ソリッド ステート デバイスおよびスパース / 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、ext3、ext4 ファイル システムに適用できます。
- **-K (xfs):** mkfs 時にブロックを破棄しません。このオプションは xfs ファイル システムに適用できます。

Active Directory の認証情報を使用して、バックエンド **SVM** に対して**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用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	"true" 指定されている場合は、 **`true` ASA r2** システムの場合。
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）。設定 **`none` ASA r2** システムの場合。	"なし"
「ナプショットポリシー」	使用するSnapshotポリシー。設定 **`none` ASA r2** システムの場合。	"なし"
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。TridentでQoSポリシーグループを使用するには、ONTAP 9 .8以降が必要です。共有されていないQoSポリシーグループを使用し、ポリシーグループが各コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
「スナップショット予約」	Snapshot用にリザーブされているボリュームの割合。**ASA r2** システムには指定しないでください。	次の場合は「0」 `snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ

パラメータ	説明	デフォルト
「暗号化」	新しいボリュームで NetApp Volume Encryption（NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、クラスター上で NVE のライセンスを取得し、有効化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、次を参照してください："TridentとNVEおよびNAEとの連携"	"false" 指定されている場合は、**true ASA r2** システムの場合。
`luksEncryption`	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。	"" 設定 **false ASA r2** システムの場合。
階層ポリシー	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

デフォルトが定義されている例を次に示します。

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

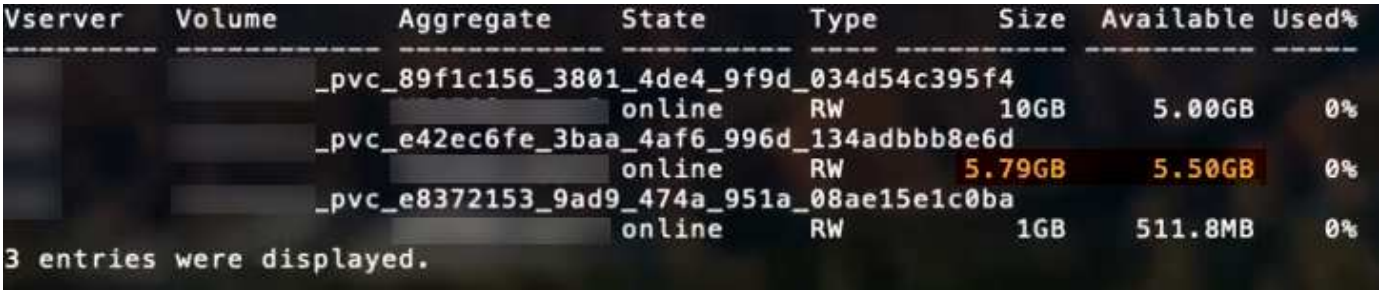
メモ	`ontap-san`ドライバを使用して作成されたすべてのボリュームについて、Trident は LUN メタデータに対応するために FlexVol に 10 パーセントの追加容量を加えます。LUN は、ユーザーが PVC で要求した正確なサイズでプロビジョニングされます。Trident は FlexVol に 10 パーセ

ントを追加します（ONTAP では利用可能なサイズとして表示されます）。ユーザーは、要求した通りの使用可能な容量を利用できるようになりました。この変更により、利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。

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

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

にTridentがFlexVolに追加する10%の容量です。 `snapshotReserve` = 5%、PVC要求 = 5 GiBの場合、ボリュームの合計サイズは5.79 GiB、使用可能なサイズは5.5 GiBです。 `volume show`コマンドを実行すると、次の例のような結果が表示されます。



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

最小限の設定例

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

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

▼ **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**

スイッチオーバーやスイッチバックの実行中にバックエンド定義を手動で更新する必要がないようにバックエンドを設定できます。 "SVMのレプリケーションとリカバリ"。

スイッチオーバーとスイッチバックをシームレスに実行するには、を使用してSVMを指定し `managementLIF`、パラメータは省略します `svm`。例：

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

▼ **ONTAP SANの経済性の例**

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

▼ 証明書ベースの認証の例

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

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

▼ 双方向**CHAP**の例

次の例では、 `useCHAP` をに設定します `true`。

ONTAP SAN CHAPの例

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

ONTAP SANエコノミーCHAPの例

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

▼ バランスの取れた配置例

この例では、ONTAP のバランスの取れた配置を有効にして、ONTAP が新しいボリュームのアグリゲートを選択できるようにします。バランスの取れた配置には、ONTAP 9.19.1 以降が必要です。詳細については、"ONTAPバックエンドでバランスの取れた配置を使用する"を参照してください。

```
---
version: 1
storageDriverName: ontap-san
backendName: san-balanced-placement
managementLIF: 10.0.0.1
```

```
svm: svm_iscsi
username: vsadmin
password: <password>
useBalancedPlacement: true
```

▼ **NVMe/TCP**の例

ONTAPバックエンドでNVMeを使用するSVMを設定しておく必要があります。これはNVMe/TCPの基本的なバックエンド構成です。

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

▼ **SCSI over FC（FCP）**の例

ONTAPバックエンドでFCを使用してSVMを設定しておく必要があります。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エコノミードライバのformatOptionsの例**

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

仮想プールを使用するバックエンドの例

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

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

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

▼ **ONTAP SANの例**

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
```

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

▼ **ONTAP SANの経済性の例**

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
```

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

▼ **NVMe/TCPの例**

```
---
```

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

バックエンドを **StorageClasses** にマッピングします

次のStorageClass定義は[仮想プールを使用するバックエンドの例]を参照しています。 `parameters.selector` フィールドを使用して、各StorageClassはボリュームのホストに使用できる仮想プールを指定します。このボリュームには、選択された仮想プールで定義された要素が含まれています。

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

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

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
```

```
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

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

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

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

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

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

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

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

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

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

NVMeデータLIFを更新する

NVMeデータLIFの更新を有効にすると、バックエンドを再作成したりTridentを再起動したりすることなく、追加、削除、または新しいアドレスが割り当てられたLIFを、公開されたNVMeボリュームで使用できるようになります。

概要

ONTAP SANバックエンドを追加すると、TridentはSVMのNVMeデータLIFを検出します。デフォルトでは、バックエンドを再度追加するかTridentを再起動するまで、TridentはそのLIFのセットを保持します。

Tridentの実行中にNVMeデータLIFを更新するには、 `enableDataLIFRefresh`を `true`に設定します。公開されたNVMeボリュームは、追加、削除、または新しいアドレスが割り当てられたLIFを使用するようになります。

このオプションはインストールパラメータであり、バックエンド構成パラメータではありません。デフォルトでは無効になっています。

NVMeデータLIFの更新は、Trident 26.10から利用可能です。

この機能は以下に適用されます：

- ユニファイドONTAPでの`ontap-san`と `sanType: nvme`。
- `ontap-san` ASA r2でNVMe/TCPを使用する。

注意	すべてのデータLIFを同時に削除または置き換えると、システムに支障をきたします。すべてのデータLIFを削除または交換する予定がある場合は、データアクセスが中断されないように、事前に代替のLIFが設置されていることを確認してください。

NVMeデータLIFのリフレッシュを有効にする

Trident オペレータ

新規インストールの場合は、 `TridentOrchestrator` spec で `enableDataLIFRefresh`を設定します：

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

マニフェストを適用します：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

既存のインストールの場合は、 `TridentOrchestrator`にパッチを適用します：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableDataLIFRefresh":true}}'
```

適用された値を確認してください：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableDataLIFRefresh}'
```

Helm

新規インストールの場合、Helm の値を設定します：

```
helm install trident netapp-trident/trident-operator \
  --namespace trident --create-namespace \
  --set enableDataLIFRefresh=true
```

既存のHelmインストールの場合、値を有効にしてリリースをアップグレードしてください：

```
helm upgrade trident netapp-trident/trident-operator \
  --namespace trident \
  --reuse-values \
  --set enableDataLIFRefresh=true
```

Helm values ファイルで `enableDataLIFRefresh: true`を設定し、 `-f <values-file>`を使用して `helm upgrade`に渡すこともできます。

Tridentctl

新規インストールの場合は、 `--enable-datalif-refresh`フラグを使用します：

```
tridentctl install --enable-datalif-refresh -n trident
```

既存の `tridentctl`インストールでオプションを有効にするには、Tridentをアンインストールし、フラグを指定して再インストールします。アンインストールする前に、"Trident をアンインストールします"の注意事項に従ってください。

```
tridentctl uninstall -n trident
tridentctl install --enable-datalif-refresh -n trident
```

メモ	`--enable-datalif-refresh`フラグはインストールオプションであり、 `tridentctl install --help`には表示されません。

ONTAP NAS ドライバ

ONTAP NASドライバの概要

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

ONTAP NASドライバの詳細

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

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

警告	• 使用 `ontap-san-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限"。 • 使用 `ontap-nas-economy` 永続的ボリュームの使用数が次の値よりも多いと予想される場合のみ "サポートされるONTAPの制限" および `ontap-san-economy` ドライバは使用でき

ません。

- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。
- ONTAP NASドライバでは、ONTAPボリュームの自動サイズ調整はサポートされていません。サポートされているSANドライバと構成の詳細については、"ONTAPボリュームの自動拡張を設定"を参照してください。

ユーザ権限

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

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

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

ONTAP内でTridentドライバが使用できる、より制限の厳しいロールを作成することは可能ですが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

ONTAP NASドライバを使用してバックエンドを設定する準備をします

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

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

要件

- すべての ONTAP バックエンドでは、Trident では少なくとも 1 つのアグリゲートを SVM に割り当てる必要があります。
- 複数のドライバを実行し、どちらか一方を参照するストレージクラスを作成できます。たとえば、を使用するGoldクラスを設定できます `ontap-nas` ドライバとを使用するBronzeクラス `ontap-nas-economy` 1つ。
- すべてのKubernetesワーカーノードに適切なNFSツールをインストールしておく必要があります。を参照してください "こちらをご覧ください" 詳細：

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

ONTAPバックエンドの認証

Tridentには、ONTAPバックエンドの認証に2つのモードがあります。

- Credential-based：このモードでは、ONTAPバックエンドに十分な権限が必要です。事前定義されたセキュリティログインロールに関連付けられたアカウントを使用することを推奨します。例： `admin` または `vsadmin` ONTAP のバージョンとの互換性を最大限に高めるため。
- 証明書ベース：このモードでは、TridentがONTAPクラスタと通信するために、バックエンドに証明書をインストールする必要があります。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式を切り替えることができます。ただし、一度にサポートされる認証方法は1つだけです。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

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

クレデンシャルベースの認証を有効にします

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

バックエンド定義の例は次のようになります。

YAML

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

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
```

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

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成と更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証を有効にします

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

- clientCertificate ： Base64 でエンコードされたクライアント証明書の値。
- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された CA を使用する場合は、このパラメータを指定する必要があります。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

手順

1. クライアント証明書とキーを生成します。生成時に、 ONTAP ユーザとして認証するように Common Name （ CN ；共通名）を設定します。

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

2. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っている可能性があります。信頼できる CA が使用されていない場合は無視します。

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

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

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

4. ONTAP セキュリティ・ログイン・ロールが 'cert' 認証方式をサポートしていることを確認します

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

5. 生成された証明書を使用して認証をテストONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。LIF のサービスポリシーが「 default-data-management 」に設定されていることを確認する必要があります。

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

```
#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
```

```
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

メモ	パスワードのローテーションを実行する際には、ストレージ管理者が最初に ONTAP でユーザのパスワードを更新する必要があります。この後にバックエンドアップデートが続きます。証明書のローテーションを実行する際に、複数の証明書をユーザに追加することができます。その後、バックエンドが更新されて新しい証明書が使用されるようになります。この証明書に続く古い証明書は、 ONTAP クラスタから削除できます。

バックエンドを更新しても、すでに作成されているボリュームへのアクセスは中断されず、その後のボリューム接続にも影響しません。バックエンドの更新が成功すると、TridentがONTAPバックエンドと通信し、以降のボリューム処理を処理できるようになります。

Trident用のカスタムONTAPロールの作成

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

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

ONTAP CLIノシヨウ

1. 次のコマンドを使用して新しいロールを作成します。

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

2. Tridentユーザのユーザ名を作成します。

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

3. ユーザにロールをマッピングします。

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

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

1. カスタムロールの作成：

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

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

b. の横にある矢印アイコン（→*）を選択します。

c. [Roles]*で[+Add]*を選択します。

d. ロールのルールを定義し、*[保存]*をクリックします。

2. ロールを**Trident**ユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。

a. で**[**アイコンの追加**]**+*を選択します。

b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。

c. [保存（ Save ）] をクリックします。

詳細については、次のページを参照してください。

- "ONTAPの管理用のカスタムロール"または"カスタムロールの定義"
- "ロールとユーザを使用する"

NFS エクスポートポリシーを管理します

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

Tridentでエクスポートポリシーを使用する場合は、次の2つのオプションがあります。

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

エクスポートポリシーを動的に管理

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

メモ	ダイナミックエクスポートポリシーを使用する場合は、Network Address Translation（NAT;ネットワークアドレス変換）を使用しないでください。NATを使用すると、ストレージコントローラは実際のIPホストアドレスではなくフロントエンドのNATアドレスを認識するため、エクスポートルールに一致しない場合はアクセスが拒否されます。

例

2 つの設定オプションを使用する必要があります。バックエンド定義の例を次に示します。

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

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

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

- `autoExportPolicy`がに設定されてい `true`ます。これは、Tridentが、このバックエンドを使用してSVMに対してプロビジョニングされたボリュームごとにエクスポートポリシーを作成し、アドレスブロックを使用してルールの追加と削除を処理すること `autoexportCIDRs`を示します `svm1`。ボリュームがノードに接続されるまでは、そのボリュームへの不要なアクセスを防止するルールのない空のエクスポートポリシーが使用されます。ボリュームがノードに公開されると、Tridentは、指定したCIDRブロック内のノードIPを含む基盤となるqtreeと同じ名前のエクスポートポリシーを作成します。これらのIPは、親FlexVol 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：
    - バックエンドUUID 403b5326-8482-40dB-96d0-d83fb3f4daec
    - `autoExportPolicy`に設定 `true`
    - ストレージプレフィックス `trident`
    - PVC UUID a79bcf5f-7b6d-4a40-9876-e2551f159c1c
    - svm_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1cという名前のqtree Tridentでは、という名前のFlexVolのエクスポートポリシー、という名前のqtreeのエクスポートポリシー、`trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`およびという名前の空のエクスポートポリシー `trident_empty`がSVM上に作成されます `trident-403b5326-8482-40db96d0-d83fb3f4daec`。FlexVolエクスポートポリシーのルールは、qtreeエクスポートポリシーに含まれるすべてのルールのスーパーセットになります。空のエクスポートポリシーは、関連付けられていないボリュームで再利用されます。
- `autoExportCIDRs`アドレスブロックのリストが含まれます。このフィールドは省略可能で、デフォルト値は ["0.0.0.0/0" 、 "::/0" です。定義されていない場合、Tridentは、パブリケーションを使用して、ワーカーノード上で見つかったグローバルスコープのユニキャストアドレスをすべて追加します。

この例では `192.168.0.0/24`、アドレス空間が提供されています。これは、パブリケーションでこのアドレス範囲に含まれるKubernetesノードIPが、Tridentが作成するエクスポートポリシーに追加されることを示します。Tridentは、実行するノードを登録すると、ノードのIPアドレスを取得し、で指定されたアドレスブロックと照合し `autoExportCIDRs`ます。公開時に、IPをフィルタリングした後、Tridentは公開先ノードのクライアントIPのエクスポートポリシールールを作成します。

バックエンドの作成後に 'autoExportPolicy' および 'autoExportCIDRs を更新できます自動的に管理されるバックエンドに新しい CIDRs を追加したり、既存の CIDRs を削除したりできます。CIDRs を削除する際は、既存の接続が切断されないように注意してください。バックエンドに対して「 autoExportPolicy 」を無効にし、手動で作成したエクスポートポリシーに戻すこともできます。これには、バックエンド構成で「 exportPolicy 」パラメータを設定する必要があります。

Tridentがバックエンドを作成または更新した後、または対応するCRDを `tridentbackend`使用してバックエンドをチェックでき `tridentctl`ます。

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

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

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

メモ	自動管理されたエクスポートポリシーを使用してバックエンドを削除すると、動的に作成され

> たエクスポートポリシーが削除されます。バックエンドが再作成されると、そのバックエンドは新しいバックエンドとして扱われ、新しいエクスポートポリシーが作成されます。

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

SMBボリュームをプロビジョニングする準備をします

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

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

作業を開始する前に

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

- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2022を実行しているKubernetesクラスタ。Tridentでは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみがサポートされます。
- Active Directoryクレデンシャルを含む少なくとも1つのTridentシークレット。シークレットを生成するには `smbcreds`：

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

- Windowsサービスとして設定されたCSIプロキシ。を設定します `csi-proxy`を参照してください "GitHub: CSIプロキシ" または "GitHub: Windows向けCSIプロキシ" Windowsで実行されているKubernetesノードの場合。

手順

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

メモ	Amazon FSx for ONTAPにはSMB共有が必要です。

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

   a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

b. 指定したSVMに関連付けられているSMB共有を作成します。

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

c. 共有が作成されたことを確認します。

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

メモ　を参照してください "SMB 共有を作成" 詳細については、

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

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

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。

覚えておいてください

- 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。
- セキュアSMBを有効にする際は、有効なADユーザーを指定してください。無効なユーザーはACLに追加されません。
- バックエンド、ストレージ クラス、PVC で同じ AD ユーザーに異なる権限を指定した場合、権限の優先順位は PVC、ストレージ クラス、バックエンドの順になります。
- セキュアSMBは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することで、セキュアSMBを確実に有効にすることができます。アノテーションに指定されたユーザー値は `trident.netapp.io/smbShareAdUser`で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。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の設定オプションと例

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

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

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

26.06 リリース以降、ONTAP のバランスの取れた配置を有効にすることで、ONTAP が新しいボリュームのアグリゲートを選択できるようになります。ONTAP 9.19.1 以降の AFX ストレージシステムでは、バランスの取れた配置が必要です。要件と例については、"ONTAPバックエンドでバランスの取れた配置を使用する" を参照してください。

以下の表は、バックエンドの設定オプションを示しています：

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前。 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。 メモ NetApp AFXシステムの場合のみ `ontap-nas`サポートされています。	`ontap-nas`、 `ontap-nas-economy`、または `ontap-nas-flexgroup`
backendName`	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタまたはSVM管理LIFのIPアドレス完全修飾ドメイン名（FQDN）を指定できます。IPv6フラグを使用してTridentがインストールされている場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。シームレスなMetroClusterスイッチオーバーについては、を参照してMetroClusterの例ください。	"10.0.0.1 ","[2001：1234：abcd：：fe]"
「重複排除	プロトコル LIF の IP アドレス。を指定することを推奨しますNetApp `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSのマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定すると、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散できます。初期設定後に変更できます。を参照してください 。IPv6フラグを使用してTridentがインストールされている場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、のように角かっこで定義する必要があります `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。* MetroClusterの場合は省略してください。*を参照してください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から取得されるアドレス（非推奨）
'VM'	使用する Storage Virtual Machine * MetroClusterの場合は省略してください。* MetroClusterの例。	SVM 「管理 LIF 」が指定されている場合に生成されます

パラメータ	説明	デフォルト
「autoExportPolicy」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Tridentでエクスポートポリシーを自動的に管理できます。	いいえ
「 autoExportCI`」	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy`と、Tridentでエクスポートポリシーを自動的に管理できます。	["0.0.0.0/0","::/0"]'
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「clientCertificate」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「clientPrivateKey」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""
「trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	""
「ユーザ名」	クラスタ/SVM に接続するためのユーザー名。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 」と入力します	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"。	
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません メモ: ontap-nas-economy と24文字以上のstoragePrefixを使用する場合、qtreeにはストレージプレフィックスが埋め込まれませんが、ボリューム名には含まれています。	"トライデント"

パラメータ	説明	デフォルト
「集約」	プロビジョニング用のアグリゲート（オプション。設定する場合は、SVMに割り当てる必要があります）。 `ontap-nas-flexgroup`ドライバーの場合、このオプションは無視されます。割り当てられていない場合は、利用可能なアグリゲートのいずれかを使用してFlexGroupボリュームをプロビジョニングできます。 メモ: SVMでアグリゲートが更新されると、Trident コントローラを再起動することなく、SVMをポーリングすることでTridentに自動的に反映されます。Tridentでボリュームをプロビジョニングするために特定のアグリゲートを設定している場合、そのアグリゲートの名前が変更されたり、SVMから移動されたりすると、SVMアグリゲートのポーリング中にバックエンドがTridentの障害状態に移行します。バックエンドをオンラインに戻すには、アグリゲートをSVM上に存在するものに変更するか、アグリゲートを完全に削除する必要があります。 **AFX** ストレージ システムには指定しないでください。 • `useBalancedPlacement`が `true`に設定されている場合は指定しないでください。*	""
`useBalancedPlacement`	新しいボリュームのアグリゲート選択を ONTAP に委任します [Boolean]。ONTAP 9.19.1 以降が必要で、`useREST` を `true` に設定する必要があります。`aggregate` と組み合わせることはできません。 要件と例については、"ONTAPバックエンドでバランスの取れた配置を使用する"を参照してください。 指定されている場合は、**ONTAP 9.19.1**以降の**AFX**ストレージシステムでは常に **`true`**に設定されます。	`false`; `true` ONTAP 9.19.1以降のAFXストレージシステムの場合
「AggreglimitateUsage 」と入力します	使用率がこのパーセンテージを超える場合、プロビジョニングは失敗します。* Amazon FSx for ONTAPには適用されません*。**AFX** ストレージ システムには指定しないでください。	"" （デフォルトでは適用されません）

パラメータ	説明	デフォルト
flexgroupAggregateList	プロビジョニング用のアグリゲートのリスト（オプション。設定されている場合はSVMに割り当てる必要があります）。SVMに割り当てられたすべてのアグリゲートを使用して、FlexGroupボリュームがプロビジョニングされます。ONTAP - NAS - FlexGroup * ストレージドライバーでサポートされています。 メモ SVMで集約リストが更新されると、Trident コントローラを再起動することなく、SVMをポーリングすることでリストがTridentに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ている場合、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の集約をポーリングしている間にバックエンドがTridentで失敗状態に移行します。バックエンドをオンラインに戻すには、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」と入力します	要求されたボリューム サイズがこの値を超える場合、プロビジョニングは失敗します。	""（デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} 使用しないでください `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
`nasType`	NFS 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。指定されている場合、常に **`nfs`AFX** ストレージ システム用。	`nfs`
「nfsvMountOptions」のように入力します	NFSマウントオプションをカンマ区切りで指定します。Kubernetesの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設定ファイルで指定されたマウントオプションを使用するようにフォールバックします。ストレージクラスまたは設定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。有効な範囲は [50 、300] です。	"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でSMB共有を作成できるようにする名前、ボリュームへの共通の共有アクセスを禁止する場合はパラメータを空白のままにします。オンプレミスのONTAPでは、このパラメータはオプションです。このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。	smb-share
「 useREST` 」	ONTAP REST API を使用するためのブール 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。指定されている場合、常に **`true`AFX** ストレージ システム用。	true ONTAP 9.15.1以降の場合は、それ以外の場合は false。
limitVolumePoolSize	ONTAP NASエコノミーバックエンドでqtreeを使用する場合の、要求可能なFlexVolの最大サイズ。	""（デフォルトでは適用されません）
denyNewVolumePools	を制限し `ontap-nas-economy`バックエンドがqtreeを格納するために新しいFlexVolボリュームを作成することます。新しいPVのプロビジョニングには、既存のFlexVolのみが使用されます。	
adAdminUser	SMB共有へのフルアクセス権を持つActive Directory管理者ユーザーまたはユーザーグループ。このパラメータを使用して、SMB共有へのフルコントロール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用のバックエンド構成オプション

これらのオプションを使用して、のデフォルトプロビジョニングを制御できます defaults 設定のセクション。例については、以下の設定例を参照してください。

パラメータ	説明	デフォルト
「平和の配分」	qtreeに対するスペース割り当て	"正しい"
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	"なし"
「ナプショットポリシー」	使用する Snapshot ポリシー	"なし"

パラメータ	説明	デフォルト
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	""
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	次の場合は「0」 `snapshotPolicy` は「none」、それ以外の場合は「」です。
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	新しいボリュームで NetApp Volume Encryption （NVE）を有効にします。デフォルトは `false`です。このオプションを使用するには、クラスター上で NVE のライセンスを取得し、有効化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、次を参照してください：TridentとNVEおよびNAEとの連携	いいえ
階層ポリシー	「none」を使用する階層化ポリシー	
「 unixPermissions 」	新しいボリュームのモード	NFSボリュームの場合は「777 」、SMBボリュームの場合は空（該当なし）
「スナップショット方向」	にアクセスする権限を管理します。 `.snapshot` ディレクトリ	`true`、 `false` （明示的に設定）。
「 exportPolicy 」と入力します	使用するエクスポートポリシー	デフォルト
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。
`nameTemplate`	カスタムボリューム名を作成するためのテンプレート。	""

メモ	TridentでQoSポリシーグループを使用するには、ONTAP 9 .8以降が必要です。共有されていないQoSポリシーグループを使用し、ポリシーグループが各コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。

ボリュームプロビジョニングの例

デフォルトが定義されている例を次に示します。

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

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 内で要求した書き込み可能な領域を確実に受け取り、要求した領域よりも少ない領域を受け取ることがなくなります。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

計算は次のとおりです。

```
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。 `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説明に従って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、 `snapshotReserve=50`以前の 2 GiB の PVC では、1 GiB の書き込み可能領域を提供するボリュームが作成されました。例えば、ボリュームのサイズを 3 GiB に変更すると、アプリケーションは 6 GiB のボリューム上で 3 GiB の書き込み可能な領域を利用できるようになります。

最小限の設定例

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

メモ	ネットアップ ONTAP で Trident を使用している場合は、 IP アドレスではなく LIF の DNS 名を指定することを推奨します。

▼ **ONTAP NAS**エコノミーの例

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

▼ **ONTAP NAS FlexGroup**の例

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

▼ **MetroCluster**の例

スイッチオーバーやスイッチバックの実行中にバックエンド定義を手動で更新する必要がないようにバックエンドを設定できます。 "SVMのレプリケーションとリカバリ"。

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

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

▼ **SMB**ボリュームの例

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

▼ 証明書ベースの認証の例

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

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

▼ 自動エクスポートポリシーの例

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

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

▼ バランスの取れた配置例

この例では、ONTAP のバランスの取れた配置を有効にして、ONTAP が新しいボリュームのアグリゲートを選択できるようにします。バランスの取れた配置には、ONTAP 9.19.1 以降が必要です。詳細については、"ONTAPバックエンドでバランスの取れた配置を使用する"を参照してください。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-balanced-placement
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
useBalancedPlacement: true
```

▼ **IPv6**アドレスの例

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

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
```

```
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

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

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

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

▼ **nameTemplate**を使用したバックエンド構成の例

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

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

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

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

▼ **ONTAP NAS**の例

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

```
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      app: wordpress
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

▼ **ONTAP NAS FlexGroupの例**

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
```

```
      creditpoints: "50000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: gold
      creditpoints: "30000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

▼ **ONTAP NASエコノミーの例**

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: nas_economy_store
region: us_east_1
```

```
storage:
  - labels:
      department: finance
      creditpoints: "6000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: engineering
      creditpoints: "3000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      department: humanresource
      creditpoints: "2000"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

バックエンドを **StorageClasses** にマッピングします

次のStorageClass定義は[仮想プールを使用するバックエンドの例]を参照しています。 `parameters.selector` フィールドを使用して、各StorageClassはボリュームのホストに使用できる仮想プールを指定します。ボリュームには、選択した仮想プールで定義された属性が含まれます。

- `protection-gold` StorageClassは、 `ontap-nas-flexgroup`バックエンドの1番目と2番目の仮想プールにマッピングされます。これらは、ゴールドレベルの保護を提供する唯一のプールです。

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

```
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

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

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

- `app-mysqldb` StorageClassは、 `ontap-nas`バックエンドの4番目の仮想プールにマッピングされます。これは、mysqldbタイプのアプリケーション向け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"
```

- `protection-silver-creditpoints-20k` StorageClassは、 `ontap-nas-flexgroup`バックエンドの3番目の仮想プールにマッピングされます。これは、シルバーレベルの保護と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"
```

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

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

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

更新 `dataLIF` 初期設定後

初期設定後にdataLIFを変更するには、次のコマンドを実行して新しいバックエンドJSONファイルに更新されたdataLIFを指定します。

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

メモ	PVCが1つ以上のポッドに接続されている場合、新しいデータLIF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稼働状態に戻す必要があります。

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
```

メモ	必ず追加してください `annotations`セキュア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
```

ONTAPバックエンドでバランスの取れた配置を使用する

ONTAP バックエンドで ONTAP のバランスの取れた配置を有効にすることができます。バランスの取れた配置では、ONTAP が新規ボリュームごとに最適なアグリゲートを選択するため、Trident でアグリゲートの割り当てを管理する必要はありません。

バランスの取れた配置の仕組み

デフォルトでは、Trident は SVM に割り当てられたアグリゲートを検出し、各アグリゲートをストレージプールとして扱い、ボリュームをプロビジョニングする際にアグリゲートを選択します。

バランス配置を有効にすると：

- Tridentは、個々のアグリゲートを報告するのではなく、SVM全体を単一のストレージプールとして報告します。
- ONTAPは新規ボリュームごとにアグリゲートを選択し、使用可能なアグリゲート全体にボリュームを分散させます。
- Tridentは、ONTAP `/application/containers` REST APIを使用してボリュームを作成します。

バランスの取れた配置は、統合 ONTAP システム（AFF、FAS、および ASA）と NetApp AFX ストレージシステムで使用できます。

要件

- バックエンドは ONTAP 9.19.1 以降を使用する必要があります。
- Trident は REST を使用して ONTAP と通信する必要があります。ZAPI ではバランス配置は利用できません。
- バックエンド構成では、アグリゲートを指定してはいけません。バランスのとれた配置と明示的なアグリゲート選択を組み合わせることはできません。

デフォルト値

`useBalancedPlacement`のデフォルト値はストレージシステムによって異なります。

ストレージ システム	デフォルト	Notes
統合ONTAP（AFF、FAS、ASA）	「偽」	バランス配置を有効にしない限り、Tridentはアグリゲートを選択します。
NetApp AFX（ONTAP 9.19.1以降）	「真」	バランスの取れた配置が必要です。`useBalancedPlacement`を `false`に設定すると、バックエンドの検証が失敗します。
NetApp AFX（ONTAP 9.19.1より前）	「偽」	バランスの取れた配置はできません。

設定オプション

パラメータ	説明	デフォルト
`useBalancedPlacement`	新しいボリュームの集約選択を ONTAP に委任します。このオプションは、バックエンド構成のルート、または個々の仮想プールに設定してください。	`false`; `true` ONTAP 9.19.1以降のAFX向け

仮想プールによるバランスの取れた配置

バランスの取れた配置は、仮想プールの有無に関わらず機能します。

- 仮想プールがなければ、Trident は SVM を単一のストレージプールとして報告し、バランスの取れた配置を使用してすべての新しいボリュームを作成します。
- 仮想プールでは、Trident は通常どおり仮想プールを報告し、ボリュームをそれらに割り当てます。一致する仮想プールでバランス配置が有効になっている場合、ONTAP はボリュームのアグリゲートを選択します。

> `useBalancedPlacement`をバックエンド構成のルートに設定してすべての仮想プールに適用するか、個々の仮想プールに設定して選択的に有効にすることができます。

例

次の例では、統合された ONTAP バックエンドでバランスの取れた配置を有効にします：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-balanced-placement
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: cluster-admin
password: <password>
useBalancedPlacement: true
```

次の例では、 `TridentBackendConfig`を使用してバランスの取れた配置を有効にします：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: nas-balanced-placement
```

```
  managementLIF: 10.0.0.1
  svm: trident_svm
  useBalancedPlacement: true
  credentials:
    name: backend-tbc-ontap-nas-secret
```

次の例では、 `gold`仮想プールのみでバランスの取れた配置を有効にします。Trident は、 `bronze`仮想プールに一致するボリュームのアグリゲートを引き続き選択します：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-virtual-pools
managementLIF: 10.0.0.1
svm: trident_svm
username: cluster-admin
password: <password>
storage:
- labels:
    performance: gold
  useBalancedPlacement: true
- labels:
    performance: bronze
```

バックエンド検証エラー

バックエンドの作成または更新が以下のケースで失敗します：

構成	結果
`useBalancedPlacement: true` （ONTAP 9.19.1より前）	検証に失敗しました。ONTAP 9.19.1以降にアップグレードしてください。
`useBalancedPlacement: true`ZAPIを使用するバックエンド	検証に失敗しました。RESTを使用するようにバックエンドを構成してください。
`useBalancedPlacement: true`特定のアグリゲートと組み合わせる	検証に失敗しました。集約構成を削除するか、バランス配置を無効にしてください。
`useBalancedPlacement: false` AFX と ONTAP 9.19.1 以降	検証に失敗しました。ONTAP 9.19.1以降のAFXには、バランスの取れた配置が必要です。

NetApp ONTAP 対応の Amazon FSX

Amazon FSx for NetApp ONTAPでTridentを使用

"NetApp ONTAP 対応の Amazon FSX"は、NetApp ONTAPストレージオペレーティングシステムを基盤とするファイルシステムを実行するフルマネージドAWSサービスで

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

要件

"Tridentの要件"FSx for ONTAPとTridentを統合するには、さらに次のものが必要です。

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

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

考慮事項

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

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

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には2つの認証モードがあります。

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

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

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

認証を有効にする方法の詳細については、使用しているドライバタイプの認証を参照してください。

- "ONTAP NAS認証"
- "ONTAP SAN認証"

テスト済みの**Amazon**マシンイメージ（**AMIS**）

EKSクラスタ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ではコンテナとEKS用に特定のAmazon Machine Images（AMIS）が最適化されています。次の AMI は NetApp Trident 25.02 でテストされています。

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

BOTTLEROCKET_ARM_64	はい**	はい。	N/A	N/A

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

メモ	目的のAMIがここにリストされていない場合、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ことを意味します。このリストは、AMI が動作することがわかっている場合のガイドとして機能します。

テスト実施項目：

- 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の両方をテストしました。
- SANについてはiSCSIのみをテストし、NVMe-oFはテストしませんでした。

実行されたテスト：

- 作成：ストレージクラス、PVC、POD
- 削除：ポッド、PVC（通常、qtree / LUN–エコノミー、NASとAWSバックアップ）

詳細については、こちらをご覧ください

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

IAMロールと**AWS Secret**を作成する

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

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

AWS Secrets Managerシークレットの作成

TridentはFSx SVMに対してAPIを発行してストレージを管理するため、そのためにはクレデンシャルが必要になります。これらのクレデンシャルを安全に渡すには、AWS Secrets Managerシークレットを使用します。そのため、AWS Secrets Managerシークレットをまだ作成していない場合は、vsadminアカウントのクレデンシャルを含むシークレットを作成する必要があります。

次の例では、Trident CSIクレデンシャルを格納する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ポリシーの作成

Tridentを正しく実行するには、AWSの権限も必要です。そのため、必要な権限をTridentに付与するポリシーを作成する必要があります。

次の例は、AWS CLIを使用してIAMポリシーを作成します。

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

Kubernetes サービスアカウントを設定して、EKS ポッド ID またはサービスアカウントの関連付け (IRSA) 用の IAM ロールを持つ AWS Identity and Access Management (IAM) ロールを引き受けることができます。これにより、この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、その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ポッド ID に対してこのロールを引き受けられるように、trust-relationship.json を作成します。次に、以下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** の関連付けを作成する:

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"
        }
      }
    }
  ]
}
```

ファイルの次の値を更新し `trust-relationship.json`ます。

- **<account_id>**-お客様のAWSアカウントID
- **<oidc_provider>**- EKSクラスタのOIDC。oidc_providerを取得するには、次のコマンドを実行します。

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

- IAMポリシーにIAMロールを関連付ける*：

ロールを作成したら、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関連付けます。

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

- OICDプロバイダが関連付けられていることを確認します*：

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

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

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

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

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

```
eksctl create iamserviceaccount --name trident-controller --namespace
```

```
trident \
  --cluster <my-cluster> --role-name AmazonEKS_FSxN_CSI_DriverRole
--role-only \
  --attach-policy-arn <IAM-Policy ARN> --approve
```

Trident をインストール

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

- Helm
- EKSアドオン

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

Helmを使用した**Trident**のインストール

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
```

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

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

```
100.2502.0    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
```

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

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

メモ

iSCSI を使用する場合は、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。AL2023Worker node OS を使用している場合は、helm インストール時に 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アドオンを使用して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、バグ修正が含まれており、AWSによってAmazon EKSと連携することが検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

前提条件

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

- アドオンサブスクリプションがあるAmazon EKSクラスタアカウント
- AWS MarketplaceへのAWS権限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

AWS向けTridentアドオンを有効にする

管理コンソール

1. でAmazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーションペインで、*[クラスタ]*を選択します。
3. NetApp Trident CSIアドオンを設定するクラスタの名前を選択します。
4. *アドオン*を選択し、*追加のアドオン*を選択します。
5. アドオンを選択するには、次の手順に従います。
   a. **AWS Marketplace** アドオン セクションまでスクロールし、検索ボックスに 「**Trident**」 と入力します。
   b. Trident by NetApp ボックスの右上隅にあるチェックボックスを選択します。
   c. 「 * 次へ * 」を選択します。
6. [Configure selected add-ons* settings]ページで、次の手順を実行します。

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

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

```
{

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

}
```

+
[Conflict resolution method]で[Override]を選択すると、既存のアドオンの1つ以上の設定をAmazon EKSアドオン設定で上書きできます。このオプションを有効にしない場合、既存の設定と競合すると、操作は失敗します。表示されたエラーメッセージを使用して、競合のトラブルシューティングを行うことができます。このオプションを選択する前に、Amazon EKSアドオンが自己管理に必要な設定を管理していないことを確認してください。

7. [次へ]*を選択します。
8. [確認して追加]ページで、*[作成]*を選択します。

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

AWS CLI

1.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:

メモ | ビジネス

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

IRSA 認証の場合は、次の形式を使用します:

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

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

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

次の例では、Trident EKSアドオンをインストールします。

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

Trident EKSアドオンの更新

管理コンソール

1. Amazon EKSコンソールを開き https://console.aws.amazon.com/eks/home#/clustersます。
2. 左側のナビゲーションペインで、*[クラスタ]*を選択します。
3. NetApp Trident CSIアドオンを更新するクラスタ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、 Edit *を選択します。
6. [Configure Trident by NetApp *]ページで、次の手順を実行します。
   a. 使用する*バージョン*を選択します。
   b. [Optional configuration settings]*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

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

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

出力例：

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

- 前の手順の出力でupdate availableで返されたバージョンにアドオンを更新します。

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

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

Trident EKSアドオンのアンインストール/削除

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

- クラスタにアドオンソフトウェアを保持–このオプションを選択すると、Amazon EKSによる設定の管理が削除されます。また、Amazon EKSが更新を通知し、更新を開始した後にAmazon EKSアドオンを自動的に更新する機能も削除されます。ただし、クラスタ上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンはAmazon EKSアドオンではなく自己管理型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。アドオンを保持するには、コマンドのオプションをそのまま使用し `--preserve` ます。
- クラスターからアドオンソフトウェアを完全に削除する–NetAppは、クラスター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推奨します。コマンドからオプションを削除してアドオンを削除し `--preserve delete` ます。

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

管理コンソール

1. でAmazon EKSコンソールを開きます https://console.aws.amazon.com/eks/home#/clusters。
2. 左側のナビゲーションペインで、*[クラスタ]*を選択します。
3. NetApp Trident CSIアドオンを削除するクラスタ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. [Remove netapp_trident-operator confirmation]*ダイアログで、次の手順を実行します。
   a. Amazon EKSでアドオンの設定を管理しないようにするには、*[クラスタに保持]*を選択します。クラスタにアドオンソフトウェアを残して、アドオンのすべての設定を自分で管理できるようにする場合は、この手順を実行します。
   b. 「netapp_trident -operator *」と入力します。
   c. 「 * 削除」を選択します。

AWS CLI

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

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

eksctl

次のコマンドは、Trident EKSアドオンをアンインストールします。

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

ストレージクラスを設定する

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

StorageClassオブジェクトを作成します

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

メモ	ストレージバックエンドを手動で構成する場合は、自動バックエンド構成なしでKubernetes StorageClassを作成するセクションを参照して、Tridentバックエンドとストレージクラスを個別に作成してください。

必須の**StorageClass**パラメータを指定します

StorageClassの作成時に定義する必要がある次の3つのパラメータ：

パラメータ	必須	を入力します	説明
`fsxFilesystemID`	はい。	文字列	FSx for NetApp ONTAP ファイルシステム ID

パラメータ	必須	を入力します	説明
'torageDriverName'	はい。	文字列	Trident ストレージ ドライバ（例： `ontap-nas`または `ontap-san`）
`credentialsName`	はい。	文字列	ONTAP クレデンシャルを含む FSx for ONTAP の Kubernetes Secret の名前

オプションのパラメータを指定します

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

StorageClass設定ファイルの例。

次の例は、自動バックエンド構成をトリガーするStorageClassを示しています。

YAML

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

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
```

```
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

StorageClassを作成します

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

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

KubernetesとTridentの両方で「basic-csi」ストレージクラスが表示され、Tridentがバックエンドでプールを検出していることを確認します。

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

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

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

バックエンド構成ステータスを確認する

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

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

ステータスを確認するには、次のコマンドを実行します：

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

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

FSxNファイルシステムを追加する

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

StorageClassを編集し、次の注釈を追加します：

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

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

運用上の考慮事項と制限事項

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

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

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

自動バックエンド構成の仕組みを理解する

Tridentは、StorageClass定義からバックエンド構成を取得します。StorageClassを適用すると、Tridentは必要なパラメータを検証し、バックエンドを作成し、StorageClassにステータスを付けて注釈を付けます。

Trident は VolumeSnapshotClass を一度だけ作成します。Trident は、その後の StorageClasses に対して同じ VolumeSnapshotClass を再利用します。

Tridentバックエンドを作成する

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

YAML

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

JSON

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

FSx for ONTAPドライバの詳細

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

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

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

バックエンドを作成します

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

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

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- Tridentバックエンド構成（TBC）が正常に作成されたことを確認します。

```
Kubectl get tbc -n trident
```

```
NAME                            BACKEND NAME          BACKEND UUID
PHASE    STATUS

backend-tbc-ontap-nas           tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9    Bound    Success
```

その他の構成オプションの詳細については、以下のバックエンドの高度な設定と例セクションを参照してください。

自動バックエンド構成*なし*でストレージクラスを構成する

以下は、TridentおよびFSx for ONTAPで使用するストレージクラス構成の例です。

NFS用ストレージクラス

この例を使用して、NFS を使用するボリュームの StorageClass を設定できます（属性の全リストについては、以下の Trident 属性セクションを参照してください）：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

iSCSI用ストレージクラス

この例を使用して、iSCSIを使用するボリュームのStorageClassを設定します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

NFSv3と**AWS Bottlerocket**を使用したストレージクラス

AWS BottlerocketでNFSv3ボリュームをプロビジョニングするには、必要なをストレージクラスに追加し`mountOptions`ます。

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

```
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

Trident StorageClass属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、azure-netapp-files、ontap-san-economy	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、 Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ontap-nas 、ontapさん、solidfireさん
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ontap-nas 、ontapさん、solidfireさん

属性	を入力します	値	提供	リクエスト	でサポートされます
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、 ONTAP-NAS-エコノミー 、 ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

[1]：ONTAP SelectまたはFSx for ONTAPシステムではサポートされていません

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

ストレージクラスを作成します

StorageClassを設定したら、Kubernetesで作成できます。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` をクリックしてKubernetesで作成します。

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

2. KubernetesとTridentの両方で「basic-csi」ストレージクラスが表示され、Tridentがバックエンドでプールを検出していることを確認します。

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

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

SMBボリュームのプロビジョニング

SMB ボリュームは、 `ontap-nas`ドライバを使用してプロビジョニングできます。ただし、そのためには以下の手順を完了する必要があります："SMBボリュームをプロビジョニングする準備をします"。

バックエンドの高度な設定と例

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	例
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
backendName`	カスタム名またはストレージバックエンド	ドライバ名+"_"+ dataLIF
「管理 LIF 」	クラスタまたはSVM管理LIFのIPアドレス完全修飾ドメイン名（FQDN）を指定できます。IPv6フラグを使用してTridentがインストールされている場合は、IPv6アドレスを使用するように設定できます。IPv6アドレスは、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。 `aws`フィールドでを指定する場合は `fsxFilesystemID`、を指定する必要はありませ `managementLIF`ん。TridentはAWSからSVM情報を取得するためです。 `managementLIF`そのため、SVMの下のユーザ（vsadminなど）のクレデンシャルを指定し、そのユーザにロールが割り当てられている必要があり `vsadmin` ます。	"10.0.0.1 ","[2001：1234：abcd：：fe]"

パラメータ	説明	例
「重複排除	プロトコル LIF の IP アドレス。**ONTAP NAS** ドライバー：NetApp では dataLIF を指定することを推奨しています。指定しない場合、 Trident は SVM から dataLIF を取得します。NFS マウント操作に使用する完全修飾ドメイン名（ FQDN ）を指定できます。これにより、複数の dataLIF 間で負荷を分散するラウンドロビン DNS を作成できます。初期設定後に変更できます。**ONTAP SAN** ドライバー： iSCSI には指定しないでください。Trident は ONTAP Selective LUN Map を使用して、マルチパス セッションの確立に必要な iSCI LIF を検出します。dataLIF が明示的に定義されている場合は警告が生成されます。Trident をIPv6フラグを使用してインストールした場合、IPv6アドレスを使用するように設定できます。IPv6アドレスは、［28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555］のように角括弧で定義する必要があります。	
「 autoExportPolicy 」を参照してください	エクスポートポリシーの自動作成と更新を有効にします[ブーリアン]。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy` と、Tridentでエクスポートポリシーを自動的に管理できます。	「偽」
「 autoExportCI` 」	が有効な場合にKubernetesのノードIPをフィルタリングするCIDRのリスト `autoExportPolicy`。オプションと `autoExportCIDRs`オプションを使用する `autoExportPolicy` と、Tridentでエクスポートポリシーを自動的に管理できます。	"["0.0.0.0/0"、"：：/0"]"
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	""
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	""
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	""

パラメータ	説明	例
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます。	""
「ユーザ名」	クラスタまたはSVMに接続するためのユーザ名。クレデンシャルベースの認証に使用されます。たとえば、vsadminのように指定します。	
「 password 」と入力します	クラスタまたはSVMに接続するためのパスワード。クレデンシャルベースの認証に使用されます。	
'VM'	使用する Storage Virtual Machine	SVM管理LIFが指定されている場合に生成されます。
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。作成後に変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`
「 AggreglimitateUsage 」と入力します	* Amazon FSx for NetApp ONTAP には指定しないでください。*指定されたと `vsadmin` には `fsxadmin`、アグリゲートの使用量を取得してTridentを使用して制限するために必要な権限が含まれていません。	使用しないでください。
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。また、qtreeおよびLUNに対して管理するボリュームの最大サイズを制限し、オプションを使用すると、FlexVol volumeあたりのqtreeの最大数をカスタマイズできます。 `qtreesPerFlexvol`	"" （デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol volumeあたりの最大LUN数は[50、200]の範囲で指定する必要があります。SANのみ。	“100”
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"api"：false、"method"：true} 使用しないでください `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null

パラメータ	説明	例
「 nfsvMountOptions 」のように入力します	NFSマウントオプションをカンマで区切ったリスト。Kubernetes永続ボリュームのマウントオプションは通常ストレージクラスで指定されますが、ストレージクラスにマウントオプションが指定されていない場合、Tridentはストレージバックエンドの構成ファイルに指定されているマウントオプションを使用してフォールバックします。ストレージクラスまたは構成ファイルでマウントオプションが指定されていない場合、Tridentは関連付けられた永続ボリュームにマウントオプションを設定しません。	""
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定オプションはです `nfs`、 `smb`、またはnull。*をに設定する必要があります `smb` SMBボリューム。*をnullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
qtreesPerFlexvol`	FlexVol volumeあたりの最大qtree数は[50、300]の範囲で指定する必要があります。	`"200"`
`smbShare`	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはTridentにSMB共有の作成を許可する名前。このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
「 useREST` 」	ONTAP REST API を使用するためのブーリアンパラメータ。に設定する `true`と、TridentはONTAP REST APIを使用してバックエンドと通信します。この機能にはONTAP 9.11.1以降が必要です。また、使用するONTAPログインロールには、アプリケーションへのアクセス権が必要です `ontap` 。これは、事前に定義された役割と役割によって実現され `vsadmin` `cluster-admin` ます。	「偽」

パラメータ	説明	例
`aws`	AWS FSx for ONTAPの構成ファイルでは、次の項目を指定できます。 - `fsxFilesystemID`：AWS FSxファイルシステムのIDを指定します。 - `apiRegion`：AWS APIリージョン名。 - `apikey`：AWS APIキー。 - `secretKey`：AWSシークレットキー。	 "" "" ""
`credentials`	AWS Secrets Managerに保存するFSx SVMのクレデンシャルを指定します。- `name`：シークレットのAmazonリソース名（ARN）。SVMのクレデンシャルが含まれています。- `type`：に設定します `awsarn`。詳細については、を参照してください "AWS Secrets Managerシークレットの作成" 。	

ボリュームのプロビジョニング用のバックエンド構成オプション

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

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	「真」
「平和のための準備」を参照してください	スペースリザベーションモード：「none」（シン）または「volume」（シック）	「 NONE 」
「ナプショットポリシー」	使用する Snapshot ポリシー	「 NONE 」

パラメータ	説明	デフォルト
「 QOSPolicy 」	作成したボリュームに割り当てるQoS ポリシーグループ。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。TridentでQoSポリシーグループを使用するには、ONTAP 9 .8以降が必要です。共有されていないQoSポリシーグループを使用し、ポリシーグループが各コンスティチュエントに個別に適用されるようにします。QoSポリシーグループを共有すると、すべてのワークロードの合計スループットの上限が適用されます。	""
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプールまたはバックエンドごとに、QOSPolicyまたはadaptiveQosPolicyのいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	""
「スナップショット予約」	Snapshot「0」用にリザーブされているボリュームの割合	がの `none`場合 `snapshotPolicy else、""
'plitOnClone	作成時にクローンを親からスプリットします	「偽」
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです。`false`このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。バックエンドでNAEが有効になっている場合、Tridentでプロビジョニングされたすべてのボリュームで NAEが有効になります。詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	「偽」
`luksEncryption`	LUKS暗号化を有効にします。"Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"を参照してください。SANのみ。	""
階層ポリシー	使用する階層化ポリシー `none`	

パラメータ	説明	デフォルト
「 unixPermissions 」	新しいボリュームのモード。* SMBボリュームは空にしておきます。*	""
'ecurityStyle'	新しいボリュームのセキュリティ形式。NFSのサポート `mixed` および `unix` セキュリティ形式SMBはをサポートします `mixed` および `ntfs` セキュリティ形式	NFSのデフォルトはです `unix` 。SMBのデフォルトはです `ntfs`。

PVCの設定

このセクションでは、設定されたKubernetes StorageClassを使用してPVを要求するPersistentVolumeClaim（PVC）を作成する方法について説明します。成功すると、PVをpodにマウントできます。

PVCの作成

A "PersistentVolumeClaim_"（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。PVC は、特定のサイズまたはアクセス モードのストレージを要求するように構成できます。関連するStorageClassを使用して、クラスタ管理者は、PersistentVolumeのサイズとアクセス モード以外にも、パフォーマンスやサービス レベルなどを制御できます。

TridentバックエンドとStorageClassを作成したら、PVCを作成できます。PVCを作成したら、そのボリュームをポッドにマウントできます。

マニフェストの例

以下の例は、基本的なPVC構成オプションを示しています。

RWXアクセスを備えた**PVC**

この例は、という名前のStorageClassに関連付けられたRWXアクセスを持つ基本的なPVCを示しています `basic-csi`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: 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 を作成します。

```
kubectl create -f pvc.yaml
```

2. PVCステータスを確認します。

```
kubectl get pvc
```

```
NAME         STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                          5m
```

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

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が作成されたら、そのPVをポッドにマウントできます。ここでは、PVをポッドに接続するためのコマンドと設定例を示します。

サンプルアプリケーションを導入する

手順

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

次に、PVCをポッドに接続するための基本的な設定例を示します。基本設定：

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

メモ　進捗状況は次を使用して監視できます。 `kubectl get pod --watch`。

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

```
kubectl exec -it 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
```

ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

EKSクラスタでの**Trident EKS**アドオンの設定

NetApp Tridentは、KubernetesでAmazon FSx for NetApp ONTAPストレージ管理を合理化し、開発者や管理者がアプリケーションの導入に集中できるようにします。NetApp Trident EKSアドオンには、最新のセキュリティパッチ、バグ修正が含まれており、AWSによってAmazon EKSと連携することが検証されています。EKSアドオンを使用すると、Amazon EKSクラスタの安全性と安定性を一貫して確保し、アドオンのインストール、構成、更新に必要な作業量を削減できます。

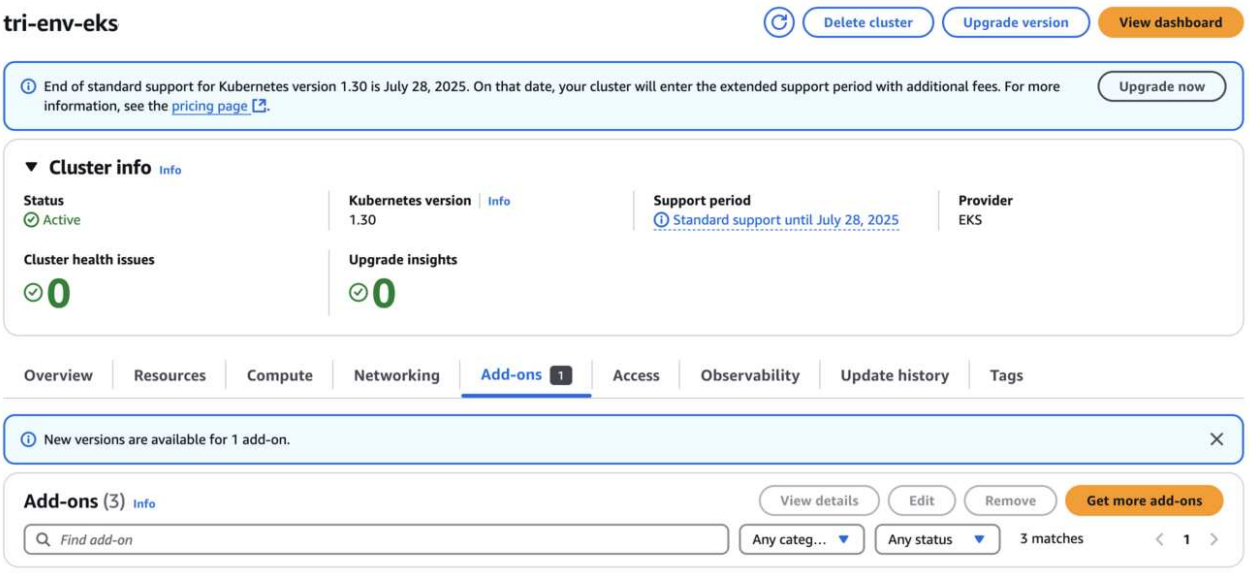
前提条件

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

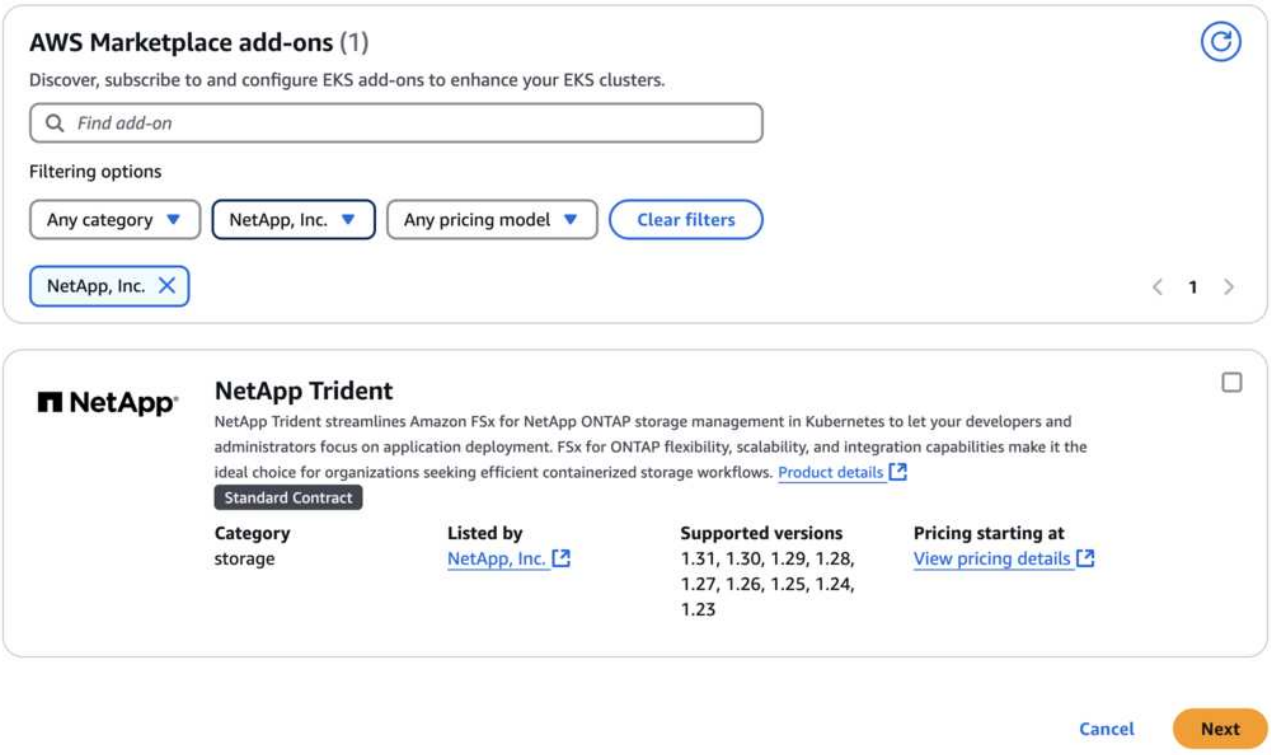
- アドオンを使用する権限を持つAmazon EKSクラスタアカウント。を参照してください "Amazon EKSアドオン"。
- AWS MarketplaceへのAWS権限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMIタイプ：Amazon Linux 2（AL2_x86_64）またはAmazon Linux 2 Arm（AL2_ARM_64）
- ノードタイプ：AMDまたはARM
- 既存のAmazon FSx for NetApp ONTAPファイルシステム

手順

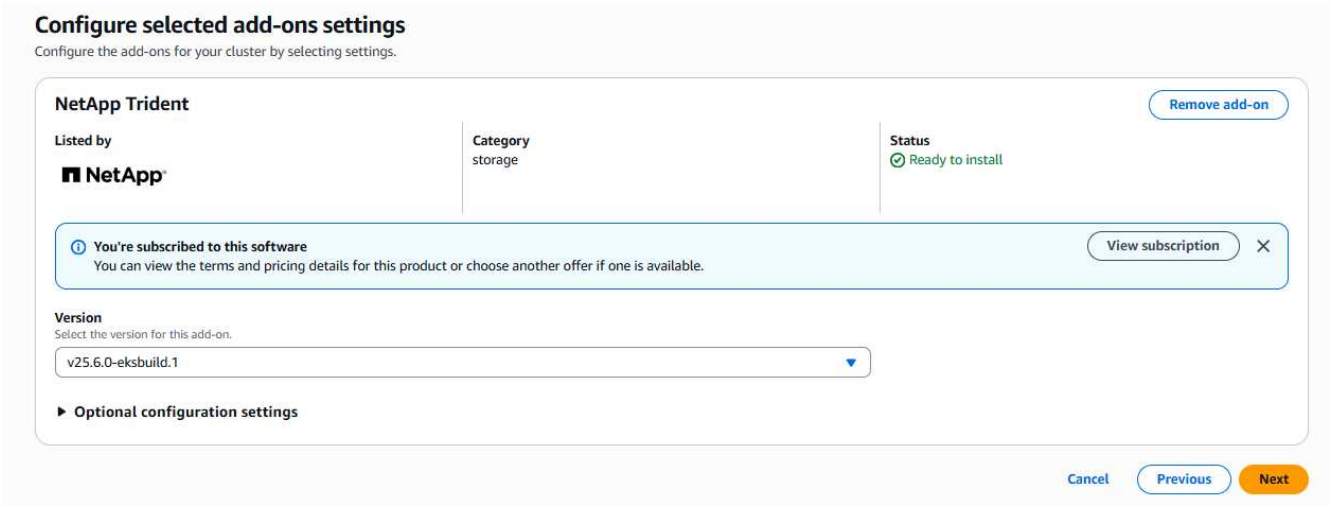
1. EKSポッドがAWSリソースにアクセスできるようにするために、IAMロールとAWSシークレットを作成してください。手順については、を参照してください"IAMロールとAWS Secretを作成する"。
2. EKS Kubernetesクラスタで、*[アドオン]*タブに移動します。



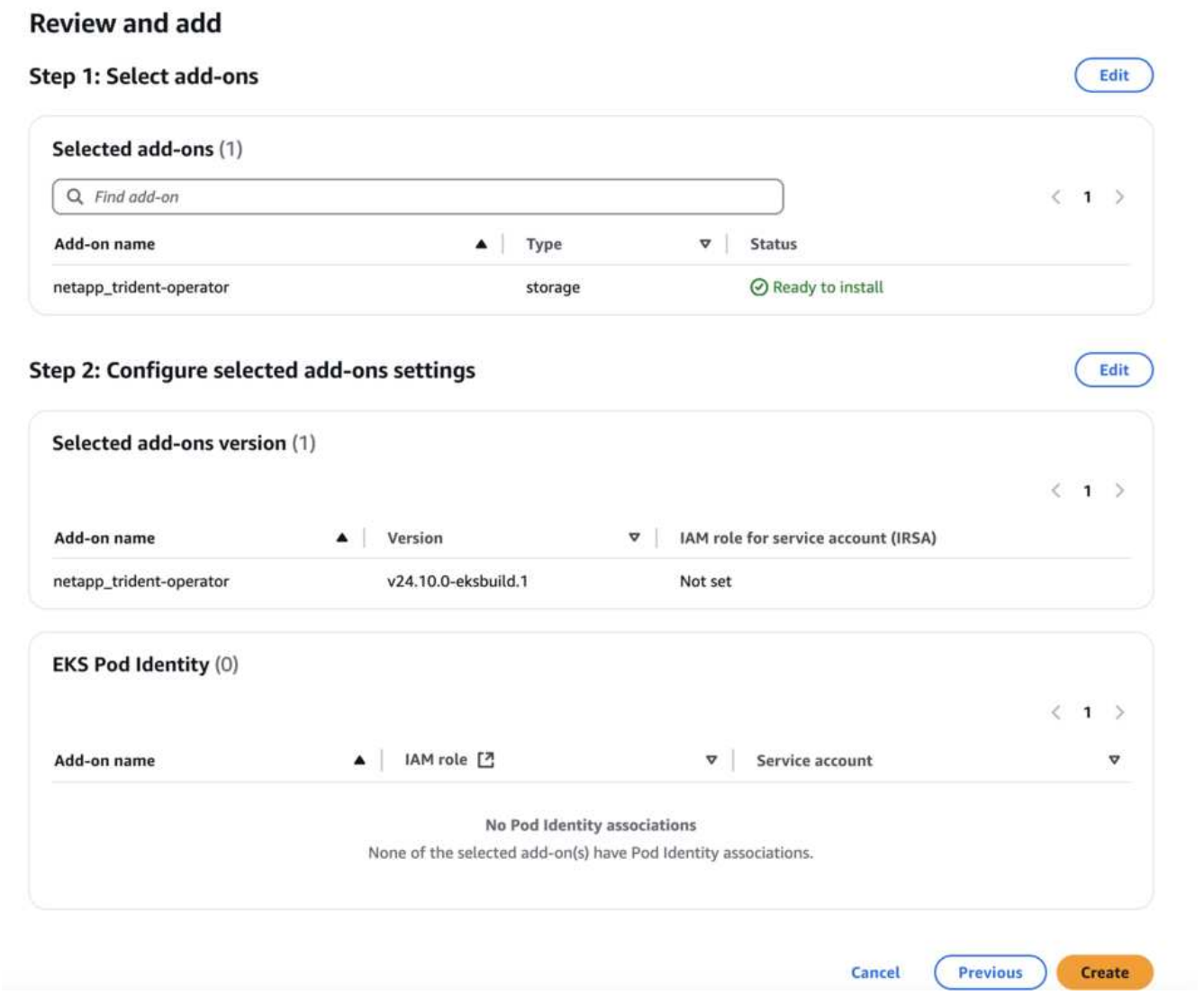
3. [AWS Marketplace add-ons]*にアクセスし、_storage_categoryを選択します。



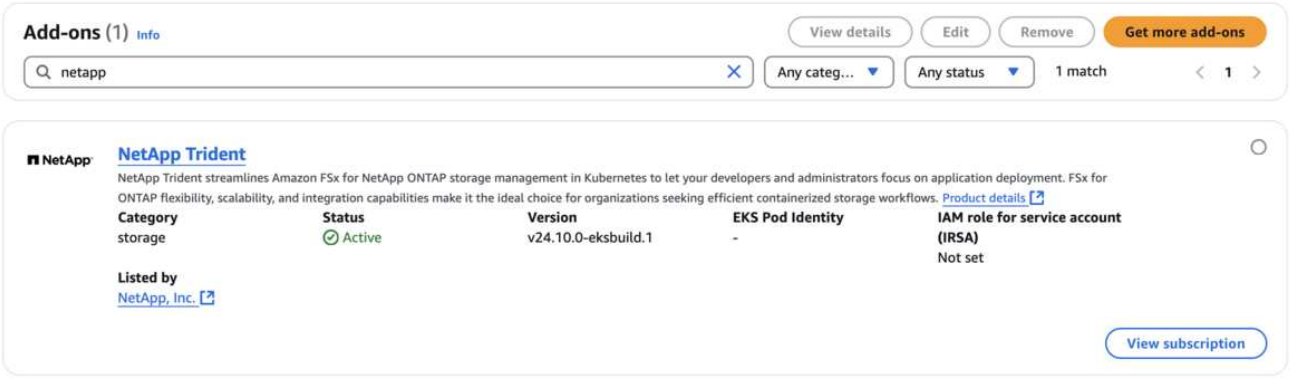
4. NetApp Trident を探し、**Trident**アドオンのチェックボックスを選択して Next *をクリックします。
5. 必要なアドオンのバージョンを選択します。



6. 必要なアドオン設定を構成します。



7. IRSA（サービスアカウントのIAMロール）を使用している場合は、追加の構成手順を参照してください。"こちらをご覧ください" 。
8. 「 * Create * 」を選択します。
9. アドオンのステータスが_Active_であることを確認します。



10. 次のコマンドを実行して、Tridentがクラスタに正しくインストールされていることを確認します。

```
kubectl get pods -n trident
```

11. セットアップを続行し、ストレージ バックエンドを構成します。詳細については、"ストレージバックエンドの設定"を参照してください。

CLIを使用したTrident EKSアドオンのインストールとアンインストール

CLIを使用して**NetApp Trident EKS**アドオンをインストールします。

次のコマンド例は、 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）

以下のコマンド例は Trident EKS アドオンバージョン 25.6.1 をインストールします：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.1-eksbuild.1`（専用バージョンを使用）

以下のコマンド例は Trident EKS アドオンバージョン 25.6.2 をインストールします：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.2-eksbuild.1`（専用バージョンを使用）

CLIを使用して**NetApp Trident EKS**アドオンをアンインストールします。

次のコマンドは、Trident EKSアドオンをアンインストールします。

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

kubectl を使用してバックエンドを作成します

バックエンドは、Tridentとストレージシステムの間の関係を定義します。Tridentは、そのストレージシステムとの通信方法や、Tridentがそのシステムからボリュームをプロビジョニングする方法を解説します。Tridentをインストールしたら、次の手順でバックエンドを作成します。 `TridentBackendConfig`Custom Resource Definition（CRD）を使用すると、Kubernetesインターフェイスから直接Tridentバックエンドを作成および管理できます。これは、またはKubernetesディストリビューション用の同等のCLIツールを使用して実行できます `kubectl`。

`TridentBackendConfig`

`TridentBackendConfig`(`tbc`, `tbconfig`, `tbackendconfig`)は、を使用してTridentバックエンドを管理できるフロントエンドの名前空間CRDです。 `kubectl`Kubernetes管理者やストレージ管理者は、Kubernetes CLIを使用して直接バックエンドを作成、管理できるようになりまし(`tridentctl`た。専用のコマンドラインユーティリティは必要ありません）。

「 TridentBackendConfig 」オブジェクトを作成すると、次のようになります。

- バックエンドは、指定した設定に基づいてTridentによって自動的に作成されます。これは内部的には（`tbe`、 `tridentbackend`）CRとして表され `TridentBackend`ます。

- は `TridentBackendConfig`、Tridentによって作成されたに一意にバインドされます `TridentBackend`。

各「 TridentBackendConfig 」は、「 TridentBackend 」を使用して 1 対 1 のマッピングを維持します。前者はバックエンドの設計と構成をユーザに提供するインターフェイスで、後者は Trident が実際のバックエンドオブジェクトを表す方法です。

警告	`TridentBackend`CRSはTridentによって自動的に作成されます。これらは * 変更しないでください。バックエンドを更新するには、オブジェクトを変更し `TridentBackendConfig`ます。

「 TridentBackendConfig 」 CR の形式については、次の例を参照してください。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

の例を確認することもできます "Trident インストーラ" 目的のストレージプラットフォーム / サービスの設定例を示すディレクトリ。

。 `spec` バックエンド固有の設定パラメータを使用します。この例では、バックエンドはを使用します `ontap-san` storage driverおよびでは、に示す構成パラメータを使用します。ご使用のストレージドライバの設定オプションのリストについては、 "ストレージドライバのバックエンド設定情報"。

「 PEC 」セクションには、「 credentials 」フィールドと「 eleetionPolicy 」フィールドも含まれています。これらのフィールドは、「 TridentBackendConfig 」 CR に新しく導入されました。

- `credentials` ：このパラメータは必須フィールドで、ストレージシステム / サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- `DeleetionPolicy`: 「 TridentBackendConfig 」が削除されたときに何が起こるかを定義します。次の 2 つの値のいずれかを指定できます。
  - 「削除」：これにより、「 TridentBackendConfig 」 CR とそれに関連付けられたバックエンドの両方が削除されます。これがデフォルト値です。
  - 「管理」：「 TridentBackendConfig 」 CR を削除しても、バックエンド定義は引き続き表示され、「 tridentctl 」で管理できます。削除ポリシーを「 retain 」に設定すると、ユーザは以前のリリース (21.04 より前) にダウングレードし、作成されたバックエンドを保持できます。このフィールドの値は、「 TridentBackendConfig 」が作成された後で更新できます。

メモ	バックエンドの名前は 'PEC.backendName' を使用して設定されます指定しない場合、バックエンドの名前は「 TridentBackendConfig 」オブジェクト（ metadata.name ）の名前に設定されます。'PEC.backendName' を使用してバックエンド名を明示的に設定することをお勧めします
ヒント	で作成されたバックエンドに `tridentctl`は、関連付けられたオブジェクトはありません `TridentBackendConfig`。このようなバックエンドをで管理するには、 `kubectl`CRを作成し `TridentBackendConfig`ます。同一の設定パラメータ（、、 `spec.storagePrefix spec.storageDriverName`など）を指定するように注意する必要があります `spec.backendName`。Tridentは、新しく作成されたを既存のバックエンドに自動的にバインドし `TridentBackendConfig`ます。

手順の概要

kubectl' を使用して新しいバックエンドを作成するには ' 次の手順を実行する必要があります

1. を作成し "Kubernetes Secret"ます。シークレットには、Tridentがストレージクラスタ/サービスと通信するために必要なクレデンシャルが含まれています。
2. 「 TridentBackendConfig 」オブジェクトを作成します。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。

バックエンドを作成したら、「 kubectl get tbc <tbc-name> -n <trident-namespac>` 」を使用してバックエンドのステータスを確認し、詳細を収集できます。

手順 **1** ： **Kubernetes Secret** を作成します

バックエンドのアクセスクレデンシャルを含むシークレットを作成します。ストレージサービス / プラットフォームごとに異なる固有の機能です。次に例を示します。

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: password
```

次の表に、各ストレージプラットフォームの Secret に含める必要があるフィールドをまとめます。

ストレージプラットフォームのシークレットフィールド概要	秘密	**Field** 概要の略
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID

ストレージプラットフォームのシークレットフィールド概要	秘密	Field 概要の略
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP
ONTAP	ユーザ名	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合

このステップで作成されたシークレットは、次のステップで作成された「 TridentBackendConfig 」オブジェクトの「 PEC.credentials 」フィールドで参照されます。

手順**2**：を作成します **`TridentBackendConfig` CR**

これで「 TridentBackendConfig 」 CR を作成する準備ができました。この例では 'ONTAP-SAN' ドライバを使用するバックエンドは ' 次に示す TridentBackendConfig オブジェクトを使用して作成されます

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
```

```
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

手順3：のステータスを確認します **`TridentBackendConfig` CR**

これで「 TridentBackendConfig 」 CR が作成され、ステータスを確認できるようになりました。次の例を参照してください。

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                       BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

バックエンドが正常に作成され、「 TridentBackendConfig 」 CR にバインドされました。

フェーズには次のいずれかの値を指定できます。

- `Bound`: `TridentBackendConfig` CRはバックエンドに関連付けられており、そのバックエンドにはが含まれています `configRef` をに設定します `TridentBackendConfig` crのuid
- Unbound ： "" を使用して表現されています「 TridentBackendConfig 」オブジェクトはバックエンドにバインドされません。新しく作成されたすべての TridentBackendConfig ' CRS は、デフォルトでこのフェーズに入ります。フェーズが変更された後、再度 Unbound に戻すことはできません。
- `Deleting`: `TridentBackendConfig` CR `deletionPolicy` が削除対象に設定されました。をクリックします `TridentBackendConfig` CRが削除され、削除状態に移行します。
  - バックエンドに永続的ボリューム要求（PVC）が存在しない場合、を削除する `TridentBackendConfig`と、`Trident`はバックエンドと`CR`を削除します `TridentBackendConfig`。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。次に 'TridentBackendConfig'CR が削除フェーズに入りますバックエンドおよび TridentBackendConfig は、すべての PVC が削除された後にのみ削除されます。
- `lost` ：「 TridentBackendConfig 」 CR に関連付けられているバックエンドが誤って削除されたか、意図的に削除されました。「 TridentBackendConfig 」 CR には削除されたバックエンドへの参照があります。「 TridentBackendConfig 」 CR は、「 $eleetionPolicy 」の値に関係なく削除できます。
- `Unknown`：TridentはCRに関連付けられたバックエンドの状態または存在を特定できません `TridentBackendConfig`。たとえば、APIサーバが応答していない場合やCRDが見つからない場合 `tridentbackends.trident.netapp.io`などです。これには介入が必要な場合があります

この段階では、バックエンドが正常に作成されます。など、いくつかの操作を追加で処理することができます

"バックエンドの更新とバックエンドの削除"。

（オプション）手順 **4** ：詳細を確認します

バックエンドに関する詳細情報を確認するには、次のコマンドを実行します。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

```
NAME                       BACKEND NAME         BACKEND UUID
PHASE    STATUS     STORAGE DRIVER    DELETION POLICY
backend-tbc-ontap-san    ontap-san-backend    8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8    Bound    Success    ontap-san         delete
```

さらに、「 TridentBackendConfig 」の YAML ／ JSON ダンプを取得することもできます。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
```

```
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

backendInfo`CRに応答して作成されたバックエンドの `TridentBackendConfig`とが `backendUUID`格納され `backendName`ます。この `lastOperationStatus`フィールドには、CRの最後の操作のステータスが表示されます。このステータス `TridentBackendConfig`は、ユーザーがトリガーした場合（ユーザーがで何かを変更した場合など）、またはTridentによってトリガーされた場合 `spec（Tridentの再起動中など）です。成功または失敗のいずれかです。 phase`CRとバックエンド間の関係のステータスを表します `TridentBackendConfig。上の例では、の phase`値がバインドされています。つまり、CRがバックエンドに関連付けられていることを意味します `TridentBackendConfig。

イベントログの詳細を取得するには、「 kubectl -n trident describe describe tbc <tbc -cr-name> 」コマンドを実行します。

警告	tridentctl を使用して ' 関連付けられた TridentBackendConfig' オブジェクトを含むバックエンドを更新または削除することはできません「 tridentctl 」と「 TridentBackendConfig 」の切り替えに関連する手順を理解するには、次の手順に従います。 "こちらを参照してください"。

バックエンドの管理

kubectlを使用した**Trident**バックエンドの管理

kubectl' を使用してバックエンド管理操作を実行する方法について説明します

バックエンドを削除します

を削除することで、 TridentBackendConfig（に基づいて）バックエンドを削除または保持するようにTridentに指示し deletionPolicy`ます。バックエンドを削除するには、がdeleteに設定されていることを確認します `deletionPolicy。のみを削除するには TridentBackendConfig、がretainに設定されていることを確認します deletionPolicy。これにより、バックエンドが引き続き存在し、を使用して管理できます tridentctl。

次のコマンドを実行します。

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

Tridentでは、で使用されていたKubernetesシークレットは削除されません TridentBackendConfig。Kubernetes ユーザは、シークレットのクリーンアップを担当します。シークレットを削除するときは注意が必要です。シークレットは、バックエンドで使用されていない場合にのみ削除してください。

既存のバックエンドを表示します

次のコマンドを実行します。

```
kubectl get tbc -n trident
```

tridentctl get backend -n trident` または tridentctl get backend -o yaml -n trident` を実行して、存在するすべて

のバックエンドのリストを取得することもできます。このリストには 'tridentctl' で作成されたバックエンドも含まれます

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新するには、オブジェクトで使用されるKubernetes Secretを `TridentBackendConfig`更新する必要があります。Tridentは、提供された最新のクレデンシャルでバックエンドを自動的に更新します。次のコマンドを実行して、 Kubernetes Secret を更新します。

```
kubectl apply -f <updated-secret-file.yaml> -n trident
```

- パラメータ（使用する ONTAP SVM の名前など）を更新する必要があります。
  - 更新できます `TridentBackendConfig` 次のコマンドを使用して、Kubernetesから直接オブジェクトを作成します。

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

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

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

メモ

- バックエンドの更新に失敗した場合、バックエンドは最後の既知の設定のまま残ります。ログを表示して原因を確認するには、「 kubectl get tbc <tbc-name> -o yaml -n trident` または「 kubectl describe tbc <tbc-name> -n trident` 」を実行します。
- 構成ファイルで問題を特定して修正したら、 update コマンドを再実行できます。

tridentctl を使用した **Trident** バックエンドの管理

tridentctl を使用してバックエンド管理操作を実行する方法について説明します

バックエンドを作成します

を作成したら "バックエンド構成ファイル"を使用して、次のコマンドを実行します。

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

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

```
tridentctl logs -n trident
```

構成ファイルの問題を特定して修正したら ' 再度 create コマンドを実行します

バックエンドを削除します

Tridentからバックエンドを削除するには、次の手順を実行します。

1. バックエンド名を取得します。

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

2. バックエンドを削除します。

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

メモ	TridentでプロビジョニングされたボリュームとこのバックエンドからSnapshotが残っている場合、バックエンドを削除すると、そのバックエンドで新しいボリュームがプロビジョニングされなくなります。バックエンドは引き続き「Deleting」状態になります。

既存のバックエンドを表示します

Trident が認識しているバックエンドを表示するには、次の手順を実行します。

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

```
tridentctl get backend -o json -n trident
```

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

バックエンドの更新が失敗した場合、バックエンドの設定に問題があるか、無効な更新を試行しました。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs -n trident
```

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

バックエンドを使用するストレージクラスを特定します

ここでは ' バックエンド・オブジェクトの tridentctl 出力と同じ JSON を使用して回答で実行できる質問の例を示しますこれには 'jq' ユーティリティが使用されますこのユーティリティをインストールする必要があります

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

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

Tridentバックエンド管理方法を選択する

Tridentでバックエンドを管理するさまざまな方法について説明します。

バックエンドを管理するためのオプション

を導入しました `TridentBackendConfig`管理者は現在、バックエンドを2つの方法で管理できるようになっています。これには、次のような質問があります。

- tridentctl を使用して作成したバックエンドは 'TridentBackendConfig' で管理できますか
- 「 TridentBackendConfig 」を使用して作成したバックエンドは、「 tridentctl 」を使用して管理できますか。

管理 **`tridentctl`** を使用してバックエンドを **`TridentBackendConfig`**

このセクションでは 'tridentBackendConfig' オブジェクトを作成して Kubernetes インターフェイスから直接 'tridentctl を使用して作成されたバックエンドの管理に必要な手順について説明します

これは、次のシナリオに該当します。

- 既存のバックエンドには `TridentBackendConfig` を使用して作成されたためです `tridentctl`。
- 「 tridentctl 」で作成された新しいバックエンドと、その他の「 TridentBackendConfig 」オブジェクトが存在します。

どちらのシナリオでも、バックエンドは引き続き存在し、Tridentはボリュームをスケジューリングして処理します。管理者には次の 2 つの選択肢があります。

- tridentctl を使用して ' バックエンドを使用して作成したバックエンドを管理します
- tridentctl を使用して作成されたバックエンドを新しい TridentBackendConfig オブジェクトにバインドしますこれは ' バックエンドが tridentctl ではなく 'kubectl' を使用して管理されることを意味します

「 kubectl 」を使用して既存のバックエンドを管理するには、既存のバックエンドにバインドする「

TridentBackendConfig 」を作成する必要があります。その仕組みの概要を以下に示します。

1. Kubernetes Secret を作成します。シークレットには、Tridentがストレージクラスタ/サービスと通信するために必要なクレデンシャルが含まれています。
2. 「 TridentBackendConfig 」オブジェクトを作成します。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。同一の構成パラメータ (`PEC.backendName`PEC.storagePrefix`PEC.storageDriverName') を指定するように注意する必要があります'PEC.backendName' は ' 既存のバックエンドの名前に設定する必要があります

手順 **0** ：バックエンドを特定します

を作成します `TridentBackendConfig` 既存のバックエンドにバインドする場合は、バックエンド設定を取得する必要があります。この例では、バックエンドが次の JSON 定義を使用して作成されているとします。

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|           NAME        | STORAGE DRIVER |                  UUID
| STATE   | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend    | ontap-nas       | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |       25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
```

```
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

手順 1 ： **Kubernetes Secret** を作成します

次の例に示すように、バックエンドのクレデンシャルを含むシークレットを作成します。

```
cat tbc-ontap-nas-backend-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password
```

```
kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

手順2：を作成します **`TridentBackendConfig` CR**

次の手順では '（この例のように）事前に存在する 'ONTAP-NAS-backend' に自動的にバインドされる 'TridentBackendConfig'CR を作成します次の要件が満たされていることを確認します。

- 「 'PEC.backendName' 」に同じバックエンド名が定義されています。
- 設定パラメータは元のバックエンドと同じです。
- 仮想プール（存在する場合）は、元のバックエンドと同じ順序である必要があります。
- クレデンシャルは、プレーンテキストではなく、 Kubernetes Secret を通じて提供されます。

この場合、「 TridentBackendConfig 」は次のようになります。

```
cat backend-tbc-ontap-nas.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
```

```
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

手順3：のステータスを確認します **`TridentBackendConfig`** CR

「 TridentBackendConfig 」が作成された後、そのフェーズは「バインド」されている必要があります。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

これで ' バックエンドは 'tbc-ontap/nas-backend`TridentBackendConfig' オブジェクトを使用して完全に管理されます

tridentBackendConfig を使用して作成されたバックエンドを一覧表示するには 'tridentctl を使用しますまた、管理者は、「 TridentBackendConfig 」を削除し、「 pec.deletionPolicy` 」が「 re 」に設定されていることを確認することで、「 tridentctl 」を使用してこのようなバックエンドを完全に管理することもできます。

手順 **0** ：バックエンドを特定します

たとえば ' 次のバックエンドが TridentBackendConfig を使用して作成されたとします

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+------------------+----------------
+--------------------------------------+--------+---------+
```

出力からはそのことがわかります `TridentBackendConfig` は正常に作成され、バックエンドにバインドされています（バックエンドのUUIDを確認してください）。

手順**1**：確認します **deletionPolicy** がに設定されます **retain**

の価値を見てみましょう `deletionPolicy`。これはに設定する必要があり `retain`ます。これにより、CRが削除されてもバックエンド定義が存在し、で管理できるように `TridentBackendConfig`なり `tridentctl`ます。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched
```

```
#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                      BACKEND NAME         BACKEND UUID
PHASE    STATUS    STORAGE DRIVER    DELETION POLICY
backend-tbc-ontap-san    ontap-san-backend    81abcb27-ea63-49bb-b606-
0a5315ac5f82    Bound    Success    ontap-san         retain
```

メモ　「削除ポリシー」が「再取得」に設定されていない限り、次の手順に進まないでください。

手順**2**：を削除します **`TridentBackendConfig` CR**

最後の手順は、「 TridentBackendConfig 」 CR を削除することです。「削除ポリシー」が「取得」に設定されていることを確認したら、削除を続行できます。

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|        NAME         | STORAGE DRIVER |                 UUID
| STATE   | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

オブジェクトが削除されると、 `TridentBackendConfig`Tridentは実際にはバックエンド自体を削除せずにオブジェクトを削除します。

ストレージクラスの作成と管理

Tridentストレージクラスの作成

Kubernetes StorageClassオブジェクトを設定してストレージクラスを作成し、Tridentでボリュームのプロビジョニング方法を指定します。

Kubernetes StorageClassオブジェクトの設定

は、 "Kubernetes StorageClassオブジェクト"そのクラスで使用されるプロビジョニングツールとしてTridentを識別し、ボリュームのプロビジョニング方法をTridentに指示します。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

ストレージクラスを作成する。

StorageClassオブジェクトを作成したら、ストレージクラスを作成できます。 [ストレージクラスノサンプル] に、使用または変更できる基本的なサンプルを示します。

手順

1. これはKubernetesオブジェクトなので、 `kubectl` をクリックしてKubernetesで作成します。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

2. KubernetesとTridentの両方で「basic-csi」ストレージクラスが表示され、Tridentがバックエンドでプールを検出していることを確認します。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
```

```
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

ストレージクラスノサンプル

Tridentが提供し "特定のバックエンド向けのシンプルなストレージクラス定義"ます。

または、 `sample-input/storage-class-csi.yaml.templ` インストーラに付属しており、 *`BACKEND_TYPE`* ストレージドライバの名前を指定します。

```
./tridentctl -n trident get backend
+-------------+----------------+--------------------------------------
+--------+---------+
|     NAME      | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+-------------+----------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+----------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
```

```
vi sample-input/storage-class-basic-csi.yaml
```

Trident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、ストレージクラスバックエンドを識別したり、ストレージクラスを削除したりできます。

既存のストレージクラスを表示します

- 既存の Kubernetes ストレージクラスを表示するには、次のコマンドを実行します。

```
kubectl get storageclass
```

- Kubernetes ストレージクラスの詳細を表示するには、次のコマンドを実行します。

```
kubectl get storageclass <storage-class> -o json
```

- Tridentの同期されたストレージクラスを表示するには、次のコマンドを実行します。

```
tridentctl get storageclass
```

- 同期されたTridentのストレージクラスの詳細を表示するには、次のコマンドを実行します。

```
tridentctl get storageclass <storage-class> -o json
```

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されています。永続ボリューム要求（ PVC ）に永続ボリュームが指定されていない場合に、永続ボリュームのプロビジョニングに使用するストレージクラスです。

- ストレージクラスの定義でアノテーションの「 torageclass.Kubernetes .io/is-default-class 」を true に設定して、デフォルトのストレージクラスを定義します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。
- 次のコマンドを使用して、既存のストレージクラスをデフォルトのストレージクラスとして設定できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 同様に、次のコマンドを使用して、デフォルトのストレージクラスアノテーションを削除できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、このアノテーションが含まれている Trident インストーラバンドルにも例があります。

メモ	クラスタ内のデフォルトのストレージクラスは一度に1つだけにしてください。Kubernetes では、技術的に複数のストレージを使用することはできますが、デフォルトのストレージクラスがまったくない場合と同様に動作します。

ストレージクラスのバックエンドを特定します

これは、Tridentバックエンドオブジェクト用に出力するJSONを使用して回答できる質問の例 `tridentctl`です。これはユーティリティを使用し `jq`ます。このユーティリティは、最初にインストールする必要がある場合があります。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

ストレージクラスを削除する

Kubernetes からストレージクラスを削除するには、次のコマンドを実行します。

```
kubectl delete storageclass <storage-class>
```

「 <storage-class> 」は、ご使用のストレージクラスに置き換えてください。

このストレージクラスを使用して作成された永続ボリュームは変更されず、Tridentで引き続き管理されます。

メモ	Tridentでは、作成するボリュームに対して空白が適用され `fsType`ます。`iSCSI`バックエンドの場合は、`StorageClass`で強制することを推奨します `parameters.fsType`。既存のストレージクラスを削除し、指定したで再作成してください `parameters.fsType`。

ボリュームのプロビジョニングと管理

Tridentボリュームをプロビジョニングする

設定したKubernetes StorageClassを使用してPVへのアクセスを要求する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

https://kubernetes.io/docs/concepts/storage/persistent-volumes["PersistentVolumeClaim_"^]（PVC）は、クラスタ上のPersistentVolumeへのアクセス要求です。

PVCは、特定のサイズまたはアクセスモードのストレージを要求するように設定できます。クラスタ管理者は、関連付けられているStorageClassを使用して、PersistentVolumeのサイズとアクセスモード（パフォーマンスやサービスレベルなど）以上を制御できます。

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の作成

手順

1. PVC を作成します。

```
kubectl create -f pvc.yaml
```

2. PVCステータスを確認します。

```
kubectl get pvc
```

```
NAME         STATUS  VOLUME     CAPACITY   ACCESS MODES  STORAGECLASS AGE
pvc-storage Bound   pv-name    1Gi        RWO                   5m
```

1. ボリュームをポッドにマウントします。

```
kubectl create -f pv-pod.yaml
```

メモ　進捗状況は次を使用して監視できます。 kubectl get pod --watch。

2. ボリュームがマウントされていることを確認します。 /my/mount/path。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. ポッドを削除できるようになりました。Podアプリケーションは存在しなくなりますが、ボリュームは残ります。

```
kubectl delete pod pv-pod
```

▼ **PersistentVolumeClaim**サンプルマニフェスト

次に、基本的なPVC設定オプションの例を示します。

RWOアクセスを備えた**PVC**

次の例は、という名前のStorageClassに関連付けられた、RWOアクセスが設定された基本的なPVCを示しています。 `basic-csi`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

NVMe / TCP対応**PVC**

この例は、という名前のStorageClassに関連付けられたNVMe/TCPの基本的なPVCとRWOアクセスを示しています。 `protection-gold`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

▼ **POD**マニフェストのサンプル

次の例は、PVCをポッドに接続するための基本的な設定を示しています。

キホンセツテイ

```
kind: Pod
apiVersion: v1
metadata:
```

```
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

NVMe/TCPの基本構成

```
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
```

ストレージクラスとパラメータおよびパラメータとの連携によるTridentによるボリュームのプロビジョニング方法の詳細については `PersistentVolumeClaim`、を参照してください"Kubernetes オブジェクトと Trident オブジェクト"。

ボリュームの移動

Trident 26.06以降、ストレージ管理者は、iSCSIボリュームを使用しているワークロードの稼働を維持したまま、同じONTAPクラスター内の別のONTAPノードおよびアグリゲートへ中断なく移動できます。移動は `TridentVolumeMove`カスタムリソースまたは `tridentctl`同等の方法で宣言的にリクエストでき、TridentがONTAPとホストを調整することでI/Oが中断されることはありません。

要件

ボリューム移動は、iSCSIボリューム専用の `ontap-san`ドライバにのみ適用されます。NVMeおよびNASプロトコルはサポートされていません。

ボリュームを移動する前に、以下の要件を満たしていることを確認してください。

- バックエンドは `ontap-san`ドライバーとONTAP REST APIを使用します。
- ONTAPクラスターには少なくとも2つのノードがあります。
- 以下のいずれかの認証情報セットを持っています。
  - 専用カスタム ONTAP ロールを使用する場合は、ONTAP 9.19.1 以降。
  - 以前の ONTAP リリースにおけるクラスター管理者クレデンシャル。

仕組み

Tridentは、同じONTAPクラスタ内の2つの場所間でボリューム（LUN）を移動します。データが移動する前に、Tridentは既存のパスに加えてターゲットへの新しいiSCSIパスを追加するため、ホストのマルチパスデバイスがアクセスを失うことはありません。ONTAPが移動を完了した後、Tridentは古いソースパスを削除します。

Tridentは、 `TridentVolumeMove`リソースで確認できる順序付けられた一連の状態を通じて移動を進めるため、移動の進捗状況を追跡できます。

ボリュームの移動

ボリュームと場所を特定する

1. 移動するPVの名前を見つけます。PV名は、移動リソースの名前でもあります。
2. ONTAP クラスターで、対象ボリュームの現在の（ソース）ノードとアグリゲートを確認し、移動先（ターゲット）ノードとアグリゲートを選択します。

移動を開始する

どちらの方法でも構いません。どちらも同じ `TridentVolumeMove`リソースを作成します。リソースの名前はPV名です。

tridentctlの使用

```
tridentctl create volume-move \
  --volume      pvc-8a814d62-bd58-4253-b0d1-82f2885db671 \
  --source-node ontap-node-01 \
  --source-pool aggr1_node01 \
  --target-node ontap-node-02 \
  --target-pool aggr1_node02 \
  -n trident
```

`tvm`は `volume-move`のエイリアスです。 `--delete-after-success 10m`を追加すると、移動が成功した

後にリソースが自動的に削除されます。

TridentVolumeMove リソースの使用

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeMove
metadata:
  name: pvc-8a814d62-bd58-4253-b0d1-82f2885db671   # the PV to move
  namespace: trident
spec:
  sourceNode: ontap-node-01
  sourcePool: aggr1_node01
  targetNode: ontap-node-02
  targetPool: aggr1_node02
  # deleteAfterSuccess: 10m   # optional
```

リソースを適用する：

```
kubectl apply -f tridentvolumemove.yaml
```

仕様フィールド

sourceNode / targetNode

ONTAPノード名（UUIDではない）。

sourcePool / targetPool

ONTAP アグリゲート名。

deleteAfterSuccess

オプションの期間（例： `30s`または `10m`）。 `0`に設定するとリソースを即座に削除し、省略するとリソースを保持します。

移動を監視する

進捗状況を追跡するには、 `tvm`略称または `tridentvolumemove`を使用します：

```
kubectl get tridentvolumemove -n trident
```

現在の状態とノードごとの進捗状況を確認するには、リソースを記述してください。

```
kubectl describe tridentvolumemove <pv-name> -n trident
```

ボリュームは、 `Moving`状態の間に同時接続を持つことができます。これは、ONTAP が大容量ボリュームの移動を完了するまでに時間がかかる場合があるためです。シームレスな移動を確保するために、Trident は

Trident コントローラーと Trident ノード間で調整を行います。各状態は、以下の手順を順番に実行し、`Succeeded`または `Failed`で終了します。

状態	何が起こっているか
`Pending`	Trident は移動要求を受け入れて検証し、ストレージの変更を開始する前に必要な情報を収集します。
`ControllerStaging`	Trident はターゲットの場所を準備します。ターゲットノードのストレージインターフェイスを LUN にマッピングし、Kubernetes を更新することで、将来のノード再起動時に正しい場所に再接続できるようにします。
`NodeStaging`	ボリュームがマウントされている各ホストで、Trident はターゲットノードからの新しいストレージパスを既存のパスと並行して追加することで、Pod の I/O が両方のパスセットにわたって継続されるようにします。
`Moving`	ONTAP はボリュームデータをコピーして、ターゲットのアグリゲートにカットオーバーします。これは最も長いフェーズであり、ONTAP が完全に処理する一方で、Trident は完了するまでポーリングを続けます。この状態では、新規の公開が許可されます。
`ControllerUnstaging`	データ移行が完了しました。Trident は Kubernetes を更新して、ソースパスを目的の状態から削除し、その後、ソースノードのストレージインターフェイスを LUN からマッピング解除します。
`NodeUnstaging`	各ホスト上で、Tridentは古いソースパスを削除します。ホストにはターゲットパスのみが残され、マルチパスデバイスは中断することなくI/O処理を継続します。
`Succeeded`	ボリュームは完全に移動され、ターゲットでオンライン状態になりました。Trident は `TridentVolumeMove`リソースを保持するか、`deleteAfterSuccess`を設定すると自動的に削除します。
`Failed`	回復不能なエラーにより、移動処理が停止したか、開始できませんでした。Trident はボリュームの状態をリセットして通常の操作を再開できるようにし、エラーメッセージとともにリソースを保持します。メッセージを確認し、根本原因を修正し、リソースを削除してから、もう一度試してください。

`.status.attachments`リストは、接続されている各ノードの進捗状況を`state`フィールドに報告します：

添付ファイルの状態	説明
`Pending`	ノードは接続済みとして識別されました。作業はまだ開始されていません。
`Bridged`	既存のパスに加えて、新しいターゲットパスが追加されました。
`Migrated`	トラフィックは新しいパスを経由して処理されるようになりました。
`Cleaned`	ノードから古いソースパスが削除されました。

添付ファイルの状態	説明
`Detached`	ノードは移動中に切り離され、Trident がアタッチメントが次のステップに進む準備ができているかどうかを評価する間、それ以上の検討から除外されました。
`Failed`	このノードでターミナルエラーが発生しました。メッセージをご覧ください。

移動リソースをクリーンアップします

`deleteAfterSuccess`を設定すると、`Trident`はリソースを自動的に削除します。それ以外の場合は、手動で削除できます。同じボリュームで別の移動をリクエストする前に、そのボリュームに `TridentVolumeMove`リソースが `Succeeded`または `Failed`の状態で存在しないことを確認してください。

メモ Trident は、失敗した移動を自動的に削除しません。Trident は `Failed`リソースを監査レコードとして保持します。根本原因を修正した後、再試行する前にリソースを手動で削除してください。

Tridentボリュームの拡張

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 ボリュームを展開します

CSI プロビジョニングを使用して、 iSCSI Persistent Volume （ PV ）を拡張できます。

メモ iSCSI ボリュームの拡張は 'ONTAP-SAN"ONTAP-SAN-エコノミー "olidfire-SAN' ドライバによってサポートされており 'Kubernetes 1.16 以降が必要です

手順 1 ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
```

```
allowVolumeExpansion: True
```

既存の StorageClass の場合は 'allowVolumeExpansion パラメータを含めるように編集します

手順 2 ：作成した **StorageClass** を使用して **PVC** を作成します

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続的ボリューム（PV）を作成し、この永続的ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

手順 3 ： **PVC** を接続するポッドを定義します

サイズを変更するポッドにPVを接続します。iSCSI PV のサイズ変更には、次の 2 つのシナリオがあります。

- PVがポッドに接続されている場合、Tridentはストレージバックエンド上のボリュームを拡張し、デバイスを再スキャンして、ファイルシステムのサイズを変更します。

- 接続されていないPVのサイズを変更しようとすると、Tridentはストレージバックエンド上のボリュームを拡張します。PVC がポッドにバインドされると、 Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、 Kubernetes は PVC サイズを更新します。

この例では、ポッドが作成され、「 1-pvc 」が使用されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ **4** ： **PV** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.storage 」を 2Gi に更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
```

```
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

手順 **5** ：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正常に機能したことを検証できます。

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+---------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE   | MANAGED |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+---------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+---------+---------+
```

FC ボリュームを拡張します

CSIプロビジョニングツールを使用して、FC永続ボリューム（PV）を拡張できます。

メモ	FCボリュームの拡張はドライバでサポートされ `ontap-san`ており、Kubernetes 1.16以降が必要です。

手順 1 ：ボリュームの拡張をサポートするようにストレージクラスを設定する

StorageClass定義を編集して `allowVolumeExpansion` フィールドからに移動します `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

既存の StorageClass の場合は 'allowVolumeExpansion パラメータを含めるように編集します

手順 2 ：作成した StorageClass を使用して PVC を作成します

PVC定義を編集し、 `spec.resources.requests.storage` 新たに必要となったサイズを反映するには、元のサイズよりも大きくする必要があります。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Tridentは永続的ボリューム（PV）を作成し、この永続的ボリューム要求（PVC）に関連付けます。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               10s
```

手順 **3** ： **PVC** を接続するポッドを定義します

サイズを変更するポッドにPVを接続します。FC PVのサイズを変更する場合は、次の2つのシナリオが考えられます。

- PVがポッドに接続されている場合、Tridentはストレージバックエンド上のボリュームを拡張し、デバイスを再スキャンして、ファイルシステムのサイズを変更します。
- 接続されていないPVのサイズを変更しようとすると、Tridentはストレージバックエンド上のボリュームを拡張します。PVC がポッドにバインドされると、 Trident はデバイスを再スキャンし、ファイルシステムのサイズを変更します。展開操作が正常に完了すると、 Kubernetes は PVC サイズを更新します。

この例では、ポッドが作成され、「 1-pvc 」が使用されます。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

```
Mounted By:    ubuntu-pod
```

ステップ **4** ： **PV** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.storage 」を 2Gi に更新します。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

手順 **5** ：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、拡張が正常に機能したことを検証できます。

```
kubectl get pvc san-pvc
```

```
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

NFS ボリュームを拡張します

Trident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` 、 そして `azure-netapp-files`バックエンド。

手順 **1** ：ボリュームの拡張をサポートするようにストレージクラスを設定する

NFS PV のサイズを変更するには ' まず 'allowVolumeExpansion フィールドを true に設定してボリュームを拡張できるようにストレージ・クラスを構成する必要があります

```
cat storageclass-ontapnas.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

このオプションを指定せずにすでにストレージ・クラスを作成している場合は 'kubectl Edit storageclass を使用して既存のストレージ・クラスを編集するだけで ' ボリュームの拡張が可能になります

手順 2 ：作成した **StorageClass** を使用して **PVC** を作成します

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident はこの PVC 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** を展開します

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
# ...
```

手順 **4** ：拡張を検証する

PVC、PV、およびTridentボリュームのサイズを確認することで、サイズ変更が正しく機能したかどうかを検証できます。

```
kubectl get pvc ontapnas20mb
NAME           STATUS    VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
```

```
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                    NAME                    |  SIZE   | STORAGE CLASS |
PROTOCOL |              BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

RWX NVMe サブシステムの制限について

NVMe プロトコルを使用する ReadWriteMany（RWX）ボリュームには、ボリュームあたり 64 ノードのスケーラビリティ制限があります。以下では、制限事項、関連する NVMe サブシステム アーキテクチャの説明、および必要な解決手順の概要を示します。

64ノードの制限について理解する

NVMe プロトコルで ReadWriteMany（RWX）ボリュームを使用する場合、単一の RWX NVMe ボリュームを Kubernetes クラスター内の 64 を超えるノードでマウントすることはできません。

同じRWX NVMe PersistentVolumeClaimをマウントするワークロードを64を超えるノードにわたってスケジュールしないでください。

この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。

NVMe サブシステム モデルを理解する

ボリュームごとのサブシステムモデル（Trident 26.02より前のリリース）

Trident 26.02 より前のリリースでは、RWX NVMe ボリュームはボリュームごとのサブシステム モデルを使用してプロビジョニングされます。各 RWX NVMe ボリュームは、ONTAP 上の専用の NVMe サブシステムにマッピングされます。

このモデルはシンプルですが、スケーラビリティの制限は低くなります。大規模な Kubernetes クラスタでは、各 RWX ボリュームが専用のサブシステムを消費するため、サブシステム コントローラの制限にすぐに達します。

スーパーサブシステムモデル（Trident 26.02で導入）

Trident 26.02 以降、RWX NVMe ボリュームは共有スーパーサブシステム モデルを使用します。複数の RWX NVMe ボリュームが同じ NVMe サブシステムを共有します。

各スーパーサブシステムは最大 1024 個の名前空間（ボリューム）をサポートします。このモデルは RWX ワークロードのスケーラビリティを大幅に向上させ、ONTAP サブシステムの制限に達する可能性を低減します。

各 RWX NVMe ボリュームは最大 64 個のノードをサポートします。

エラーの症状を特定する

RWX NVMe ボリュームを大規模に作成または接続すると、次のようなエラーが発生することがあります：

```
Maximum number of controllers reached. No more controllers can be created.
```

このエラーは、ONTAP NVMe サブシステム コントローラの制限に達したことを示しています。

サブシステム制限エラーを解決する

ボリュームごとのサブシステムの制限を超えてスーパーサブシステムモデルを活用するには、Trident 26.02以降にアップグレードしてください。

Tridentをアップグレードしてスーパーサブシステムモデルを適用する

RWX NVMe ボリュームにスーパーサブシステム モデルを適用するには：

1. Trident をバージョン 26.02 以降にアップグレードしてください。
2. RWX NVMe ボリュームを使用するすべてのポッドをゼロレプリカにスケールダウンします。
3. RWX NVMe ボリュームをアクティブに使用しているワークロードがないことを確認します。
4. ポッドをスケールアップし直します。

この再起動シーケンスにより、RWX NVMe ボリュームがスーパーサブシステム モデルを使用して接続されます。

- この制限は、NVMe プロトコルを使用する RWX ボリュームにのみ適用されます。
- RWX NVMe ボリュームごとに 64 ノードの制限が適用されます。
- 他のアクセス モードおよび他のプロトコルは影響を受けません。

コントローラのスケーラビリティ

Tridentは、複数のストレージドライバ間での同時実行性を向上させることで、コントローラのスケーラビリティを改善します。コントローラのスケーラビリティを有効にすると、Tridentコントローラはストレージ操作を直列処理するのではなく並列処理するため、同時実行操作が多いKubernetes環境でのスループットが向上します。

デプロイする前に、Trident 26.10において、どの Trident ドライバーが一般提供時にコントローラのスケーラビリティをサポートし、どのドライバーがテクニカルプレビューとして利用可能かを確認してください。これにより、十分な情報に基づいたデプロイの意思決定を行い、リスクを管理することができます。コントローラのスケーラビリティは、デフォルトでは無効になっています。

主要な概念と定義

コントローラのスケーラビリティ

コントローラーのスケーラビリティとは、Trident コントローラが、複数のストレージ操作を単一のロックの背後で直列化するのではなく、並列に処理できる能力のことです。これらの操作には、ボリュームの作成、削

除、サイズ変更、スナップショットの作成と削除、ボリュームの公開と非公開、およびバックエンド管理が含まれます。

コントローラーのスケーラビリティを有効にすると、異なるボリュームやバックエンドでの操作が同時に実行されます。これにより、PersistentVolumeClaimおよびVolumeSnapshotの同時操作数が多い環境でのスループットが向上し、エンドツーエンドの操作時間が短縮されます。

デフォルトの動作（シリアルモード）

デフォルトでは、Tridentコントローラーは一度に1つの操作を処理します。作成、削除、サイズ変更、またはスナップショットの各リクエストは、Tridentが次の処理を開始する前に完了します。シリアルモードはすべてのインストールでサポートされているデフォルトであり、設定は不要です。

ほとんどのワークロードでは、シリアルモードで十分です。コントローラーのスケーラビリティを有効にするのは、処理量が増加した結果、シリアル処理でバックログが発生した場合のみにしてください。

コントローラのスケーラビリティサポート

Tridentは、さまざまなストレージドライバに対して、異なる成熟度レベルでコントローラの拡張性をサポートします。

一般提供

以下のドライバは、Trident 26.10 の一般提供開始時点でコントローラのスケーラビリティをサポートしています：

- 「 ontap - san 」
- 「 ONTAP - NAS 」
- 「 ONTAP - NAS - エコノミー」
- 「 ONTAP-NAS-flexgroup 」
- 「 ONTAP - SAN - エコノミー」
- `google-cloud-netapp-volumes`
- 「 azure-NetApp-files 」と入力します
- 「 olidfire -san 」

メモ

`google-cloud-netapp-volumes`と `google-cloud-netapp-volumes-san`のドライバーは異なります。Tridentは `google-cloud-netapp-volumes`のみをサポートします。バックエンドの設定や例では `google-cloud-netapp-volumes-san`を使用しないでください。

テクニカルプレビュー

以下のドライバは、Trident 26.10においてテクニカルプレビュー機能としてコントローラのスケーラビリティをサポートしています：

- `asa-r2`（SANおよびNVMe）

このドライバーには以下の制限があります：

- コントローラの並行処理機能は、評価およびテスト目的でのみ利用可能です。
- 動作はリリースごとに変化する可能性があります。
- NetApp は実稼働環境での使用を推奨しません。

enableConcurrency の仕組み

`enableConcurrency`を `true`に設定すると、Tridentはコントローラが管理するすべてのバックエンドに並行処理を適用します。この設定は、すべてのバックエンドに同時に適用されます。個々のバックエンドや個々のドライバーに対して有効にすることはできません。

設定済みのバックエンドはすべて、一般提供版またはテクニカルプレビュー版のリストに掲載されているドライバを使用する必要があります。バックエンドがサポートされていないドライバを使用している場合、Trident は起動せず、Trident はサポートされていないドライバを使用するバックエンドを追加しません。

起動を復元するには、サポートされていないドライバを使用しているバックエンドを削除または再構成するか、コントローラの拡張性を無効にしてください。

有効にする前に

重要	コントローラーのスケーラビリティを有効にする前に、設定されているすべてのバックエンドが、一般提供版またはテクニカルプレビュー版のリストに含まれるドライバーを使用していることを確認してください。バックエンドがサポートされていないドライバーを使用している場合、この機能を有効にした後、Trident は起動しません。

コントローラーのスケーラビリティを有効にするかどうかは、以下の表を参考にしてください。

お使いの環境が	操作
サポートされているドライバーのみを使用し、コントローラー操作のバックログが発生します	コントローラーのスケーラビリティを有効にする。
サポートされていないドライバーを使用する	デフォルトのシリアルモードを維持します。コントローラーのスケーラビリティを有効にしないでください。
処理量が少なく、バックログなしで対応できます。	デフォルトのシリアルモードを維持します。

コントローラのスケーラビリティを有効にする

この `enableConcurrency`設定オプションは、コントローラーのスケーラビリティを制御します。このオプションは、Trident のインストール時、または既存のデプロイメントを更新する際に、明示的に有効にする必要があります。

Trident オペレータの導入

Trident operatorでコントローラの拡張性を有効にするには、 `TridentOrchestrator`カスタムリソース（CR）で `enableConcurrency`を `true`に設定します。

新規インストール

`TridentOrchestrator` CR を作成または編集し、 `enableConcurrency` を `true` に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

CRを適用します。

```
kubectl apply -f tridentorchestrator_cr.yaml
```

既存のインストール

既存の `TridentOrchestrator` CR にパッチを適用してコントローラーのスケーラビリティを有効にします：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

Trident が設定を適用したことを確認します：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm デプロイメント

Helmでコントローラーのスケーラビリティを有効にするには、 `enableConcurrency`の値を `true`に設定します。

新規インストール

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

既存のインストール

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

または、カスタム `values.yaml`ファイルで `enableConcurrency`を `true`に設定します：

```
# values.yaml
enableConcurrency: true
```

次に、値ファイルを使用してインストールまたはアップグレードします。

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl デプロイメント

`tridentctl`でコントローラのスケーラビリティを有効にするには、インストール時に `--enable-concurrency`フラグを渡します。

新規インストール

```
tridentctl install -n trident --enable-concurrency
```

既存のインストール

既存の `tridentctl`デプロイメントでコントローラーのスケーラビリティを有効にするには、Trident をアンインストールし、次のフラグを付けて再インストールしてください：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

コントローラーのスケーラビリティが有効になっていることを確認します。

コントローラのスケーラビリティを有効にした後、Trident コントローラが同時実行を有効にして動作していることを確認してください。コントローラポッドのログを確認してください：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

出力には、並行処理が有効になっていることを確認するログエントリが含まれています。

同時実行動作

コントローラーのスケーラビリティが有効になっている場合、Tridentコントローラーは以下の動作を適用します：

- Tridentは、単一のグローバルロックをリソースごとのきめ細かいロックに置き換えます。
- Tridentは、同じリソースを変更する操作を直列化することで、データの一貫性を維持します。
- リソースから読み取りのみを行う操作は、そのリソースに対する他の読み取り操作と並行して実行されます。
- Tridentは、バックエンドストレージシステムの過負荷を防ぐため、管理LIFあたりの同時ONTAP APIリクエスト数を20に制限します。
- 複数のバックエンドが同じ管理LIFを共有している場合、この20リクエストの制限も共有されます。

注意点と制限事項

Trident 26.10 のコントローラのスケーラビリティには、以下の考慮事項が適用されます：

- コントローラーのスケーラビリティは、一般提供版およびテクニカルプレビュー版のリストに掲載されているドライバーのみをサポートします。詳細については、有効にする前にを参照してください。
- Tridentコントローラーは内部的に並行処理を管理します。このリリースでは、ユーザーが設定可能な同時実行制限は提供されていません。
- 全体の処理能力は、使用するストレージドライバ、バックエンドの応答性、およびKubernetes APIサーバーのパフォーマンスに依存します。
- 同時接続数が多いと、バックエンドのストレージシステムへの負荷が増加する可能性があります。
- コントローラの拡張性に関する動作は、すべてのドライバで同一ではありません。
- テクニカルプレビュー版のドライバーは、高負荷時にパフォーマンスが不安定になる場合があり、リリースごとに動作が変わる可能性があります。
- 並列実行のため、同時実行処理のデバッグはより複雑になる場合があります。メトリクスとログには、インターリーブされた操作出力が表示される場合があります。

推奨

コントローラーのスケーラビリティを有効にする前に、以下の手順を完了してください。

1. 設定されているすべてのバックエンドが、一般提供版またはテクニカルプレビュー版のリストに掲載されているドライバを使用していることを確認してください。
2. 本番環境に適用する前に、非本番環境のクラスターで変更内容をテストしてください。
3. 変更を適用した後、コントローラーのスケーラビリティが有効になっていることを確認してください。

コントローラーのスケーラビリティを運用する際には、以下の一般的な推奨事項を適用してください。

- 高い拡張性が求められる本番環境では、汎用ドライバを使用してください。
- テクニカルプレビュー版ドライバーは、本番環境以外でのみ評価してください。
- 大規模運用を行う際は、バックエンドとコントローラーのパフォーマンスを監視してください。
- 自動化スクリプトでは、操作の順序を前提としないでください。

自動ボリューム拡張

自動ボリューム拡張により、Tridentによってプロビジョニングされた永続ボリュームは、使用容量が定義されたしきい値に達すると自動的に拡張されます。この機能により、運用上のオーバーヘッドが削減され、容量不足によるアプリケーションの中断を防ぐことができます。ボリュームの自動拡張は、自動拡張ポリシーを使用して実装されます。自動拡張ポリシーは以下を定義します：

- 拡張のトリガーとなる使用率のしきい値
- ボリュームが増加する量
- ボリュームが到達できる最大サイズ

TridentのAutogrowポリシーはPVCを拡張します。これは、PVCのサイズを変更せずに基盤となるFlexVolを拡張する"ONTAPボリュームの自動サイズ調整"とは異なります。

メモ	定義された使用率しきい値を超えると、ボリュームのサイズは自動的に増加します。ボリュームが自動的に縮小されることはありません。

要件

ボリュームの自動拡張を構成する前に、次の要件が満たされていることを確認してください。

- Trident 26.02以降
- カスタムリソースを作成するためのロールベースのアクセス制御権限 `TridentAutogrowPolicy`
- StorageClassesで構成 `allowVolumeExpansion: true`
- サポートされているONTAPプロトコル：
  - Network File System（NFS）
  - Internet Small Computer Systems Interface（iSCSI）
  - 不揮発性メモリエクスプレス（NVMe）
  - Fiber Channel Protocol（FCP）

制限

- ONTAP 9.16.1 より前の ONTAP Non-Volatile Memory Express raw ブロックボリュームは、自動拡張をサポートしていません。
- ストレージエリアネットワークボリュームの場合、 `growthAmount`が50メビバイト以下であれば、Tridentは、結果のサイズが `maxSize`を超えない限り、サイズ変更前に値を自動的に51メビバイトに増やします。
- ブラウンフィールド環境では、ボリューム公開の移行動作により、特定の既存ボリュームに対して自動拡張が機能しない場合があります。
- ボリュームが `maxSize`に達すると、それ以上の拡張は発生しません。
- 自動ボリューム拡張でサポートされているプロトコル：
  - Network File System（NFS）

- Internet Small Computer Systems Interface（iSCSI）
- 不揮発性メモリエクスプレス（NVMe）
- Fiber Channel Protocol（FCP）

ボリュームの自動拡張ポリシーによるプロビジョニング

自動拡張ポリシーは、次の2つのレベルで構成できます：

- ストレージクラスレベル：すべてのボリュームのデフォルトを設定します（アノテーションを使用）
- PVC レベル：ストレージクラスのデフォルトをオーバーライドします（アノテーションを使用）

自動拡張ポリシーを作成する

自動拡張ポリシーを使用すると、ボリュームが定義された容量しきい値に達したときにボリュームを自動的に拡張できます。

以下のものを用意してください：

- Trident 26.02以降がインストールされている
- `TridentAutogrowPolicy`リソースを作成するためのロールベースのアクセス制御権限
- ワークロードの増加要件の理解

ボリュームの自動拡張ポリシーは、定義された容量しきい値に達したときにボリュームが自動的に拡張される方法を定義します。

自動拡張ポリシーはワークフローのどの時点でも作成できます：

- StorageClassesとボリュームが作成される前
- StorageClassesが存在した後
- ボリュームのプロビジョニング後

この柔軟性により、既存のリソースを再作成せずに自動拡張を導入できます。

自動拡張ポリシーの仕様

自動拡張ポリシーは、次のように定義される Kubernetes カスタムリソースです：

フィールド	説明	の形式で入力し	必須	例	デフォルト
名前	一意のポリシー識別子	文字列	はい。	production-db-ポリシー	なし
usedThreshold	拡張をトリガーする容量の割合	パーセンテージ文字列	はい。	「80%」	なし

フィールド	説明	の形式で入力し	必須	例	デフォルト
growthAmount	しきい値に達したときに増加する量	パーセンテージまたはサイズ	いいえ	「10%」または「5Gi」	「10%」
maxSize	最大ボリュームサイズの制限	Kubernetesの数量	いいえ	「500Gi」	無制限

自動拡張ポリシーを作成する

手順

1. 自動拡張ポリシーを定義する YAML ファイルを作成します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. クラスターにポリシーを適用します：

```
kubectl apply -f autogrow-policy.yaml
```

3. ポリシーが作成されたことを確認します：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

期待される出力

```
NAME                USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow   80%              10%             Success
```

ポリシーの状態

ポリシーを作成したら、Tridentが仕様を検証し、次のいずれかの状態を割り当てます：

状態	説明	必要なアクション
成功	ポリシーは検証され、使用できる状態です。	なし。

状態	説明	必要なアクション
失敗しました	検証エラーが検出されました。	仕様を確認して修正してください。
削除中	削除中です。	完了するまでお待ちください。

ポリシーを**StorageClass**に関連付ける

`trident.netapp.io/autogrowPolicy`アノテーションを使用して、自動拡張ポリシーをStorageClassに関連付けることができます。そのStorageClassからプロビジョニングされたすべてのボリュームは、ポリシーを継承します。

メモ StorageClassには `allowVolumeExpansion: true`が必要です。

手順

1. Autogrow Policyアノテーションを使用してStorageClassを作成または変更します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. StorageClassを適用します。

```
kubectl apply -f storageclass.yaml
```

3. アノテーションを確認します：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

期待される出力

```
production-db-policy
```

ポリシーの優先順位

自動拡張ポリシー注釈がStorageClassとPVCの両方に設定されている場合、Tridentは次の優先順位ルールを適用します：

1. *PVC注釈が優先されます。*PVCが `trident.netapp.io/autogrowPolicy`を設定した場合、その値が常に使用されます。
2. **StorageClass** アノテーションは、**PVC** にアノテーションがない場合にのみ適用されます。
3. *どちらにもアノテーションがない場合、*ボリュームの自動拡張ポリシーは適用されません。

StorageClassアノテーション	**PVC**アノテーション	効果的な行動
`trident.netapp.io/autogrowPolicy: standard-agp`	未設定	用途 `standard-agp`
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	使用 `logs-policy`（PVCはStorageClassをオーバーライドします）。
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	自動拡張ポリシーなし（PVC は自動拡張を無効にします）。
未設定	`trident.netapp.io/autogrowPolicy: dev-policy`	用途 `dev-policy`
未設定	未設定	自動拡張ポリシーなし。

設定例

次の例は、さまざまなユースケースにおける一般的な Autogrow Policy 構成を示しています。

本番データベースに対する保守的なポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

固定増分によるログストレージ

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
```

```
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

デフォルト付きの最小限のポリシー

`growthAmount`と `maxSize`を省略すると、Tridentはデフォルトの(`10%`（成長、無制限のサイズ）を使用します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

カスタム**maxSize**およびデフォルト**growthAmount**を使用したポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

無制限の**maxSize**による積極的な成長

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

小数パーセンテージを含むポリシー

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

> メモ
>
> 小数点以下のパーセンテージがサポートされています。小数点以下3桁以上を指定すると、Tridentは値を小数点以下3桁に丸めます。

自動拡張機能を備えた **NAS StorageClass**

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass（自動拡張付き）

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

ONTAPボリュームの自動拡張を設定

Tridentを設定して、永続ボリュームのバックエンドとなるFlexVolボリュームでONTAPのボリューム自動拡張を有効にします。使用済みスペースが設定されたしきい値に達すると、ONTAPは自動的にFlexVolを最大サイズまで拡張します。これにより、FlexVolを手動でサイズ変更することなく、スナップショットとストレージ オーバーヘッドのための追加容量が提供されます。

ONTAPボリュームの自動サイズ拡張は、バックエンドのFlexVolのみを拡張します。PersistentVolumeClaim（PVC）のサイズは変更しません。これは、PVCを拡張する"Trident自動拡張ポリシー"とは異なります。

Tridentは、ボリュームの作成、複製、サイズ変更時、および管理対象のボリュームのインポート時に、バックエンドの自動サイズ設定を適用します。

要件

- Trident 26.10以降。
- 以下のドライバーのいずれか：
  - `ontap-san`iSCSI、NVMe、またはファイバーチャネルプロトコル（FCP）に対応。
  - `ontap-san-economy`iSCSIを使用する場合。
- オンプレミスONTAP：
  - ONTAP REST APIを使用するバックエンドの場合は、ONTAP 9.12.1以降。NVMeバックエンドはONTAP REST APIを使用する必要があります。
  - ONTAPI（ZAPI）を使用するバックエンドの場合、Tridentでサポートされている任意のONTAPバージョン。"Tridentの要件"を参照してください。

制限

- ONTAPボリュームの自動サイズ調整はFlexVolsを縮小しません。
- 自動サイズ調整はバックエンドごとに設定されます。異なるワークロードに対して異なる設定を使用するために、別々のバックエンドを作成します。
- ONTAPボリュームの自動サイズ調整は、ONTAP NASドライバ、ASA r2システム、NetApp AFXやその他のディスアグリゲーテッドONTAPシステム、またはGoogle Cloud NetApp Volumesではサポートされていません。これらの構成では、バックエンドの作成が失敗します。
- ONTAPボリュームの自動拡張では、TridentMirrorRelationshipレプリケーションおよびMetroCluster構成はサポートされていません。Tridentはこれらの構成を妨げるものではありません。
- `ontap-san-economy` の場合、 `maximumSize` はパーセンテージである必要があります。
- `ontap-san`の場合、絶対的な `maximumSize`はサイズ制限として機能します。バックエンドのFlexVolがその制限を超える必要がある場合、PVCの作成または拡張は失敗します。
- 1つの `ontap-san-economy` FlexVolには、複数のPVC用のLUNを含めることができます。各ボリューム操作は、現在のバックエンドの自動サイズ設定をFlexVol全体に適用します。複数のバックエンドがFlexVolsを共有する場合、最新の操作によって設定が決定されます。 `ontap-san-economy`の場合は、すべてのバックエンドで同じデフォルトが使用されるように、 `mode: grow`のみを設定し、`growThresholdPercent`と `maximumSize`は省略してください。

ONTAPボリュームの自動サイズ調整オプション

バックエンド設定の最上位の `ontapAutosize`ブロックで、これらのオプションを設定します。

オプション	説明	の形式で入力し	必須	`ontapAutosize` が存在する場合のデフォルト	例
`mode`	自動サイズ調整を有効または無効にします。 `grow` は自動サイズ調整を有効にし、 `off` は無効にします。	文字列	いいえ	`off`	`grow` または `off`
`growThresholdPercent`	ONTAPがFlexVolを拡張する使用済みスペースの割合です。有効な値は2から99までです。	整数	いいえ	`85`	`90`
`maximumSize`	最大FlexVolサイズは、そのサイズに対する割合、または絶対サイズとして表されます。絶対サイズは、 `ontap-san` でのみサポートされます。	パーセンテージまたはKubernetesの数量	いいえ	`"20%"`	`"20%"` または `"2Ti"`

`ontapAutosize`ブロックが存在する場合、デフォルトが適用されます。ブロックを省略した場合、TridentはFlexVolsの自動サイズ設定を変更しません。

Tridentはパーセンテージ `maximumSize`を絶対サイズに変換し、ボリュームのサイズが変更されるたびに再計算します。たとえば、 `maximumSize: "20%"`を使用すると、100 GiBのFlexVolは120 GiBに拡張できます。

ONTAPボリュームの自動拡張を設定

1. バックエンド設定に `ontapAutosize`ブロックを追加します：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  backendName: ontap-san-autosize
  managementLIF: 10.0.0.1
  svm: svm_san
  sanType: iscsi
  ontapAutosize:
    mode: grow
    growThresholdPercent: 85
    maximumSize: "20%"
```

```
  credentials:
    name: backend-tbc-ontap-san-secret
```

2. バックエンドを作成します：

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

3. このバックエンドからボリュームをプロビジョニングした後、そのTridentボリュームに適用された自動サイズ設定を確認します：

```
kubectl get tridentvolume <pv-name> -n trident \
  -o jsonpath='{.config.ontapAutosizeMode}
{.config.ontapAutosizeGrowThresholdPercent}
{.config.ontapAutosizeMaximumSize}'
```

上記の構成の場合、期待される出力は次のとおりです：

```
grow 85 20%
```

バックエンドで自動サイズ設定を更新する

バックエンドを更新することで、 `ontapAutosize`ブロックを追加、変更、または削除できます。この変更によって、既存のFlexVolsが即座に更新されることはありません：

- `ontap-san`の場合、新しいボリューム、クローン、および管理対象インポートでは、新しい設定が使用されます。既存のボリュームは、サイズ変更時も含め、現在の設定が保持されます。
- `ontap-san-economy`の場合、次回Tridentがその中にボリュームを作成、サイズ変更、クローン作成、またはインポートする際に、共有FlexVolは新しい設定を受け取ります。

設定例

ontap-san FCPと絶対最大サイズ

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_san",
  "username": "vsadmin",
  "password": "<password>",
  "sanType": "fcp",
  "ontapAutosize": {
    "mode": "grow",
```

```
    "growThresholdPercent": 90,
    "maximumSize": "2Ti"
  }
}
```

「**ONTAP - SAN - エコノミー**」

この例では `mode`のみを設定しているため、Tridentは `growThresholdPercent`と `maximumSize`にデフォルト値を使用します：

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "svm": "svm_san",
  "username": "vsadmin",
  "password": "<password>",
  "ontapAutosize": {
    "mode": "grow"
  }
}
```

よくある質問

TridentのAutogrowポリシーでONTAPボリュームのautosizeを使用できますか？

はい。Trident Autogrow PolicyはPVCを拡張し、Tridentはサイズ変更のたびにONTAPのautosizeの最大値を再計算します。絶対的な `maximumSize`を `ontap-san`とともに使用する場合は、バックエンドのFlexVolがその制限内に収まるように、ポリシーの `maxSize`を十分に低く設定してください。TridentがSnapshotリザーブとメタデータ用のスペースを追加するため、FlexVolはPVCよりも大きくなります。

`ontapAutosize`を省略する場合と `mode: off`を設定する場合の違いは何ですか？

`ontapAutosize`を省略した場合、TridentはONTAPのFlexVolsの自動サイズ設定を変更しません。 `mode: off`を設定した場合、Tridentは、作成または更新するFlexVols（ソースから自動サイズ設定を継承したクローンを含む）の自動サイズ設定を無効にします。

PVCの拡張が `ontapAutosize maximum ... is below required FlexVol size`エラーで失敗したのはなぜですか？

ボリュームは絶対値 `maximumSize`を使用しており、拡張されたPVCにはより大きなFlexVolが必要です。`ontap-san`ボリュームは作成時の最大サイズを保持するため、バックエンドを更新しても既存のボリュームの設定は変更されません。新規ボリュームの場合は、 `maximumSize`を増やすか、パーセンテージを使用してください。

ボリューム作成が自動サイズエラーで失敗したのはなぜですか？

ONTAPが自動サイズ設定を拒否し、PVCイベントにはONTAPからの理由が含まれています。たとえば、最大サイズがONTAPシステムでサポートされている最大のFlexVolサイズを超える可能性があります。バックエンドで `maximumSize`を下げます。Tridentは、次回ボリュームの作成を試行するときに更新された値を使用します。

関連情報

- "Trident Autogrowポリシーを使用した自動ボリューム拡張"
- "ONTAPコマンドリファレンス：volume autosize"

自動拡張ポリシーの管理

自動成長ポリシーを作成した後、それらを表示、更新、削除することができます。特定のポリシーを使用しているボリュームを監視することもできます。

自動拡張ポリシーを表示

すべてのポリシーを一覧表示する

`kubectl`を使用して、クラスター内のすべての自動拡張ポリシーを一覧表示します：

```
kubectl get tridentautogrowpolicy
```

あるいは、 `tridentctl`：

```
tridentctl get autogrowpolicy
```

ポリシーの詳細を表示

ポリシーの完全な仕様とステータスを表示するには：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

YAML 形式でポリシーとそれに関連付けられたボリュームを表示するには：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

自動拡張ポリシーを更新する

既存のポリシーを変更して、しきい値、増加量、または最大サイズを変更できます。変更は、ポリシーを使用するすべてのボリュームに対して直ちに有効になります。

重要	変更内容は、当該ポリシーを使用するすべてのボリュームに影響します。可能であれば、まず非本番環境で変更をテストしてください。

手順

1. ポリシーを編集します：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

2. 必要に応じて `spec`フィールドを変更します：

```
spec:
  usedThreshold: "75%"     # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

3. 保存して終了します。変更は直ちに有効になります。

更新に関する考慮事項

- 即時適用： このポリシーを使用するすべてのボリュームは、次回の成長評価時に新しいパラメータを採用します。
- ボリュームの再起動は不要： 変更は次回の拡張操作に適用されます。
- **Test first:** 可能な場合は、非本番環境で変更を検証します。
- *変更を伝える：*共有ポリシーを変更するときにチームに通知します。

自動拡張ポリシーを削除する

自動拡張ポリシーは、ボリュームがアクティブに使用されている間、誤って削除されないようにファイナライザー保護を使用します。

手順

1. ポリシーを削除します：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. ボリュームがまだポリシーを使用している場合、削除は `Deleting`状態になります。影響を受けるボリュームを確認してください。

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 影響を受ける各ボリュームからポリシーを削除します。次のいずれかのオプションを選択します：

- オプション**A**：明示的に自動拡張を無効にする アノテーションを `"none"`に設定します：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

- オプション**B**：注釈を完全に削除する：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

削除動作

シナリオ	動作
このポリシーを使用するボリュームはありません	ポリシーは直ちに削除されます。
ボリュームはポリシーを使用します	ポリシーが `Deleting`状態になります。ファイナライザーは、すべてのボリュームが削除されるまで完了をブロックします。
すべてのボリュームがポリシーから削除されます	ファイナライザーが削除され、ポリシーが削除されます。

自動拡張ポリシーの使用状況を監視する

ポリシーを使用してボリュームをチェックする

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

ボリュームが使用するポリシーを確認する

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

ポリシーイベントを監視する

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

サポートされるプロトコル

Autogrow は次のストレージプロトコルをサポートしています：

- NFS
- iSCSI
- FCP
- NVMe

メモ	SANボリュームの場合、構成された `growthAmount`が50 MiB以下である場合、Tridentは、結果のサイズが `maxSize`を超えない限り、サイズ変更操作の増加量を自動的に51MBに増加します。

サポートされているバックエンド

AutogrowはONTAPバックエンドをサポートしています。AutogrowはNASボリュームとSANボリュームの両方に対応するGoogle Cloud NetApp Volumes（GCNV）もサポートしています。

既知の制限事項

- **ONTAP NVMe raw**ブロックボリューム： ONTAP 9.16.1より前のバージョンで作成されたボリュームは、自動拡張をサポートしていません。
- 既存のボリューム（ブラウンフィールド展開）： 有効な自動拡張ポリシーが適用されていても、既存のボリュームでは自動拡張が機能しない場合があります。この制限は、ボリュームパブリケーションの移行が進行中であることに起因します。移行が完了したことを確認するには、Trident コントローラーログで `"Migration completed"`メッセージを確認してください。

よくある質問

Tridentはいつ閾値を評価しますか？

Tridentはボリュームの使用状況を継続的に監視します。使用容量が `usedThreshold`を超えると、Tridentは内部サイズ変更要求を作成し、設定された `growthAmount`でボリュームを拡張します。

たとえば、このポリシーは容量の80%で拡張をトリガーし、ボリュームを毎回10%ずつ最大500 GiBまで増加させます：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

ボリュームがすでにプロビジョニングされた後にポリシーを適用できますか？

はい。自動拡張ポリシーはいつでも作成でき、 `trident.netapp.io/autogrowPolicy`アノテーションを追加または更新することで既存のPVCに適用できます。PVCまたはStorageClassを再作成する必要はありません。

既存の PVC にポリシーを適用するには：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

既存のStorageClassにポリシーを適用するには：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

StorageClassとPVCの両方に自動拡張ポリシーを設定するとどうなりますか？

PVC 注釈は常に優先されます。PVC に `trident.netapp.io/autogrowPolicy`注釈がある場合、 Trident はStorageClassの指定に関係なくその値を使用します。詳細については、"ポリシーの優先順位"を参照してください。

例えば、このStorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

そして、この PVC は StorageClass ポリシーを上書きします：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Tridentは `logs-policy`を `database-pvc`に使用し、 `standard-agp`は使用しません。

特定のボリュームの自動拡張を無効にするにはどうすればよいですか？

PVC アノテーションを `"none"`に設定します。この設定は、そのボリュームに対する StorageClass レベルのポリシーを上書きします：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

自動拡張が無効になっていることを確認できます：

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

期待される出力

```
none
```

ボリュームが**maxSize**に達するとどうなりますか？

Tridentはボリュームの拡張を停止します。使用量が上限を超えて増加し続けても、そのボリュームに対するサイズ変更要求はそれ以上発生しません `usedThreshold`。

例えば、このポリシーでは、Trident はボリュームが 100 GiB に達した後、それ以上拡張しません：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

無制限に成長させるには、 `maxSize`を省略するか、 `0`に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
```

```
  usedThreshold: "85%"
  growthAmount: "10%"
```

ボリュームを再起動せずにポリシーを変更できますか？

はい。ポリシーを更新すると、そのポリシーを使用しているすべてのボリュームは、次回の成長評価時に新しいパラメータを適用します。ボリュームの再起動は不要です。

既存のポリシーを更新するには：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

必要に応じてフィールドを変更します：

```
spec:
  usedThreshold: "75%"     # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

保存して終了します。更新されたポリシーを確認します：

```
kubectl get tridentautogrowpolicy production-db-policy
```

期待される出力

```
NAME                    USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy    75%              20%             Success
```

ポリシーが失敗状態になっているのはなぜですか？

A `Failed`状態は、ポリシー仕様に検証エラーが含まれていることを示します。エラーの詳細を表示するには、次のコマンドを実行します：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

一般的な原因には、無効な `usedThreshold`（1〜99%である必要があります）、 `growthAmount`が`maxSize`を超えている場合、または無効なKubernetes数量フォーマットなどがあります。仕様を修正して再適用してください：

```
kubectl apply -f autogrow-policy.yaml
```

ポリシーを削除できないのはなぜですか？

ポリシーはファイナライザー保護を使用します。ボリュームがまだポリシーを使用している場合、削除は`Deleting`状態になり、ポリシーからすべてのボリュームを削除するまで待機します。

影響を受けるボリュームを特定します：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

次に、各 PVC から注釈を削除します：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

すべてのボリュームを削除すると、Trident はファイナライザーを解放し、ポリシーを削除します。

autogrowはどのバックエンドをサポートしていますか？

AutogrowはONTAPバックエンドをサポートしています。また、NASボリュームとSANボリュームの両方について、Google Cloud NetApp Volumes（GCNV）もサポートしています。

Autogrowは、NFS、iSCSI、FCP、およびNVMeプロトコルをサポートしています。NVMe rawブロックボリュームには、ONTAP 9.16.1以降が必要です。

既存のボリュームをブラウンフィールド環境で使用している場合、自動拡張が有効になる前にボリューム公開の移行を完了する必要がある場合があります。移行状況を確認するには、Trident コントローラーログを確認してください：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
completed"
```

次のStorageClass例では、NAS および SAN バックエンドに設定された自動拡張を示しています：

NASバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
```

```
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SANバックエンド

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN ボリュームの最小増加量はどれくらいですか？

SAN ボリュームの場合、有効な最小増加量は 51 MB です。 `growthAmount`を 50 MiB 以下に設定すると、Trident はサイズ変更操作のために自動的に増加量を 51 MB に増加します。

例えば、このポリシーは `growthAmount`の `"40Mi"`を設定しますが、Tridentはこれを使用するすべてのSANボリュームに51MBの拡張を適用します（

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

この自動調整を回避するには、 `growthAmount` を 50 MiB を超える値に設定します：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
```

```
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

Tridentボリュームのインポート

`tridentctl import`を使用するか、Tridentインポート
アノテーションを使用して永続ボリューム要求（PVC）を作成することで、既存のストレージ
ボリュームをKubernetes PVとしてインポートできます。

概要と考慮事項

Tridentにボリュームをインポートする目的は次のとおりです。

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションにはデータセットのクローンを使用
- 障害が発生したKubernetesクラスタを再構築します
- ディザスタリカバリ時にアプリケーションデータを移行

考慮事項

ボリュームをインポートする前に、次の考慮事項を確認してください。

- Tridentでインポートできるのは、RW（読み取り/書き込み）タイプのONTAPボリュームのみです。DP（データ保護）タイプのボリュームはSnapMirrorデスティネーションボリュームです。ボリュームをTridentにインポートする前に、ミラー関係を解除する必要があります。
- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

警告	Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- PVCで指定する必要がありますが、`StorageClass`Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、最初にににに設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーが `delete`の場合、PVが削除されると、Tridentはストレージボリ

ュームを削除します。

- デフォルトでは、Trident は PVC を管理し、バックエンドの FlexVol ボリュームおよび LUN または qtree の名前を変更します。 `--no-manage` フラグを渡すと管理対象外のボリュームをインポートでき、 `--no-rename` フラグを渡すとボリューム名を保持できます。
  - `--no-manage*` - を使用する場合 `--no-manage`フラグが設定されている場合、 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。
  - `--no-rename*` - `--no-rename`フラグを使用する場合、`Trident` はボリュームをインポートする際に既存のボリューム名を保持し、ボリュームのライフサイクルを管理します。このオプションは、`ontap-nas`、 `ontap-san`（ASA r2 システムを含む）、 `ontap-san-economy`、および `ontap-nas-economy`（qtree インポート）ドライバーでのみサポートされています。

ヒント	これらのオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、それ以外の場合は Kubernetes の外部でストレージ ボリュームのライフサイクルを管理する場合に役立ちます。

- PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

ボリュームをインポートします

`tridentctl import`を使用するか、`Trident`インポートアノテーションを使用して`PVC`を作成することで、ボリュームをインポートできます。

メモ	PVC アノテーションを使用する場合は、 `tridentctl`をダウンロードしたり使用してボリュームをインポートしたりする必要はありません。

tridentctlの使用

手順

1. PVCを作成するために使用するPVCファイル（例： `pvc.yaml`）を作成します。PVCファイルには `name`、 `namespace`、 `accessModes`、および `storageClassName`を含める必要があります。必要に応じて、PVC定義で `unixPermissions`を指定することもできます。

   最小仕様の例を次に示します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
```

```
  storageClassName: my_storage_class
```

メモ	必須パラメータのみを入力してください。PV名やボリュームサイズなどの追加パラメータは、インポートコマンドの失敗の原因となる可能性があります。

2. 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) 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について、次のボリュームインポートの例を確認してください。

ONTAP NASおよびONTAP NAS FlexGroup

Tridentは、ドライバと `ontap-nas-flexgroup`ドライバを使用したボリュームインポートをサポートしてい `ontap-nas`ます

メモ

- `ontap-nas-economy`ドライバを使用してqtreeをインポートするには、NFS qtreeのインポートを参照してください。Tridentがqtreeをインポートする際、そのqtreeを含む親FlexVolもバケットとしてインポートします。qtreeは個別にインポートする必要があります。
- 。 `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

ドライバを使用して作成される各ボリューム `ontap-nas`は、ONTAPクラスタ上のFlexVol volumeになります。ドライバを使用したFlexVolボリュームのインポート `ontap-nas`も同様に機能します。ONTAPクラスタにすでに存在するFlexVolボリュームは、PVCとしてインポートできます `ontap-nas`。同様に、FlexGroupボリュームはPVCとしてインポートできます `ontap-nas-flexgroup`。

tridentctl を使用した ONTAP NAS の例

次の例は、 `tridentctl`を使用して管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています。

管理対象ボリューム

次の例は、という名前のボリュームをインポートします `managed_volume` という名前のバックエンドで `ontap_nas`：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |              BACKEND UUID              | STATE   | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+---------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
```

```
+----------+----------------------------------------+--------+---------+
```

管理対象外のボリューム

引数を使用した場合 `--no-manage`、Tridentはボリュームの名前を変更しません。

次に、をインポートする例を示します `unmanaged_volume` をクリックします `ontap_nas` バックエンド：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

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。

管理対象ボリューム

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>
```

管理対象外のボリューム

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>
```

qtree をインポート

メモ	qtree インポートは Trident 26.10 のテクニカルプレビュー機能です。テクニカルプレビュー機能は、本番環境での使用はサポートされていません。

`ontap-nas-economy`ドライバを使用して、既存のONTAP qtreeを Tridentにインポートできます。qtreeを特定するには、それを含むバックエンドのUUIDとともに、 `flexvol/qtree`形式のFlexVolボリューム名とqtree名を使用します。<< 概要と考慮事項>>の一般的なインポートの考慮事項は、qtreeのインポートにも適用されます。

tridentctlを使用して**qtree**をインポートする

qtreeを `flexvol/qtree`形式でボリューム名として指定します。次の例では、バックエンド `ontap_nas_economy`上のFlexVol volume `flexvol1`からqtree `qtree1`をインポートします。

```
tridentctl import volume ontap_nas_economy flexvol1/qtree1 -f <path-to-
pvc-file>
```

qtree の名前を変更せずにインポートするには、 `--no-rename`フラグを追加してください。Trident がライフサイクルを管理せずに qtree をインポートするには、 `--no-manage`フラグを追加してください。

PVCアノテーションを使用して**qtree**をインポートする

以下のアノテーションを付けてPVCを作成します：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "flexvol1/qtree1"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

既存のFlexVol volumeとqtree名を保持するには、 `trident.netapp.io/importNoRename: "true"`アノテーションを追加します。qtreeをアンマネージドとしてインポートするには、 `trident.netapp.io/notManaged: "true"`アノテーションを追加します。

命名と管理の動作

管理インポート（デフォルト）

Trident は、qtree の名前を PVC 名に合わせて変更し、FlexVol ボリュームの名前を `ontap-nas-economy` 命名規則に合わせて変更します。ジャンクションパスは変更されません。

`--no-rename`

Trident は既存の FlexVol volume と qtree 名を保持します。名前はすでに Trident の命名規則に従っている必要があります。従っていない場合、Trident は qtree の削除またはマウント操作を管理できません。

`--no-manage`

Trident は qtree を参照しますが、そのライフサイクルは管理しません。名前は Trident の命名規則に従う必要があり、Trident は qtree の自動エクスポートポリシーを使用または管理しません。

エクスポートポリシーの動作

バックエンドが自動エクスポートポリシーを使用している場合、インポートされたqtreeはqtreeごとのエクスポートポリシーモデルを使用し、各qtreeが独自のエクスポートポリシーを持ちます。Tridentはインポート時

にqtreeに空のエクスポートポリシーを割り当てます。公開時に、自動エクスポートが有効になっており、既存のFlexVolエクスポートポリシーが `default`または空の場合、TridentはFlexVolボリュームをバックエンドごとのエクスポートポリシーに更新します。

注意	FlexVol ボリュームのエクスポートポリシーを更新すると、同じ FlexVol ボリューム内で現在マウントされている他の qtree に影響を与える可能性があります。qtree をインポートするのは、FlexVol ボリューム内の qtree がマウントされていない場合のみにしてください。
警告	2 つの Kubernetes クラスターを同じアクティブな SVM に向けないでください。また、別のクラスターが引き続き管理している qtree をインポートしないでください。両方のクラスターが同じボリュームを管理している場合、競合する操作によってデータが破損または消失する可能性があります。

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。

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：

管理対象ボリューム

管理対象ボリュームの場合、TridentはFlexVol volumeの名前を形式に、FlexVol volume内のLUNの名前をに `lun0`変更 `pvc-<uuid>`します。

次に、バックエンドにあるFlexVol volume `ontap_san_default`をインポートする例を示し `ontap-san-managed`ます。

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+-----------------------------------------+--------+---------------
+----------+-----------------------------------------+--------+---------+
```

管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `ontap_san` バックエンド：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、エラーが表示されます。`LUN already mapped to initiator(s) in this group`。ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economyの例

以下の例は、 `ontap-san-economy`バックエンドの管理ボリュームと非管理ボリュームをインポートする方法を示しています。

管理対象ボリューム

管理ボリュームをインポートすると、TridentがFlexVolの所有権を取得し、名前を変更します。同じFlexVolから複数のLUNをインポートする場合は、この名前変更を考慮する必要があります。

次の例では、 `lun1`をFlexVol `toimport`から `vol-managed-saneco`という名前の管理ボリュームとしてインポートします：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

インポート後 `lun1`、TridentはFlexVolの名前を変更します（例： `trident_lun_pool_xyz`）。同じFlexVolから追加のLUNをインポートするには、新しいFlexVol名を使用します：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

メモ	`ontap-san-economy`バックエンドは一度に1つのLUNをインポートします。スクリプトを使用すれば、複数のインポートを自動化できます。

管理対象外のボリューム

管理されていないボリュームをインポートすると、TridentはFlexVolの所有権を取得しません。ただし、FlexVolとLUNはTridentの命名規則に従う必要があります。

FlexVol命名形式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX`は、バックエンド設定の `storagePrefix`の値です。デフォルトは `trident`です。
- `RANDOMSTRING`は任意の文字列です。

LUN の命名要件

LUN には `lun0`という名前を付ける必要があります。

例

もしあなたの `storagePrefix`が `xyz`の場合、LUNへの完全パスは次のとおりです：

```
trident_lun_pool_xyz_randomstring/lun0
```

要素（ **Element** ）

Tridentは、NetApp Elementソフトウェアとドライバを使用したNetApp HCIボリュームインポートをサポートしています `solidfire-san`。

メモ	Element ドライバではボリューム名の重複がサポートされます。ただし、ボリューム名が重複している場合、Tridentはエラーを返します。回避策としてボリュームをクローニングし、一意のボリューム名を指定して、クローンボリュームをインポートします。

次に、をインポートする例を示します `element-managed` バックエンドのボリューム `element_default`。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files の特長

Tridentはドライバを使用したボリュームインポートをサポートしてい `azure-netapp-files`ます。

メモ	Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、マウントパスがの場合などです `10.0.0.2:/importvol1`、ボリュームのパスはです `importvol1`。

次に、をインポートする例を示します `azure-netapp-files` バックエンドのボリューム `azurenetappfiles_40517` を指定します `importvol1`。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetAppボリューム

Tridentはドライバを使用したボリュームインポートをサポートしてい `google-cloud-netapp-volumes`ます。

次の例では、ボリューム `testvoleasiaeast1`を持つバックエンド `backend-tbc-gcnv1`上のボリュームをインポ

ートします。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

次の例は、同じリージョンに2つのボリュームがある場合にボリュームをインポートし `google-cloud-netapp-volumes`ます。

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

Tridentのボリューム名とラベルのカスタマイズ

Tridentでは、作成したボリュームにわかりやすい名前とラベルを割り当てることができます。これにより、ボリュームを特定し、それぞれのKubernetesリソース（PVC）に簡単にマッピングできます。また、バックエンドレベルでテンプレートを定義してカスタムボリューム名とカスタムラベルを作成することもできます。作成、インポート、またはクローンを作成するボリュームは、テンプレートに準拠します。

作業を開始する前に

カスタマイズ可能なボリューム名とラベルのサポート：

- ボリュームの作成、インポート、クローニングの各処理。
- の場合 `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。
- の場合 `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

- カスタム 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。
- カスタムラベルは、 `ontap-san`、 `ontap-nas`、 そして `ontap-nas-flexgroup` ドライバー。
- カスタム ボリューム名は既存のボリュームには適用されません。

カスタマイズ可能なボリューム名の主な動作

- 名前テンプレートの無効な構文が原因でエラーが発生した場合、バックエンドの作成は失敗します。ただし、テンプレートアプリケーションが失敗した場合は、既存の命名規則に従ってボリュームに名前が付けられます。
- バックエンド構成の名前テンプレートを使用してボリュームの名前が指定されている場合、ストレージプレフィックスは適用されません。任意のプレフィックス値をテンプレートに直接追加できます。

名前テンプレートとラベルを使用したバックエンド構成の例

カスタム名テンプレートは、ルートレベルまたはプールレベルで定義できます。

ルートレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
```

```
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

プールレベルの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

名前テンプレートの例

*例1 *：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

*例2 *：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

考慮すべきポイント

1. ボリュームインポートの場合、既存のボリュームに特定の形式のラベルがある場合にのみラベルが更新されます。例：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 管理対象ボリュームのインポートの場合、ボリューム名はバックエンド定義のルートレベルで定義された名前テンプレートの後に続きます。
3. Tridentでは、storageプレフィックスを指定したスライス演算子の使用はサポートされていません。
4. テンプレートによってボリューム名が一意にならない場合、Tridentではいくつかのランダムな文字が追加されて一意のボリューム名が作成されます。
5. NASエコノミーボリュームのカスタム名の長さが64文字を超える場合、Tridentは既存の命名規則に従ってボリュームに名前を付けます。他のすべてのONTAPドライバでは、ボリューム名が名前の上限を超えると、ボリュームの作成プロセスが失敗します。

名前空間をまたいで**Trident NFS**ボリュームを共有する

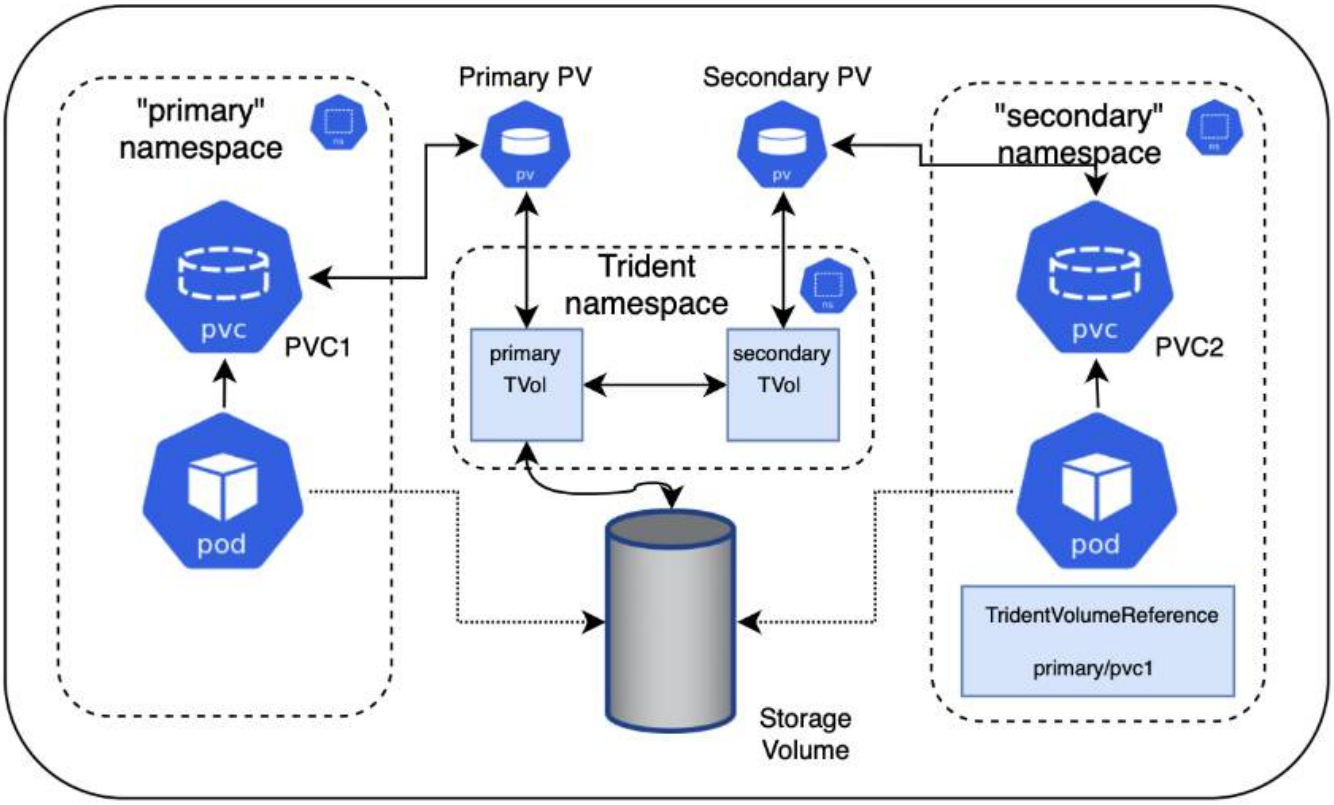
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

の機能

TridentVolumeReference CRを使用すると、1つ以上のKubernetesネームスペース間でReadWriteMany（RWX）NFSボリュームを安全に共有できます。このKubernetesネイティブ解決策 には、次のようなメリットがあります。

- セキュリティを確保するために、複数のレベルのアクセス制御が可能です
- すべてのTrident NFSボリュームドライバで動作
- tridentctlやその他の非ネイティブのKubernetes機能に依存しません

この図は、2つのKubernetesネームスペース間でのNFSボリュームの共有を示しています。



クイックスタート

NFSボリューム共有はいくつかの手順で設定できます。

1 ボリュームを共有するようにソース**PVC**を設定します

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2 デスティネーションネームスペースに**CR**を作成する権限を付与します

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3 デスティネーションネームスペースに**TridentVolumeReference**を作成します

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4 宛先名前空間に下位**PVC**を作成します

宛先名前空間の所有者は、送信元PVCからのデータソースを使用する下位PVCを作成します。

ソースネームスペースとデスティネーションネームスペースを設定します

セキュリティを確保するために、ネームスペース間共有では、ソースネームスペースの所有者、クラスタ管理

者、および宛先ネームスペースの所有者によるコラボレーションとアクションが必要です。ユーザロールは各手順で指定します。

手順

1. ソース名前空間の所有者： PVCを作成します (`pvc1`）をソースネームスペースに追加し、デスティネーションネームスペースとの共有権限を付与します (`namespace2`)を使用します `shareToNamespace` アノテーション

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Tridentは、PVとそのバックエンドNFSストレージボリュームを作成します。

メモ

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例： `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- を使用して、すべてのネームスペースに共有できます `*`。例： `trident.netapp.io/shareToNamespace: *`
- PVCを更新してを含めることができます `shareToNamespace` アノテーションはいつでも作成できます。

2. クラスター管理者: 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. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
```

```
  pvcNamespace: namespace1
```

4. 宛先名前空間の所有者： PVCを作成します (`pvc2`）をデスティネーションネームスペースに展開します (`namespace2`)を使用します `shareFromPVC` 送信元PVCを指定する注釈。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

メモ　宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

結果

Tridentはデスティネーションpvcのアノテーションを読み取り `shareFromPVC`、ソースPVストレージリソースを共有する独自のストレージリソースのない下位ボリュームとしてデスティネーションPVを作成します。宛先PVCとPVは、通常どおりバインドされているように見えます。

共有ボリュームを削除

複数のネームスペースで共有されているボリュームは削除できます。Tridentは、ソースネームスペース上のボリュームへのアクセスを削除し、そのボリュームを共有する他のネームスペースへのアクセスを維持します。このボリュームを参照しているネームスペースをすべて削除すると、Tridentによってボリュームが削除されます。

使用 **`tridentctl get`** 下位のボリュームを照会する

[`tridentctl`ユーティリティを使用して、 `get`コマンドを実行すると、従属ボリュームを取得できます。詳細については、link：../trident-reference/trident-cl.html[`tridentctl`コマンドとオプション]を参照してください。

```
Usage:
  tridentctl get [option]
```

フラグ：

- `-h, --help`：ボリュームのヘルプ。

- `--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`:クエリをボリュームの下位に制限します。

制限

- Tridentでは、デスティネーションネームスペースが共有ボリュームに書き込まれないようにすることはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace` または `shareFromNamespace` 注釈またはを削除します `TridentVolumeReference` CR。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- のデモをご覧ください "ネットアップTV"。

名前空間をまたいだ**Trident**ボリュームのクローン作成

Tridentを使用すると、同じKubernetesクラスタ内の別のネームスペースから既存のボリュームまたはボリュームSnapshotを使用して新しいボリュームを作成できます。

前提条件

ボリュームをクローニングする前に、ソースとデスティネーションのバックエンドのタイプとストレージクラスが同じであることを確認してください。

メモ	名前空間をまたがるクローン作成は、 `ontap-san`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

ボリュームクローニングはわずか数ステップでセットアップできます。

1. ボリュームのクローンを作成するためのソース**PVC**の設定

ソースネームスペースの所有者は、ソースPVCのデータにアクセスする権限を付与します。

2. デスティネーションネームスペースに**CR**を作成する権限を付与します

クラスタ管理者が、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。

3. デスティネーションネームスペースに**TridentVolumeReference**を作成します

宛先名前空間の所有者は、送信元PVCを参照するためにTridentVolumeReference CRを作成します。

4 デスティネーションネームスペースにクローン**PVC**を作成します。

宛先ネームスペースの所有者は、PVCを作成して、送信元ネームスペースからPVCを複製します。

ソースネームスペースとデスティネーションネームスペースを設定します

セキュリティを確保するために、ネームスペース間でボリュームをクローニングするには、ソースネームスペースの所有者、クラスタ管理者、およびデスティネーションネームスペースの所有者が協力して対処する必要があります。ユーザロールは各手順で指定します。

手順

1. ソースネームスペース所有者：(`pvc1`ソースネームスペースにPVCを作成(`namespace1`）。注釈(`namespace2`を使用して、デスティネーションネームスペースと共有する権限を付与します。`cloneToNamespace`

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Tridentは、PVとそのバックエンドストレージボリュームを作成します。

メモ

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。たとえば、`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`です。
- を使用して、すべてのネームスペースと共有できます `*`。例えば、`trident.netapp.io/cloneToNamespace: *`
- PVCはいつでも更新してアノテーションを含めることができます `cloneToNamespace`。

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. 宛先ネームスペースの所有者： ソースPVCを参照するTridentVolumeReference CRを宛先ネームスペースに作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先ネームスペースの所有者：(pvc2`宛先ネームスペースに `cloneFromNamespace`PVCを作成(`namespace2`)。または cloneFromSnapshot`アノテーションを使用して、送信元PVCを指定します `cloneFromPVC。

```
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

- ONTAP NASエコノミードライバを使用してプロビジョニングされたPVCでは、読み取り専用クローンはサポートされません。

SnapMirrorによるボリュームのレプリケート

Tridentでは、ディザスタリカバリ用にデータをレプリケートするために、ピア関係にあるクラスタのソースボリュームとデスティネーションボリュームの間のミラー関係をサポートしています。 Trident Mirror Relationship (TMR) と呼ばれる名前空間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（PVC）間のミラー関係を作成する
- ボリューム間のミラー関係の削除
- ミラー関係を解除する

- 災害時（フェイルオーバー）にセカンダリボリュームを昇格する
- クラスタからクラスタへのアプリケーションのロスレス移行の実行（計画的なフェイルオーバーまたは移行時）

レプリケーションの前提条件

作業を開始する前に、次の前提条件を満たしていることを確認してください。

ONTAP クラスタ

- * Trident *：Tridentバージョン22.10以降が、バックエンドとしてONTAPを利用するソースとデスティネーションの両方のKubernetesクラスタに存在している必要があります。
- ライセンス：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデスティネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

  ONTAP 9.10.1 以降、すべてのライセンスは、複数の機能を有効にする単一のファイルである NetApp ライセンス ファイル (NLF) として提供されます。詳細については、を参照してください "ONTAP Oneに含まれるライセンス" 。

メモ	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にピア関係が設定されている必要があります。詳細については、を参照してください "クラスタと SVM のピアリングの概要" 。

重要	2つのONTAPクラスタ間のレプリケーション関係で使用されるSVM名が一意であることを確認してください。

- * TridentとSVM *：ピア関係にあるリモートSVMをデスティネーションクラスタのTridentで使用できる必要があります。

サポートされるドライバ

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 テクノロジーによるボリューム 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)

メモ	SnapMirrorを使用したボリュームレプリケーションは、ASA r2システムではサポートされていません。ASAr2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の詳細" 。

ミラー**PVC**の作成

以下の手順に従って、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の間にミラー関係を作成します。

手順

1. プライマリKubernetesクラスタで次の手順を実行します。
   a. パラメータを指定してStorageClassオブジェクトを作成し `trident.netapp.io/replication:`

`true` ます。

例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "nfs"
  trident.netapp.io/replication: "true"
```

b. 以前に作成したStorageClassを使用してPVCを作成します。

例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. ローカル情報を含むMirrorRelationship CRを作成します。

例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Tridentは、ボリュームの内部情報とボリュームの現在のデータ保護（DP）状態をフェッチし、MirrorRelationshipのstatusフィールドに値を入力します。

d. TridentMirrorRelationship CRを取得して、PVCの内部名とSVMを取得します。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. セカンダリKubernetesクラスタで次の手順を実行します。

a. 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. デスティネーションとソースの情報を含むMirrorRelationship CRを作成します。

例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
```

```
  replicationPolicy: MirrorAllSnapshots
  replicationSchedule: 5min
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

TMR `spec`には、4つの最上位フィールドがあります：目的の関係状態を設定する `state`（[ボリュームレプリケーションの状態]を参照）、ローカルPVCをリモートボリュームハンドルに関連付ける `volumeMappings`、および以下で説明するオプションの `replicationPolicy`と `replicationSchedule`フィールドです。

> `replicationPolicy`および `replicationSchedule`フィールドはオプションです。 `replicationPolicy`にはONTAPの非同期SnapMirrorポリシーの名前を設定し、 `replicationSchedule`にはONTAPスケジュールの名前を設定します。いずれかのフィールドを省略した場合、Tridentはバックエンドで設定されている対応するレプリケーションポリシーまたはスケジュールを使用します。これらのフィールドは関係を作成する際に設定します。関係が確立された後にポリシーやスケジュールを変更することはできません。異なる設定を使用するには、TMRを削除して再作成します。

Tridentは、設定されたレプリケーションポリシーとスケジュールを使用して、SnapMirror関係を作成および初期化します。

1. セカンダリ（SnapMirrorデスティネーション）として機能するStorageClassを作成して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を、MirrorRelationshipで定義されているリモートSVMとピア関係にあるSVMに配置します。

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端を表すCRDです。宛先TMRには、目的の状態をTridentに通知する状態があります。宛先TMRの状態は次のとおりです。

- 確立済み：ローカルPVCはミラー関係のデスティネーションボリュームであり、これは新しい関係です。
- 昇格：ローカルPVCはReadWriteでマウント可能であり、ミラー関係は現在有効ではありません。
- * reestablished *：ローカルPVCはミラー関係のデスティネーションボリュームであり、以前はそのミラー関係に含まれていました。
  - デスティネーションボリュームはデスティネーションボリュームの内容を上書きするため、ソースボリュームとの関係が確立されたことがある場合は、reestablished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の関係になかった場合、再確立状態は失敗します。

計画外フェールオーバー時にセカンダリ**PVC**を昇格する

セカンダリKubernetesクラスタで次の手順を実行します。

- TridentMirrorRelationshipの_spec.state_フィールド をに更新します `promoted`。

計画的フェイルオーバー中にセカンダリ**PVC**を昇格

計画的フェイルオーバー（移行）中に、次の手順を実行してセカンダリPVCをプロモートします。

手順

1. プライマリKubernetesクラスタでPVCのSnapshotを作成し、Snapshotが作成されるまで待ちます。
2. プライマリKubernetesクラスター上で、内部の詳細を取得するためにTridentTridentSnapshotInfoCRを作成します。

   例

   ```
   kind: TridentSnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshotName csi-nas-snapshot
   ```

   a. 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、*spec.volumeMappings[].promotedSnapshotHandle* をスナップショットの internalName に更新します。
3. セカンダリKubernetesクラスタで、TridentMirrorRelationshipのステータス（status.stateフィールド）がPromotedになっていることを確認します。

フェイルオーバー後にミラー関係をリストアする

ミラー関係をリストアする前に、新しいプライマリとして作成する側を選択します。

手順

1. セカンダリKubernetesクラスタで、TridentMirrorRelationshipの_spec.remoteVolumeHandle_field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クラスターで、TridentMirrorRelationshipの_spec.state_フィールドを`reestablished`に更新します。

その他の処理

Tridentでは、プライマリボリュームとセカンダリボリュームで次の処理がサポートされます。

新しいセカンダリ**PVC**へのプライマリ**PVC**の複製

プライマリPVCとセカンダリPVCがすでに存在していることを確認します。

手順

1. PersistentVolumeClaim CRDとTridentMirrorRelationship CRDを、確立されたセカンダリ（デスティネーション）クラスタから削除します。
2. プライマリ（ソース）クラスタからTridentMirrorRelationship CRDを削除します。
3. 確立する新しいセカンダリ（デスティネーション）PVC用に、プライマリ（ソース）クラスタに新しいTridentMirrorRelationship CRDを作成します。

ミラー、プライマリ、またはセカンダリ**PVC**のサイズ変更

PVCは通常どおりサイズ変更できます。データ量が現在のサイズを超えると、ONTAPは自動的に宛先フレフxolを拡張します。

PVCからのレプリケーションの削除

レプリケーションを削除するには、現在のセカンダリボリュームで次のいずれかの操作を実行します。

- セカンダリPVCのMirrorRelationshipを削除します。これにより、レプリケーション関係が解除されます。
- または、spec.stateフィールドを_promoted_に更新します。

（以前にミラーリングされていた）**PVC**の削除

Tridentは、レプリケートされたPVCがないかどうかを確認し、レプリケーション関係を解放してからボリュームの削除を試行します。

TMRの削除

ミラー関係の片側のTMRを削除すると、Tridentが削除を完了する前に、残りのTMRが_PROMOTED_STATEに移行します。削除対象として選択されたTMRがすでに_promoted_state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_にプロモートします。この削除により、ONTAP内のローカルボリュームのSnapMirrorメタデータが解放されます。このボリュームを今後ミラー関係で使用する場合は、新しいミラー関係を作成するときに、レプリケーション状態が_established_volumeである新しい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

ミラー関係は、確立後にいつでも更新できます。フィールドまたはフィールドを使用して関係を更新できます

`state: promoted` `state: reestablished` 。デスティネーションボリュームを通常のReadWriteボリュームに昇格する場合は、_promotedSnapshotHandle_を使用して、現在のボリュームのリストア先となる特定のSnapshotを指定できます。

ONTAPがオフラインの場合にミラー関係を更新

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更新を実行できます。次のTridentActionMirrorUpdateの形式例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_のいずれかの値を指定できます。

CSI トポロジを使用します

Tridentでは、を使用して、Kubernetesクラスタ内のノードを選択的に作成して接続できます "CSI トポロジ機能"。

概要

CSI トポロジ機能を使用すると、領域およびアベイラビリティゾーンに基づいて、ボリュームへのアクセスをノードのサブセットに制限できます。現在、クラウドプロバイダは、 Kubernetes 管理者がゾーンベースのノードを生成できるようになっています。ノードは、リージョンによって異なるアベイラビリティゾーンに配置することも、リージョンによって配置することもできます。マルチゾーンアーキテクチャでワークロード用のボリュームのプロビジョニングを容易にするために、TridentではCSIトポロジを使用しています。

ヒント	CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

- `VolumeBindingMode`をに設定する `Immediate`と、Tridentはトポロジを認識せずにボリュームを作成します。ボリュームバインディングと動的プロビジョニングは、 PVC が作成されるときに処理されます。これはデフォルト `VolumeBindingMode`であり、トポロジの制約を適用しないクラスタに適しています。永続ボリュームは、要求元ポッドのスケジュール要件に依存することなく作成されます。
- VolumeBindingMode を「 WaitForFirstConsumer 」に設定すると、 PVC の永続ボリュームの作成とバインドは、 PVC を使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

メモ	「 WaitForFirstConsumer 」バインディングモードでは、トポロジラベルは必要ありません。

> これは CSI トポロジ機能とは無関係に使用できます。

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- を実行するKubernetesクラスタ "サポートされるKubernetesバージョン"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- クラスタ内のノードには、トポロジ認識を導入するラベルが必要です `topology.kubernetes.io/region`および `topology.kubernetes.io/zone`）。Tridentでのトポロジ対応プロビジョニングには、 `topology.kubernetes.io/region`ラベルが必須です。これらのラベルは、Tridentをインストールする前にノードに存在している必要があります。インストール後にトポロジラベルを追加または変更した場合、Tridentが変更を認識するには、Tridentノードポッドを再起動するだけで十分です。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
```

```
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

手順 1 ：トポロジ対応バックエンドを作成する

Tridentストレージバックエンドは、アベイラビリティゾーンに基づいて選択的にボリュームをプロビジョニングするように設計できます。各バックエンドは、サポートされているゾーンとリージョンのリストを表すオプションのブロックを運ぶことができます `supportedTopologies` 。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
```

```
      }
    ]
}
```

メモ	`supportedTopologies`は、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、 StorageClass で指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含むストレージクラスの場合、Tridentはバックエンドにボリュームを作成します。

また ' ストレージ・プールごとに 'upportedTopologies を定義することもできます次の例を参照してください。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

この例では、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

クラスタ内のノードに提供されるトポロジラベルに基づいて、トポロジ情報を含めるように StorageClasses を定義できます。これにより、作成された PVC 要求の候補となるストレージプール、および Trident によってプロビジョニングされたボリュームを使用できるノードのサブセットが決まります。

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

前述のStorageClass定義では、 `volumeBindingMode`がに設定されて `WaitForFirstConsumer`います。この `StorageClass` で要求された `PVC` は、ポッドで参照されるまで処理されません。およびに、 `allowedTopologies`使用するゾーンとリージョンを示します。StorageClassは `netapp-san-us-east1`、上記で定義したバックエンドにPVCを作成し`san-backend-us-east1`ます。

ステップ **3** ： **PVC** を作成して使用する

StorageClass を作成してバックエンドにマッピングすると、 PVC を作成できるようになりました。

以下の例「 PEC 」を参照してください。

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: 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、次のような結果になります。

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

Trident でボリュームを作成して PVC にバインドするには、ポッド内の PVC を使用します。次の例を参照してください。

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
```

```
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

この podSpec は 'us-east1` 領域に存在するノード上のポッドをスケジュールするよう Kubernetes に指示し 'us-east1-a' または `us-east1-b' ゾーン内に存在する任意のノードから選択します

次の出力を参照してください。

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

バックエンドを更新して追加 **supportedTopologies**

既存のバックエンドは 'tridentctl backend update を使用して 'upportedTopologies のリストを含むように更新できますこれは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用され

ます。

詳細については、こちらをご覧ください

- "コンテナのリソースを管理"
- "ノードセレクタ"
- "アフィニティと非アフィニティ"
- "塗料および耐性"

Trident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Snapshotを使用すると、ボリュームのポイントインタイムコピーを作成できます。Tridentを使用して作成したボリュームのSnapshotの作成、Tridentの外部で作成したSnapshotのインポート、既存のSnapshotからの新しいボリュームの作成、Snapshotからのボリュームデータのリカバリを実行できます。

概要

ボリューム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`、 `azure-netapp-files`、 そして `google-cloud-netapp-volumes`ドライバー。

作業を開始する前に

スナップショットを操作するには、外部スナップショットコントローラとカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

Kubernetesディストリビューションにスナップショットコントローラ とCRDが含まれていない場合は、を参照してください ボリュームSnapshotコントローラの導入。

メモ	GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKEでは、内蔵の非表示のスナップショットコントローラを使用します。

ボリューム **Snapshot** を作成します

手順

1. を作成します `VolumeSnapshotClass`。詳細については、を参照してください "ボリュームSnapshotクラス"。
   - は `driver`Trident CSIドライバを示しています。
   - `deletionPolicy` は、です `Delete` または `Retain`。に設定すると `Retain`を使用すると、ストレージクラスタの基盤となる物理`Snapshot`が、の場合でも保持されます `VolumeSnapshot` オブジェクトが削除された。

例

```
cat snap-sc.yaml
```

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

2. 既存のPVCのスナップショットを作成します。

例

- 次に、既存のPVCのスナップショットを作成する例を示します。

```
cat snap.yaml
```

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

- 次の例は、という名前のPVCのボリュームSnapshotオブジェクトを作成します。 `pvc1` Snapshotの名前はに設定されます `pvc1-snap`。ボリュームSnapshotはPVCに似ており、に関連付けられています `VolumeSnapshotContent` 実際のスナップショットを表すオブジェクト。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 次の情報を確認できます。 `VolumeSnapshotContent` のオブジェクト `pvc1-snap` ボリュームSnapshot。ボリュームSnapshotの詳細を定義します。。 `Snapshot Content Name` このSnapshotを提供するVolumeSnapshotContentオブジェクトを特定します。。 `Ready To Use` パラメータは、スナップショットを使用して新しいPVCを作成できることを示します。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

ボリュームSnapshotからPVCを作成

を使用できます `dataSource` という名前のVolumeSnapshotを使用してPVCを作成するには `<pvc-name>` データのソースとして。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> 警告 PVCはソースボリュームと同じバックエンドに作成されます。を参照してください "KB：Trident PVCスナップショットからPVCを作成することは代替バックエンドではできない"。

次に、を使用してPVCを作成する例を示します。 `pvc1-snap` をデータソースとして使用します。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
```

```
    apiGroup: snapshot.storage.k8s.io
```

ボリューム**Snapshot**のインポート

Tridentでは、クラスタ管理者がを"Kubernetesの事前プロビジョニングされたSnapshotプロセス"使用して、オブジェクトを作成したり、Tridentの外部で作成されたSnapshotをインポートしたりできます `VolumeSnapshotContent`。

作業を開始する前に

TridentでSnapshotの親ボリュームが作成またはインポートされている必要があります。

手順

1. *クラスタ管理者：*バックエンドSnapshotを参照するオブジェクトを作成します `VolumeSnapshotContent`。これにより、TridentでSnapshotワークフローが開始されます。
   - バックエンドスナップショットの名前を `annotations` として `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - で指定します `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` `snapshotHandle`。この情報は、呼び出しで外部スナップショットによってTridentに提供される唯一の情報です `ListSnapshots`。

メモ	。 `<volumeSnapshotContentName>` CRの命名規則のため、バックエンドスナップショット名が常に一致するとは限りません。

     例

     次の例では、 `VolumeSnapshotContent` バックエンドスナップショットを参照するオブジェクト `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. クラスタ管理者： `VolumeSnapshot` を参照するCR `VolumeSnapshotContent` オブジェクト。これにより、 `VolumeSnapshot` 指定された名前空間内。

   例

   次の例では、 `VolumeSnapshot` CR名 `import-snap` を参照しています。 `VolumeSnapshotContent` 名前付き `import-snap-content`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. *内部処理（アクション不要）：*外部スナップショットは、新しく作成されたを認識して `VolumeSnapshotContent`呼び出しを実行します `ListSnapshots`。Tridentによってが作成され`TridentSnapshot`ます。
   - 外部スナップショットは、 `VolumeSnapshotContent` 終了： `readyToUse` および `VolumeSnapshot` 終了： `true`。
   - Tridentのリターン `readyToUse=true`。
4. 任意のユーザー： `PersistentVolumeClaim` 新しい `VolumeSnapshot`を参照してください `spec.dataSource` （または `spec.dataSourceRef`）nameは `VolumeSnapshot` 名前。

   例

   次に、を参照するPVCを作成する例を示します。 `VolumeSnapshot` 名前付き `import-snap`。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
```

```
    apiGroup: snapshot.storage.k8s.io
```

Snapshotを使用してボリュームデータをリカバリします

snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` スナップショットからデータを直接リカバリするディレクトリ。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

メモ	Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

Snapshotからのインプレースボリュームのリストア

Tridentでは、（TASR）CRを使用してSnapshotからボリュームをインプレースで迅速にリストアできます `TridentActionSnapshotRestore`。このCRはKubernetesの必須アクションとして機能し、処理の完了後も維持されません。

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` 、 `google-cloud-netapp-volumes` 、 そして `solidfire-san`ドライバー。

作業を開始する前に

バインドされたPVCと使用可能なボリュームSnapshot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Snapshotを使用する準備が完了してい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およびボリュームスナップショット用のCRを作成し `pvc1` `pvc1-snapshot`ます。

メモ	TASR CRは、PVCおよび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リストアするにはCRを適用します。この例では、Snapshotからリストアし `pvc1` ます。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

結果

Tridentはスナップショットからデータをリストアします。Snapshotリストアのステータスを確認できます。

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
```

```
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

メモ

- ほとんどの場合、障害が発生したときにTridentで処理が自動的に再試行されることはありません。この操作を再度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ユーザは、アプリケーションネームスペースにTASR CRを作成するために、管理者から権限を付与されなければならない場合があります。

Snapshotが関連付けられている**PV**を削除する

Snapshotが関連付けられている永続ボリュームを削除すると、対応するTridentボリュームが「削除中」に更新されます。ボリュームSnapshotを削除してTridentボリュームを削除します。

ボリューム**Snapshot**コントローラの導入

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のように導入できます。

手順

1. ボリュームのSnapshot作成

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. スナップショットコントローラを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
```

```
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

メモ	必要に応じて、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

関連リンク

- "ボリューム Snapshot"
- "ボリュームSnapshotクラス"

Tridentボリュームグループのスナップショットの操作

Kubernetes の永続ボリューム（PV）のボリュームグループスナップショット NetApp Trident は、複数のボリュームのスナップショット（ボリュームスナップショットのグループ）を作成する機能を提供します。このボリュームグループスナップショットは、同じ時点で作成された複数のボリュームのコピーを表します。

メモ	VolumeGroupSnapshot は、ベータ API を備えた Kubernetes のベータ機能です。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、次のストレージ ドライバーでサポートされます。

- `ontap-san`ドライバー - iSCSI および FC プロトコルのみで、NVMe/TCP プロトコルには使用できません。
- `ontap-san-economy` - iSCSI プロトコルのみ。
- 「 ONTAP - NAS 」

メモ	ボリューム グループ スナップショットは、 NetApp ASA r2 または AFX ストレージ システムではサポートされていません。

作業を開始する前に

- Kubernetes のバージョンが K8s 1.32 以上であることを確認してください。
- スナップショットを操作するには、外部スナップショットコントローラとカスタムリソース定義（CRD）が必要です。Kubernetesオーケストレーションツール（例：Kubeadm、GKE、OpenShift）の役割を担っています。

  Kubernetesディストリビューションに外部スナップショットコントローラとCRDが含まれていない場合は、 ボリュームSnapshotコントローラの導入 。

メモ	GKE 環境でオンデマンド ボリューム グループ スナップショットを作成する場合は、スナップショット コントローラを作成しないでください。GKEでは、内蔵の非表示のスナップショットコントローラを使用します。

- 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、を参照してください "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。

。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ユニットとして復元することはできません。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

メモ	Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

Snapshotからのインプレースボリュームのリストア

Tridentでは、（TASR）CRを使用してSnapshotからボリュームをインプレースで迅速にリストアできます `TridentActionSnapshotRestore`。このCRはKubernetesの必須アクションとして機能し、処理の完了後も維持されません。

詳細については、を参照してください "Snapshotからのインプレースボリュームのリストア"。

関連付けられた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、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**Snapshot**コントローラの導入

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のように導入できます。

手順

1. ボリュームのSnapshot作成

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
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を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

メモ	必要に応じて、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

関連リンク

- "ボリュームグループスナップショットクラス"
- "ボリューム Snapshot"

Tridentの管理と監視

Tridentのアップグレード

Tridentのアップグレード

24.02リリース以降、Tridentはリリースサイクルを4カ月に短縮し、毎年3つのメジャーリリースを提供しています。新しいリリースは、以前のリリースに基づいて構築され、新機能、パフォーマンスの強化、バグの修正、および改善が提供されます。Tridentの新機能を利用するには、少なくとも年に1回アップグレードすることをお勧めします。

アップグレード前の考慮事項

Tridentの最新リリースにアップグレードする場合は、次の点を考慮してください。

- 特定のKubernetesクラスタ内のすべてのネームスペースには、Tridentインスタンスを1つだけインストールする必要があります。
- Trident 23.07以降では、v1ボリュームスナップショットが必要です。alphaまたはbetaスナップショットはサポートされません。
- をアップグレードする場合は、を `StorageClasses`Tridentで使用するために指定することが重要です `parameter.fsType`。既存のボリュームを停止することなく、削除や再作成を実行できます `StorageClasses`。
  - これは、強制の 要件 です "セキュリティコンテキスト" SAN ボリュームの場合。
  - sample inputディレクトリには、https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templなどの例が含まれています[`storage-class-basic.yaml.templ`）とリンク：https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-bronze-default.yaml[`storage-class-bronze-default.yaml`^]をクリックします。
  - 詳細については、を参照してください "既知の問題"。

ステップ**1**：バージョンを選択します

Tridentバージョンは、日付ベースの命名規則に従い `YY.MM`ます。「YY」は年の最後の2桁、「mm」は月です。ドットリリースは規則に従い `YY.MM.X`ます。「X」はパッチレベルです。アップグレード前のバージョンに基づいて、アップグレード後のバージョンを選択します。

- インストールされているバージョンの4リリースウィンドウ内にある任意のターゲットリリースに直接アップグレードできます。たとえば、24.06（または任意の24.06 DOTリリース）から25.06に直接アップグレードできます。
- 4つのリリースウィンドウ以外のリリースからアップグレードする場合は、複数の手順でアップグレードを実行します。4リリースのウィンドウに適合する最新リリースにアップグレードするには、アップグレード元ののののアップグレード手順を使用し "以前のバージョン" ます。たとえば、23.07を実行していて、25.06にアップグレードする場合は、次の手順を実行します。
  a. 23.07から24.06への最初のアップグレード。
  b. その後、24.06から25.06にアップグレードします。

メモ	OpenShift Container PlatformでTridentオペレータを使用してアップグレードする場合は、Trident 21.01.1以降にアップグレードする必要があります。21.01.0 でリリースされた Trident オペレータには、 21.01.1 で修正された既知の問題が含まれています。詳細については、 "GitHub の問題の詳細"。

ステップ**2:**元のインストール方法を決定します

Tridentを最初にインストールしたバージョンを確認するには、次の手順を実行します。

1. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してTridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Tridentは手動またはHelmを使用してTridentオペレータを使用してインストールされています。
2. オペレータポッドがある場合は、を使用して、 `kubectl describe torc`TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合、TridentはHelmを使用してインストールされています。
   - Helmラベルがない場合、TridentはTridentオペレータを使用して手動でインストールされています。

ステップ**3**：アップグレード方法を選択します

通常は、最初のインストールと同じ方法でアップグレードする必要がありますが、"インストール方法を切り替えます"。Trident をアップグレードするには 2 つのオプションがあります。

- "Tridentオペレータを使用してアップグレード"

ヒント	レビューすることをお勧めします "オペレータのアップグレードワークフローについて理解する" オペレータでアップグレードする前に。

*

オペレータにアップグレードしてください

Tridentオペレータのアップグレードワークフローを理解する

Tridentオペレータを使用してTridentをアップグレードする前に、アップグレード中に発生するバックグラウンドプロセスについて理解しておく必要があります。これには、Tridentコントローラ、コントローラポッドとノードポッド、およびローリング更新を可能にするノードデーモンセットに対する変更が含まれます。

Tridentオペレータのアップグレード処理

Trident のインストールとアップグレードの多くの"Tridentオペレータを使用するメリット"の 1 つは、既存のマウントされたボリュームを中断することなく、Trident と Kubernetes オブジェクトを自動的に処理することです。このようにして、Trident はダウンタイムなしでアップグレードをサポートできます。または"ローリング更新"。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 Controller環境とノードデーモンセットを削除して再作成します。

- Tridentコントローラポッドと Tridentノードポッドを新しいバージョンに置き換えます。
  - 更新されていないノードは、残りのノードの更新を妨げません。
  - ボリュームをマウントできるのは、Trident Node Podを実行しているノードだけです。

ヒント	KubernetesクラスタのTridentアーキテクチャの詳細については、を参照してください"Tridentのアーキテクチャ"。

オペレータのアップグレードワークフロー

Tridentオペレータを使用してアップグレードを開始すると、次の処理が実行されます。

1. Trident演算子*：
   a. 現在インストールされているTridentのバージョン（version_n_）を検出します。
   b. CRD、RBAC、Trident SVCなど、すべてのKubernetesオブジェクトを更新
   c. version_n_用のTrident Controller環境を削除します。
   d. version_n+1_用のTrident Controller環境を作成します。
2. * Kubernetes *は、_n+1_用にTridentコントローラポッドを作成します。
3. Trident演算子*：
   a. _n_のTridentノードデーモンセットを削除します。オペレータは、Node Podが終了するのを待たない。
   b. _n+1_のTridentノードデーモンセットを作成します。
4. * Kubernetes * Trident Node Pod_n_を実行していないノードにTridentノードポッドを作成します。これにより、1つのノードに複数のTrident Node Pod（バージョンに関係なく）が存在することがなくなります。

Trident operatorまたは**Helm**を使用した**Trident**インストールのアップグレード

Trident オペレーターを使用して、手動または Helm を使用して Trident をアップグレードできます。Trident オペレーターのインストールから別の Trident オペレーターのインストールにアップグレードすることも、 `tridentctl`のインストールから Trident オペレーターバージョンにアップグレードすることもできます。Trident オペレーターのインストールをアップグレードする前に"アップグレード方法を選択します"を確認してください。

手動インストールのアップグレード

クラスター 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。

メモ	名前空間を対象とした演算子（バージョン20.07～20.10）を使用してインストールされたTridentからアップグレードするには、Tridentののアップグレード手順を使用します"インストールされているバージョン"。

このタスクについて

Tridentにはバンドルファイルが用意されています。このファイルを使用して、オペレータをインストールしたり、Kubernetesバージョンに対応する関連オブジェクトを作成したりできます。

- クラスタでKubernetes 1.25以降を実行している場合は、を使用します "bundle_post_1_25.yaml"。

作業を開始する前に

を実行しているKubernetesクラスタを使用していることを確認します "サポートされるKubernetesバージョン"。

手順

1. Tridentのバージョンを確認します。

```
./tridentctl -n trident version
```

2. 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. を使用して初期インストールをカスタマイズした場合 `TridentOrchestrator` 属性を編集できます `TridentOrchestrator` インストールパラメータを変更するオブジェクト。これには、ミラーリングされたTridentおよびCSIイメージレジストリをオフラインモードに指定したり、デバッグログを有効にしたり、イメージプルシークレットを指定したりするための変更が含まれます。
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。

警告	TridentがインストールされているKubernetesクラスタを1.24から1.25以降にアップグレードする場合は `helm upgrade`、クラスタをアップグレードする前に、values.yamlをに `true`設定するかコマンドに追加する `--set excludePodSecurityPolicy=true`ように更新する必要があります。 `excludePodSecurityPolicy`

Trident Helmをアップグレードせずにkubernetesクラスタを1.24から1.25にアップグレードした場合、Helmのアップグレードは失敗します。Helmのアップグレードを実行するには、次の手順を前提条件として実行します。

1. からhelm-mapkubeapisプラグインをインストールします https://github.com/helm/helm-mapkubeapis。
2. Tridentがインストールされているネームスペースで、Tridentリリースのドライランを実行します。リソースが一覧表示され、クリーンアップされます。

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. クリーンアップを実行するには、helmを使用してフルランを実行します。

```
helm mapkubeapis trident  --namespace trident
```

手順

1. "Helmを使用してTridentをインストール"の場合は、 `helm upgrade trident netapp-trident/trident-operator --version 100.2610.0`を使用してワンステップでアップグレードできます。Helmリポジトリを追加していない場合、またはアップグレードに使用できない場合：
   a. から最新のTridentリリースをダウンロードし"GitHubの_Assets_sectionを参照してください"ます。
   b. `helm upgrade`コマンドを使用します。 `trident-operator-26.10.0.tgz`にはアップグレード先のバージョンを指定します。

```
helm upgrade <name> trident-operator-26.10.0.tgz
```

メモ	初期インストール時にカスタムオプションを設定した場合（TridentイメージとCSIイメージのプライベートなミラーレジストリの指定など）は、 `helm upgrade` コマンド `--set` これらのオプションがupgradeコマンドに含まれるようにするため、それらのオプションの値をdefaultにリセットします。

2. を実行します `helm list` グラフとアプリのバージョンが両方ともアップグレードされていることを確認します。を実行します `tridentctl logs` デバッグメッセージを確認します。

からのアップグレード `tridentctl` Tridentオペレータへのインストール

からTridentの最新リリースにアップグレードできます `tridentctl` インストール：既存のバックエンドとPVCは自動的に使用可能になります。

メモ	インストール方法を切り替える前に、"インストール方法を切り替える"を確認してください。

手順

1. 最新のTridentリリースをダウンロードします。

```
# Download the release required [26.10.0]
```

```
mkdir 26.10.0
cd 26.10.0
wget
https://github.com/NetApp/trident/releases/download/v26.10.0/trident-
installer-26.10.0.tar.gz
tar -xf trident-installer-26.10.0.tar.gz
cd trident-installer
```

2. マニフェストから「 tridentオーケストラ 」 CRD を作成します。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. クラスタを対象としたオペレータを同じネームスペースに導入します。

```
kubectl create -f deploy/<bundle-name.yaml>

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

4. TridentをインストールするためのCRを作成し `TridentOrchestrator`ます。

```
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
```

```
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m
trident-csi-xrst8                   2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0          5m41s
```

5. Tridentが目的のバージョンにアップグレードされたことを確認

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v26.10.0
```

tridentctlを使用したTridentのアップグレード

を使用して、既存のTridentインストールを簡単にアップグレードできます `tridentctl`。

このタスクについて

Tridentのアンインストールと再インストールは、アップグレードとして機能します。Tridentをアンインストールしても、Trident環境で使用されている永続的ボリューム要求（PVC）と永続的ボリューム（PV）は削除されません。すでにプロビジョニングされているPVCは、Tridentがオフラインの間も使用できます。また、その間に作成されたPVCがオンラインに戻ったあとも、Tridentはボリュームをプロビジョニングします。

作業を開始する前に

"アップグレード方法を選択します"を使用してアップグレードする前に `tridentctl`を確認してください。

手順

1. のuninstallコマンドを実行し `tridentctl`て、CRDと関連オブジェクトを除くTridentに関連付けられているすべてのリソースを削除します。

```
./tridentctl uninstall -n <namespace>
```

2. Trident を再インストールします。"tridentctlを使用したTridentのインストール"を参照してください。

重要	アップグレードプロセスを中断しないでください。インストーラが完了するまで実行されることを確認します。

tridentctlを使用したTridentの管理

には、 "Trident インストーラバンドル"Tridentに簡単にアクセスできるコマンドラインユーティリティが含まれてい `tridentctl`ます。十分なPrivilegesを持つKubernetesユーザ

は、Tridentをインストールしたり、Tridentポッドを含むネームスペースを管理したりできます。

コマンドとグローバルフラグ

走れ `tridentctl help` 使用可能なコマンドのリストを取得するには `tridentctl` または、 `--help` 特定のコマンドのオプションとフラグのリストを取得するには、任意のコマンドにフラグを付けます。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl`ユーティリティは、次のコマンドとグローバルフラグをサポートしています。

▼ コマンド

create
Tridentにリソースを追加します。

delete
Tridentから1つ以上のリソースを削除します。

get
Tridentから1つ以上のリソースを取得します。

help
任意のコマンドに関するヘルプ。

images
Tridentが必要とするコンテナイメージの表を印刷します。

import
既存のリソースをTridentにインポートします。

install
Trident をインストール

logs
Tridentからログを印刷します。

send
Tridentからリソースを送信します。

uninstall
Tridentをアンインストールします。

update
Tridentでリソースを変更します。

update backend state
バックエンド処理を一時的に中断します。

upgrade

Tridentでリソースをアップグレードします。

「バージョン」

Tridentのバージョンを印刷します。

▼ グローバルフラグ

-d、 --debug

デバッグ出力。

-h、 --help

ヘルプ `tridentctl`。

-k、 --kubeconfig string

を指定します。 `KUBECONFIG` コマンドをローカルまたはKubernetesクラスタ間で実行するパス。

メモ	または、 `KUBECONFIG` 特定のKubernetesクラスタと問題をポイントする変数 `tridentctl` そのクラスタにコマンドを送信します。

-n、 --namespace string

Trident環境のネームスペース。

-o、 --output string

出力形式。JSON の 1 つ | yaml | name | wide | ps （デフォルト）。

-s、 --server string

Trident RESTインターフェイスのアドレス/ポート。

警告	Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

コマンドオプションとフラグ

作成

コマンドを使用し `create`て、Tridentにリソースを追加します。

```
tridentctl create [option]
```

オプション（ **Options** ）

`backend`：Tridentにバックエンドを追加します。

削除

コマンドを使用し `delete`て、Tridentから1つ以上のリソースを削除します。

```
tridentctl delete [option]
```

オプション（ **Options** ）

`backend`：Tridentから1つ以上のストレージバックエンドを削除します。
`snapshot`：Tridentから1つ以上のボリュームSnapshotを削除します。
`storageclass`：Tridentから1つ以上のストレージクラスを削除します。
`volume`：Tridentから1つ以上のストレージボリュームを削除します。

取得

コマンドを使用し `get`て、Tridentから1つ以上のリソースを取得します。

```
tridentctl get [option]
```

オプション（ **Options** ）

`backend`：Tridentから1つ以上のストレージバックエンドを取得します。
`snapshot`：Tridentから1つ以上のスナップショットを取得します。
`storageclass`：Tridentから1つ以上のストレージクラスを取得します。
`volume`：Tridentから1つ以上のボリュームを取得します。

フラグ

`-h、 --help`：ボリュームのヘルプ。
`--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
`--subordinateOf string`:クエリをボリュームの下位に制限します。

イメージ

フラグを使用して `images`、Tridentが必要とするコンテナイメージのテーブルを印刷します。

```
tridentctl images [flags]
```

フラグ

`-h、 --help`:画像のヘルプ。
`-v、 --k8s-version string`: Kubernetesクラスタのセマンティックバージョン。

ボリュームをインポートします

コマンドを使用し `import volume`て、既存のボリュームをTridentにインポートします。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

エイリアス

`volume、 v`

フラグ

`-f、 --filename string`: YAMLまたはJSON PVCファイルへのパス。
`-h、 --help`：ボリュームのヘルプ。
`--no-manage`：PV/PVCのみを作成します。ボリュームのライフサイクル管理を想定しないでください。

をインストールします

フラグを使用し `install`てTridentをインストールします。

```
tridentctl install [flags]
```

フラグ

`--autosupport-image string`: Autosupport 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 (デフォルトは 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

フラグを使用して `logs`Tridentからログを出力します。

```
tridentctl logs [flags]
```

フラグ

`-a、 --archive`：特に指定がないかぎり、すべてのログを含むサポートアーカイブを作成します。
`-h、 --help`：ログのヘルプ。
`-l、 --log string`：表示するTridentログ。Trident | auto | Trident - operator | allのいずれか（デフォルトは「auto」）。
`--node string`：ノードポッドログの収集元となるKubernetesノード名。
`-p,: --previous`以前のコンテナインスタンスが存在する場合は、そのインスタンスのログを取得します。
`--sidecars`:サイドカーコンテナのログを取得します。

送信

Tridentからリソースを送信するには、コマンドを使用し `send`ます。

```
tridentctl send [option]
```

オプション（ **Options** ）

`autosupport`：ネットアップにAutoSupport アーカイブを送信します。

をアンインストールします

フラグを使用して `uninstall`Tridentをアンインストールします。

```
tridentctl uninstall [flags]
```

フラグ

`-h, --help`:アンインストールのヘルプ。
`--silent`:アンインストール中のほとんどの出力を無効にします。

更新

Tridentのリソースを変更するには、コマンドを使用し `update`ます。

```
tridentctl update [option]
```

オプション（ **Options** ）

`backend`: Tridentのバックエンドを更新します。

バックエンドの状態を更新

を使用します `update backend state` バックエンド処理を一時停止または再開するコマンド。

```
tridentctl update backend state <backend-name> [flag]
```

考慮すべきポイント

- TridentBackendConfig（tbc）を使用してバックエンドを作成した場合、ファイルを使用してバックエンドを更新することはできません `backend.json` 。
- がtbcに設定されている場合 `userState` は、コマンドを使用して変更することはできません `tridentctl update backend state <backend-name> --user-state suspended/normal` 。
- tbcで設定した後にvia tridentctlを設定できるようにするには `userState`、 `userState`tbcからフィールドを削除する必要があります。これは、コマンドを使用して実行でき `kubectl edit tbc` ます。フィールドを削除したら `userState`、コマンドを使用してバックエンドのを変更 `userState`できます `tridentctl update backend state`。
- を使用して `tridentctl update backend state` を変更し `userState`ます。またはファイルを使用して更新することもでき `userState` `TridentBackendConfig` `backend.json` ます。これにより、バックエンドの完全な再初期化がトリガーされ、時間がかかる場合があります。

  フラグ

  `-h、 --help`:バックエンド状態のヘルプ。

`--user-state`:に設定 `suspended` バックエンド処理を一時停止します。をに設定します `normal` バックエンド処理を再開します。に設定すると `suspended`：

- `AddVolume` `Import Volume` 一時停止しています。
- `CloneVolume`、 `ResizeVolume`、 `PublishVolume`、 `UnPublishVolume`、、 `CreateSnapshot` `GetSnapshot` `RestoreSnapshot`、、 `DeleteSnapshot`、 `RemoveVolume`、、 `GetVolumeExternal` `ReconcileNodeAccess` 引き続き使用できます。

バックエンド構成ファイルまたはのフィールドを使用して、バックエンドの状態を更新することもできます `userState` `TridentBackendConfig` `backend.json`。詳細については、およびを参照して "バックエンドを管理するためのオプション" "kubectl を使用してバックエンド管理を実行します"ください。

- 例：*

JSON

ファイルを使用してを更新するには、次の手順を実行し `userState` `backend.json` ます。

1. ファイルを編集して `backend.json` 、値が「中断」に設定されたフィールドを含め `userState` ます。
2. 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"
}
```

YAML

tbcが適用されたら、コマンドを使用して編集できます `kubectl edit <tbc-name> -n <namespace>` 。次に、オプションを使用してバックエンド状態をsuspendに更新する例を示し `userState: suspended` ます。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
```

```
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
  userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

バージョン

使用 `version` のバージョンを印刷するためのフラグ `tridentctl` 実行中のTridentサービス

```
tridentctl version [flags]
```

フラグ

`--client`:クライアントバージョンのみ(サーバは不要)。
`-h, --help`:バージョンのヘルプ。

プラグインのサポート

Tridentctlはkubectlに似たプラグインをサポートしています。Tridentctlは、プラグインバイナリファイル名が"tridentctl -<plugin>"というスキームに沿っている場合にプラグインを検出し、そのバイナリがPATH環境変数のリストにあるフォルダにあることを示します。検出されたすべてのプラグインは、tridentctlヘルプのpluginセクションに表示されます。オプションで、環境変数TRIDENTCTL_PLUGIN_PATHにプラグインフォルダを指定して検索を制限することもできます(例: `TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`)。変数が使用されている場合、tridenctlは指定されたフォルダのみを検索します。

Tridentストレージ操作の監視

Tridentには、Tridentのパフォーマンスの監視に使用できるPrometheus指標エンドポイントのセットが用意されています。

概要

Tridentの指標を使用すると、次のことを実行できます。

- Tridentの健全性と設定を常に確認しておきます。処理が成功した方法と、想定どおりにバックエンドと通信できるかどうかを調べることができます。
- バックエンドの使用状況の情報を調べて、バックエンドでプロビジョニングされているボリュームの数や消費されているスペースなどを確認します。
- 利用可能なバックエンドにプロビジョニングされたボリュームの量のマッピングを維持します。
- パフォーマンスを追跡する。Tridentがバックエンドと通信して処理を実行するのにかかる時間を確認できます。

メモ	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。ポートでHTTPS経由でTridentメトリックを使用するように設定できます `8444`同じように。

必要なもの

- TridentがインストールされたKubernetesクラスタ。
- Prometheus インスタンス。これは a である場合もある "コンテナ化された Prometheus 環境" または、Prometheus をとして実行することもできます "ネイティブアプリケーション"。

手順 1 ： Prometheus ターゲットを定義する

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。見る"Prometheus Operator ドキュメント"。

手順 2 ： Prometheus ServiceMonitor を作成します

Trident のメトリックを使用するには、「 trident-csi 」サービスを監視し、「 m etrics 」ポートを監視する Prometheus ServiceMonitor を作成する必要があります。ServiceMonitor のサンプルは次のようになります。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - port: metrics
      interval: 15s
```

このServiceMonitor定義は、 `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。

Kubeletは、Tridentから直接利用できるメトリクスに加えて、独自のメトリクスエンドポイントを介して多くのメトリクスを公開して `kubelet_volume_*`います。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "こちらをご覧ください"。

HTTPS経由でTridentメトリクスを消費する

HTTPS (ポート 8444) 経由でTridentメトリックを使用するには、ServiceMonitor 定義を変更して TLS 構成を含める必要があります。また、 `trident-csi`秘密から `trident`Prometheus が実行されている名前空間への名前空間。次のコマンドを使用してこれを実行できます。

```
kubectl get secret trident-csi -n trident -o yaml | sed 's/namespace:
trident/namespace: monitoring/' | kubectl apply -f -
```

HTTPS メトリックのサンプル ServiceMonitor は次のようになります。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - interval: 15s
      path: /metrics
      port: https-metrics
      scheme: https
      tlsConfig:
        ca:
          secret:
            key: caCert
            name: trident-csi
        cert:
          secret:
            key: clientCert
            name: trident-csi
        keySecret:
          key: clientKey
          name: trident-csi
        serverName: trident-csi
```

Trident は、tridentctl、Helm chart、Operator など、すべてのインストール方法で HTTPS メトリックをサポートします。

- を使用している場合 `tridentctl install`コマンドを実行すると、 `--https-metrics` HTTPS メトリックを有効にするフラグ。
- Helmチャートを使用している場合は、 `httpsMetrics` HTTPS メトリックを有効にするパラメータ。
- YAMLファイルを使用している場合は、 `--https_metrics`旗に `trident-main`コンテナ内の `trident-deployment.yaml`ファイル。

ステップ 3 ： PromQL を使用して Trident 指標を照会する

PromQL は、時系列データまたは表データを返す式を作成するのに適しています。

次に、 PromQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Trident**からの**HTTP 2XX**応答の割合

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- ステータスコードによる**Trident**からの**REST**応答の割合

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Trident**によって実行された操作の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Tridentの使用状況情報の取得

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

メモ	これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Trident AutoSupportテレメトリの詳細

デフォルトでは、TridentはPrometheus指標と基本的なバックエンド情報を1日おきにNetAppに送信します。

- TridentからNetAppへのPrometheus指標と基本的なバックエンド情報の送信を停止するには、Tridentのインストール時にフラグを渡し `--silence-autosupport`ます。
- Tridentは、経由でコンテナログをNetAppサポートにオンデマンドで送信することもできます `tridentctl send autosupport`。ログをアップロードするには、Tridentをトリガーする必要があります。ログを送信する前に、NetAppを承認する必要がありま すhttps://www.netapp.com/company/legal/privacy-policy/["プライバシーポリシー"^]。
- 指定されていない場合、Tridentは過去24時間のログをフェッチします。
- フラグを使用してログの保持期間を指定できます `--since`。例： `tridentctl send autosupport --since=1h`。この情報は、Tridentと一緒にインストールされたコンテナを介して収集および送信され `trident-autosupport`ます。コンテナイメージはから入手できます "Trident AutoSupport の略"。
- Trident AutoSupportは、個人を特定できる情報（PII）または個人情報を収集または送信しません。付属の "EULA"は、Tridentコンテナイメージ自体には適用されません。NetAppのデータセキュリティと信頼への取り組みの詳細については、 "こちらをご覧ください"を参照してください。

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、ネットアップの AutoSupport エンドポイントに送信されます。プライベートレジストリを使用してコンテナイメージを格納している場合は '--image_registry' フラグを使用できます
- インストール YAML ファイルを生成してプロキシ URL を設定することもできます。これは 'tridentctl install --generate-custom-yaml を使用して YAML ファイルを作成し 'trident-deployment.yaml の trident-autosupport コンテナに --proxy-url 引数を追加することによって実行できます

Trident指標を無効にする

- メトリックがレポートされないようにするには '--generate-custom-yaml' フラグを使用してカスタム YAML を生成し ' これらを編集して 'trident-main' コンテナに対して --metrics フラグが呼び出されないようにします

Trident をアンインストールします

Tridentのアンインストールには、Tridentのインストールと同じ方法を使用する必要があります。

このタスクについて

- アップグレード、依存関係の問題、またはアップグレードの失敗または不完全な実行後に見つかったバグの修正が必要な場合は、Tridentをアンインストールし、該当する手順に従って以前のバージョンを再インストールする必要があります"バージョン"。これは、以前のバージョンに_downgrade_toを実行するための唯一の推奨方法です。
- アップグレードと再インストールを簡単に行うために、Tridentをアンインストールしても、Tridentによって作成されたCRDや関連オブジェクトは削除されません。Tridentとそのすべてのデータを完全に削除する必要がある場合は、を参照してください"TridentとCRDを完全に取り外します。"。

作業を開始する前に

Kubernetesクラスタの運用を停止する場合は、をアンインストールする前に、Tridentで作成されたボリュームを使用するすべてのアプリケーションを削除する必要があります。これにより、PVCが削除される前にKubernetesノードで非公開になります。

元のインストール方法を決定する

Tridentのアンインストールには、インストール時と同じ方法を使用する必要があります。アンインストールする前に、Tridentの最初のインストールに使用したバージョンを確認してください。

1. 使用 `kubectl get pods -n trident` ポッドを検査するために。
   - オペレータポッドがない場合は、を使用してTridentがインストールされています `tridentctl`。
   - オペレータポッドがある場合、Tridentは手動またはHelmを使用してTridentオペレータを使用してインストールされています。
2. オペレータポッドがある場合は、を使用して、 `kubectl describe tproc trident`TridentがHelmを使用してインストールされているかどうかを確認します。
   - Helmラベルがある場合、TridentはHelmを使用してインストールされています。
   - Helmラベルがない場合、TridentはTridentオペレータを使用して手動でインストールされています。

Tridentオペレータのインストールをアンインストールする

Tridentオペレータのインストールは手動でアンインストールすることも、Helmを使用してアンインストールすることもできます。

手動インストールのアンインストール

オペレータを使用してTridentをインストールした場合は、次のいずれかの方法でアンインストールできます。

1. 編集 **`TridentOrchestrator` CR**を実行し、アンインストールフラグを設定します：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"uninstall":true}}'
```

   をクリックします `uninstall` フラグはに設定されています `true`は、TridentオペレータがTridentをアンインストールしますが、TridentOrchestrator自体は削除されません。Trident を再度インストールする場合は、 TridentOrchestrator をクリーンアップして新しい Trident を作成する必要があります。

2. 削除 **`TridentOrchestrator`**：Tridentの展開に使用したCRを削除する `TridentOrchestrator` と、`Trident`をアンインストールするようにオペレータに指示します。オペレータがの削除を処理し `TridentOrchestrator`、インストール時に作成したTridentポッドを削除して、Tridentデプロイメントとデーモンセットの削除に進みます。

```
kubectl delete -f deploy/<bundle.yaml> -n <namespace>
```

Helmインストールのアンインストール

Helmを使用してTridentをインストールした場合は、を使用してアンインストールできます `helm uninstall`。

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

のアンインストール **`tridentctl`** インストール

Tridentに関連付けられているすべてのリソース（CRDおよび関連オブジェクトを除く）を削除するには、の

コマンドを `tridentctl`使用し `uninstall`ます。

```
./tridentctl uninstall -n <namespace>
```

Trident for Docker

導入の前提条件

Tridentを導入する前に、必要なプロトコルの前提条件をホストにインストールして設定する必要があります。

要件を確認します

- の導入がすべてを満たしていることを確認します "要件"。
- サポートされているバージョンの Docker がインストールされていることを確認します。Docker のバージョンが最新でない場合は、 "インストールまたは更新します"。

```
docker --version
```

- プロトコルの前提条件がホストにインストールおよび設定されていることを確認します。

NFSツール

オペレーティングシステム用のコマンドを使用して、NFSツールをインストールします。

RHEL 8以降

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

警告 | NFSツールをインストールしたあとにワーカーノードをリブートして、コンテナにボリュームを接続する際の障害を回避します。

iSCSIツール

使用しているオペレーティングシステム用のコマンドを使用して、iSCSIツールをインストールします。

RHEL 8以降

1. 次のシステムパッケージをインストールします。

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-mapper-multipath
```

2. iscsi-initiator-utils のバージョンが 6.2.0.874-2.el7 以降であることを確認します。

```
rpm -q iscsi-initiator-utils
```

3. スキャンを手動に設定：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
```

4. マルチパスを有効化：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

メモ	「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

5. 「 iscsid 」と「 multipathd 」が実行されていることを確認します。

```
sudo systemctl enable --now iscsid multipathd
```

6. 'iSCSI' を有効にして開始します

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. 次のシステムパッケージをインストールします。

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools
scsitools
```

2. open-iscsi バージョンが 2.0.874-5ubuntu2.10 以降（ bionic の場合）または 2.0.874-7.1ubuntu6.1 以降（ Focal の場合）であることを確認します。

```
dpkg -l open-iscsi
```

3. スキャンを手動に設定：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
```

```
/etc/iscsi/iscsid.conf
```

4. マルチパスを有効化：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

メモ	「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されていることを確認します。

```
sudo systemctl status multipath-tools
sudo systemctl enable --now open-iscsi.service
sudo systemctl status open-iscsi
```

NVMeツール

オペレーティングシステムに対応したコマンドを使用してNVMeツールをインストールします。

メモ	• NVMeにはRHEL 9以降が必要です。 • Kubernetesノードのカーネルバージョンが古すぎる場合や、使用しているカーネルバージョンに対応するNVMeパッケージがない場合は、ノードのカーネルバージョンをNVMeパッケージで更新しなければならないことがあります。

RHEL 9

```
sudo yum install nvme-cli
sudo yum install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

Ubuntu

```
sudo apt install nvme-cli
sudo apt -y install linux-modules-extra-$(uname -r)
```

```
sudo modprobe nvme-tcp
```

FCツール

オペレーティングシステム用のコマンドを使用して、FCツールをインストールします。

- FC PVS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、StorageClassでmountOptionを指定してインラインのスペース再生を実行します `discard`。を参照してください "Red Hat のドキュメント"。

RHEL 8以降

1. 次のシステムパッケージをインストールします。

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. マルチパスを有効化：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

メモ	「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

3. が実行中であることを確認し `multipathd`ます。

```
sudo systemctl enable --now multipathd
```

Ubuntu

1. 次のシステムパッケージをインストールします。

```
sudo apt-get install -y lsscsi sg3-utils multipath-tools scsitools
```

2. マルチパスを有効化：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
```

```
sudo service multipath-tools restart
```

メモ	「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。

3. が有効で実行中であることを確認し `multipath-tools`ます。

```
sudo systemctl status multipath-tools
```

Tridentの導入

Trident for Dockerは、NetAppストレージプラットフォームのDockerエコシステムと直接統合できます。ストレージプラットフォームから Docker ホストまで、ストレージリソースのプロビジョニングと管理をサポートします。また、将来プラットフォームを追加するためのフレームワークもサポートします。

同じホスト上でTridentの複数のインスタンスを同時に実行できます。これにより、複数のストレージシステムとストレージタイプへの同時接続が可能になり、 Docker ボリュームに使用するストレージをカスタマイズできます。

必要なもの

を参照してください"導入の前提条件"。前提条件を満たしていることを確認したら、Tridentを導入する準備が整います。

Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降）

メモ	作業を開始する前に 従来のデーモンメソッドでDocker 1.13/17.03より前のTridentを使用している場合は、マネージプラグインメソッドを使用する前に、Tridentプロセスを停止してDockerデーモンを再起動してください。

1. 実行中のインスタンスをすべて停止します。

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. Docker を再起動します。

```
systemctl restart docker
```

3. Docker Engine 17.03 （新しい 1.13 ）以降がインストールされていることを確認します。

```
docker --version
```

バージョンが最新でない場合は、 "インストール環境をインストールまたは更新します"。

手順

1. 構成ファイルを作成し、次のオプションを指定します。
   - config: デフォルトのファイル名は 'config.json ですが ' ファイル名に config オプションを指定することで ' 選択した任意の名前を使用できます構成ファイルは ' ホスト・システムの /etc/netappdvp ディレクトリに格納されている必要があります
   - 「 log-level 」：ログレベルを指定します（「 debug` 」、「 info 」、「 warn 」、「 error 」、「 fatal 」）。デフォルトは「 info 」です。
   - ebug: デバッグログを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。
     i. 構成ファイルの場所を作成します。

     ```
     sudo mkdir -p /etc/netappdvp
     ```

     ii. 構成ファイルを作成します

     ```
     cat << EOF > /etc/netappdvp/config.json
     ```

     ```
     {
       "version": 1,
       "storageDriverName": "ontap-nas",
       "managementLIF": "10.0.0.1",
       "dataLIF": "10.0.0.2",
       "svm": "svm_nfs",
       "username": "vsadmin",
       "password": "password",
       "aggregate": "aggr1"
     }
     EOF
     ```

2. 管理プラグインシステムを使用してTridentを起動します。を、使用しているプラグインのバージョン（xxx.xx.x）に置き換えます <version>。

   ```
   docker plugin install --grant-all-permissions --alias netapp
   netapp/trident-plugin:<version> config=myConfigFile.json
   ```

3. Tridentを使用して、構成済みシステムのストレージを消費します。

a. 「 firstVolume 」という名前のボリュームを作成します。

```
docker volume create -d netapp --name firstVolume
```

b. コンテナの開始時にデフォルトのボリュームを作成します。

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

c. ボリューム「 firstVolume 」を削除します。

```
docker volume rm firstVolume
```

従来の方法（バージョン **1.12** 以前）

作業を開始する前に

1. バージョン 1.10 以降の Docker がインストールされていることを確認します。

```
docker --version
```

使用しているバージョンが最新でない場合は、インストールを更新します。

```
curl -fsSL https://get.docker.com/ | sh
```

または "ご使用のディストリビューションの指示に従ってください"。

2. NFS または iSCSI がシステムに対して設定されていることを確認します。

手順

1. NetApp Docker Volume Plugin をインストールして設定します。

a. アプリケーションをダウンロードして開梱します。

```
wget
https://github.com/NetApp/trident/releases/download/10.0/trident-
installer-26.10.0.tar.gz
tar zxf trident-installer-26.10.0.tar.gz
```

b. ビンパス内の場所に移動します。

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/
```

```
sudo chown root:root /usr/local/bin/trident
sudo chmod 755 /usr/local/bin/trident
```

c. 構成ファイルの場所を作成します。

```
sudo mkdir -p /etc/netappdvp
```

d. 構成ファイルを作成します

```
cat << EOF > /etc/netappdvp/ontap-nas.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. バイナリを配置して構成ファイルを作成したら、目的の構成ファイルを使用してTridentデーモンを起動します。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

メモ　指定されていない場合、ボリュームドライバのデフォルト名は「NetApp」です。

デーモンを起動したら、Docker CLIインターフェイスを使用してボリュームを作成および管理できます。

3. ボリュームを作成します

```
docker volume create -d netapp --name trident_1
```

4. コンテナの開始時に Docker ボリュームをプロビジョニング：

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
```

```
alpine ash
```

5. Docker ボリュームを削除します。

```
docker volume rm trident_1
```

```
docker volume rm trident_2
```

システム起動時にTridentを起動する

システムベースのシステムのサンプルユニットファイルは、から入手できます `contrib/trident.service.example` Gitリポジトリで実行します。RHELでファイルを使用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

   複数のインスタンスを実行している場合は、ユニットファイルに一意の名前を使用してください。

```
cp contrib/trident.service.example
/usr/lib/systemd/system/trident.service
```

2. ファイルを編集し、概要（ 2 行目）を変更してドライバ名と構成ファイルのパス（ 9 行目）を環境に合わせます。
3. 変更を取り込むためにシステムをリロードします。

```
systemctl daemon-reload
```

4. サービスを有効にします。

   この名前は '/usr/lib/systemd/system' ディレクトリ内のファイルの名前によって異なります

```
systemctl enable trident
```

5. サービスを開始します。

```
systemctl start trident
```

6. ステータスを確認します。

```
systemctl status trident
```

メモ | ユニット・ファイルを変更するときは ' 変更を認識するために 'systemctl daemon-reload コマンドを実行します

Trident をアップグレードまたはアンインストールする

使用中のボリュームに影響を与えることなく、Trident for Dockerを安全にアップグレードできます。アップグレードプロセスでは、 `docker volume`プラグインへのコマンドが正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

Trident for Dockerをアップグレードするには、次の手順を実行します。

手順

1. 既存のボリュームを表示します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

2. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインをアップグレードします。

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```

メモ | Tridentの18.01リリースは、nDVPに代わるものです。イメージからイメージに `netapp/trident-plugin`直接アップグレードする必要があり `netapp/ndvp-plugin`ます。

4. プラグインを有効にします。

```
docker plugin enable netapp:latest
```

5. プラグインが有効になっていることを確認します。

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

重要

古いバージョンのTrident（20.10より前のバージョン）からTrident 20.10以降にアップグレードすると、エラーが発生することがあります。詳細については、を参照してください "既知の問題"。エラーが発生した場合は、まずプラグインを無効にしてからプラグインを削除し、追加のconfigパラメータを渡して必要なTridentバージョンをインストールする必要があります。
`docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

をアンインストールします

Trident for Dockerをアンインストールするには、次の手順を実行します。

手順

1. プラグインで作成されたボリュームをすべて削除します。
2. プラグインを無効にします。

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. プラグインを削除します。

```
docker plugin rm netapp:latest
```

ボリュームを操作します

必要に応じてTridentドライバ名を指定した標準コマンドを使用すると、ボリュームの作成、クローニング、および削除を簡単に実行でき `docker volume`ます。

ボリュームを作成します

- デフォルトの名前を使用して、ドライバでボリュームを作成します。

```
docker volume create -d netapp --name firstVolume
```

- 特定のTridentインスタンスを使用してボリュームを作成します。

```
docker volume create -d ntap_bronze --name bronzeVolume
```

> メモ　何も指定しない場合 "オプション（ Options ）"、ドライバのデフォルトが使用されます。

- デフォルトのボリュームサイズを上書きします。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
```

> ヒント　ボリュームサイズは、オプションの単位（ 10G 、 20GB 、 3TiB など）を含む整数値で指定します。単位を指定しない場合、デフォルトは g ですサイズの単位は、 2 の累乗（ B 、 KiB 、 MiB 、 GiB 、 TiB ）または 10 の累乗（ B 、 KB 、 MB 、 GB 、 TB ）のいずれかです。略記単位では、 2 の累乗が使用されます（ G=GiB 、 T=TiB 、…）。

ボリュームを削除します

- 他の Docker ボリュームと同様にボリュームを削除します。

```
docker volume rm firstVolume
```

> 重要　「 olidfire -san 」ドライバを使用する場合、上記の例ではボリュームを削除してパージします。

Trident for Dockerをアップグレードするには、次の手順を実行します。

ボリュームのクローンを作成します

使用する際は `ontap-nas`、 `ontap-san` 、 そして `solidfire-san`ストレージ ドライバーを使用すると、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。

- ボリュームを調べて Snapshot を列挙します。

```
docker volume inspect <volume_name>
```

- 既存のボリュームから新しいボリュームを作成します。その結果、新しい Snapshot が作成されます。

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume>
```

- ボリューム上の既存の Snapshot から新しいボリュームを作成します。新しい Snapshot は作成されません。

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

例

```
docker volume inspect firstVolume

[
  {
    "Driver": "ontap-nas",
    "Labels": null,
    "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
    "Name": "firstVolume",
    "Options": {},
    "Scope": "global",
    "Status": {
      "Snapshots": [
        {
          "Created": "2017-02-10T19:05:00Z",
          "Name": "hourly.2017-02-10_1505"
        }
      ]
    }
  }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
```

```
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

外部で作成したブロックデバイス（またはそのクローン）には、パーティションがなく、ファイルシステムがTridentでサポートされている場合（例：-formatted `/dev/sdc1`はTrident経由でアクセスできません）にのみ、`Trident *`を使用してコンテナからアクセスできます `ext4`。

ドライバ固有のボリュームオプション

ストレージドライバにはそれぞれ異なるオプションがあり、ボリュームの作成時に指定することで結果をカスタマイズできます。構成済みのストレージシステムに適用されるオプションについては、以下を参照してください。

ボリューム作成処理では、これらのオプションを簡単に使用できます。CLI の操作中に '-o' 演算子を使用して ' オプションと値を指定しますこれらは、 JSON 構成ファイルの同等の値よりも優先されます。

ONTAP ボリュームのオプション

NFS、iSCSI、およびFCのボリューム作成オプションには、次のものがあります。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 1GiB です。
「平和のための準備」を参照してください	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値は 'none(シン・プロビジョニング) と 'volume(シック・プロビジョニング) です
「ナプショットポリシー」	Snapshot ポリシーが目的の値に設定されます。デフォルトはで、 `none`ボリュームのSnapshotは自動的に作成されません。ストレージ管理者が変更しないかぎり、「default」というポリシーは、毎時6個、日次2個、週次2個のSnapshotを作成および保持するすべてのONTAPシステムに存在します。ボリューム内の任意のディレクトリを参照することで、Snapshotに保存されているデータをリカバリでき `.snapshot`ます。

オプション	説明
「スナップショット予約」	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5% ）。Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、 ONTAP-NAS-エコノミー を除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
'plitOnClone	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトはです `false`。クローンボリュームのクローニングは、作成直後に親からクローンをスプリットする方法を推奨します。これは、ストレージ効率化の効果がまったくないためです。たとえば、空のデータベースをクローニングしても時間は大幅に短縮されますが、ストレージはほとんど削減されません。そのため、クローンはすぐにスプリットすることを推奨します。
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。 バックエンドでNAEが有効になっている場合、Tridentでプロビジョニングされたすべてのボリュームでが有効になります。 詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。
階層ポリシー	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、 NFS * のみ * 用の追加オプションです。

オプション	説明
「 unixPermissions 」	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは ' アクセス権は '--rwxr-xr-x' または数値表記 0755 に設定され 'root' は所有者になりますテキスト形式または数値形式のどちらかを使用できます。

オプション	説明
「スナップショット方向」	これをに設定します `true` がを作成します `.snapshot` ボリュームにアクセスしているクライアントから認識できるディレクトリ。デフォルト値はです `false`の可視性を意味します `.snapshot` ディレクトリはデフォルトで無効になっています。一部のイメージ（公式のMySQLイメージなど）が、 `.snapshot` ディレクトリが表示されます。
「 exportPolicy 」と入力します	ボリュームで使用するエクスポートポリシーを設定します。デフォルトは「デフォルト」です。
'ecurityStyle'	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトは「 unix 」です。有効な値は「 unix 」と「 immimixed 」です。

以下の追加オプションは、 iSCSI * のみ * 用です。

オプション	説明
「 filesystemtype 」です	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトは「 ext4 」です。有効な値は「 ext3 」、「 ext4 」、「 xfs 」です。
「平和の配分」	これをに設定します `false` LUNのスペース割り当て機能を無効にします。デフォルト値はです `true`つまり、ボリュームのスペースが不足し、ボリューム内のLUNに書き込みを受け付けられなくなったときに、ONTAP からホストに通知されます。また、このオプションで ONTAP 、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 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。

スナップショット予約が指定されておらず、スナップショットポリシーがの場合、 `none`Tridentは0%のスナップショット予約を使用します。

- Snapshot ポリシーがなく、 Snapshot リザーブがないボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- Snapshot ポリシーがなく、カスタムの Snapshot リザーブが 10% のボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- Snapshot ポリシーを使用し、カスタムの Snapshot リザーブを 10% に設定してボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- Snapshotポリシーを使用してボリュームを作成し、ONTAPのデフォルトのSnapshotリザーブ（通常は5%）をそのまま使用します。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element ソフトウェアのボリュームオプション

Element ソフトウェアのオプションでは、ボリュームに関連付けられているサービス品質（ QoS ）ポリシーのサイズと QoS を指定できます。ボリュームが作成されると '-o type=service_level' という命名法を使用して ' ボリュームに関連付けられた QoS ポリシーが指定されます

Element ドライバを使用して QoS サービスレベルを定義する最初の手順は、少なくとも 1 つのタイプを作成し、構成ファイル内の名前に関連付けられた最小 IOPS 、最大 IOPS 、バースト IOPS を指定することです。

Element ソフトウェアのその他のボリューム作成オプションは次のとおりです。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 1 GiB または構成エントリ…「defaults」: {"size": "5G"}。

オプション	説明
「ブロックサイズ」	512 または 4096 のいずれかを使用します。デフォルトは 512 または config エントリ DefaultBlockSize です。

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

```
{
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}
```

上記の構成では、 Bronze 、 Silver 、 Gold の 3 つのポリシー定義を使用します。これらの名前は任意です。

- 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
```

ログを収集します

トラブルシューティングに役立つログを収集できます。ログの収集方法は、 Docker プラグインの実行方法によって異なります。

トラブルシューティング用にログを収集する

手順

1. 推奨される管理プラグイン方式を使用して（コマンドを使用して）Tridentを実行している場合は `docker plugin`、次のように表示します。

```
docker plugin ls
```

```
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin    false
journalctl -u docker | grep 4fb97d2b956b
```

標準的なロギングレベルでは、ほとんどの問題を診断できます。十分でない場合は、デバッグロギングを有効にすることができます。

2. デバッグロギングをイネーブルにするには、デバッグロギングをイネーブルにしてプラグインをインストールします。

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

または、プラグインがすでにインストールされている場合にデバッグログを有効にします。

```
docker plugin disable <plugin>
```

```
docker plugin set <plugin> debug=true
```

```
docker plugin enable <plugin>
```

3. ホストでバイナリ自体を実行している場合、ログはホストの `/var/log/netappdvp` ディレクトリ。デバッグロギングを有効にするには、を指定します `-debug` プラグインを実行すると、

一般的なトラブルシューティングのヒント

- 新しいユーザーが実行する最も一般的な問題は、プラグインの初期化を妨げる構成ミスです。この場合、プラグインをインストールまたは有効にしようとすると、次のようなメッセージが表示されることがあります。

  「デーモンからのエラー応答 : ダイヤル UNIM/run/docx/plugins/<id>/NetApp/smock: connect: no such file or directory`

  これは、プラグインの起動に失敗したことを意味します。幸い、このプラグインには、発生する可能性の高い問題のほとんどを診断するのに役立つ包括的なログ機能が組み込まれています。

- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl ステータス rpcbind またはそれに相当する処理を実行することで確認できます

複数のTridentインスタンスの管理

複数のストレージ構成を同時に使用する必要がある場合は、 Trident の複数のインスタンスが必要です。複数のインスタンスの鍵は、コンテナ化されたプラグインでは「 --alias 」オプション、ホストで Trident をインスタンス化する場合は「 --volume-driver 」オプションを使用して、それぞれ異なる名前を指定することです。

Docker Managed Plugin （バージョン 1.13 / 17.03 以降）の手順

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. 別のエイリアスと構成ファイルを指定して、 2 番目のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

   たとえば、 gold ボリュームの場合：

```
docker volume create -d gold --name ntapGold
```

たとえば、 Silver ボリュームの場合：

```
docker volume create -d silver --name ntapSilver
```

従来の（バージョン **1.12** 以前）の場合の手順

1. カスタムドライバ ID を使用して NFS 設定でプラグインを起動します。

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. カスタムドライバ ID を使用して、 iSCSI 構成でプラグインを起動します。

```
sudo trident --volume-driver=netapp-san --config=/path/to/config
-iscsi.json
```

3. ドライバインスタンスごとに Docker ボリュームをプロビジョニングします。

   たとえば、 NFS の場合：

```
docker volume create -d netapp-nas --name my_nfs_vol
```

   たとえば、 iSCSI の場合：

```
docker volume create -d netapp-san --name my_iscsi_vol
```

ストレージ構成オプション

Trident構成で使用可能な設定オプションを参照してください。

グローバル構成オプション

これらの設定オプションは、使用するストレージプラットフォームに関係なく、すべてのTrident構成に適用されます。

オプション	説明	例
「バージョン」	構成ファイルのバージョン番号	1

オプション	説明	例
'torageDriverName'	ストレージドライバの名前	`ontap-nas、 ontap-san、 ontap-nas-economy、 ontap-nas-flexgroup、 solidfire-san`
'toragePrefix'	ボリューム名のオプションのプレフィックス。デフォルト： `netappdvp_`。	`staging_`
「 limitVolumeSize 」と入力します	ボリュームサイズに関するオプションの制限。デフォルト：""（強制なし）	`10g`

ヒント	Elementバックエンドには（デフォルトを含めて）使用しない `storagePrefix`でください。デフォルトでは、 `solidfire-san`ドライバはこの設定を無視し、プレフィックスは使用しません。NetAppでは、Dockerボリュームマッピングに特定のtenantIDを使用するか、名前のマンジングが使用されている可能性がある場合には、DockerからDockerのバージョン、ドライバ情報、およびraw名が入力された属性データを使用することを推奨しています。

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。「 size 」オプションは、すべてのコントローラタイプで使用できます。デフォルトのボリュームサイズの設定方法の例については、 ONTAP の設定に関するセクションを参照してください。

オプション	説明	例
「 size 」	新しいボリュームのオプションのデフォルトサイズ。デフォルト： `1G`	`10G`

ONTAP の設定

ONTAP を使用する場合は、上記のグローバル構成値に加えて、次のトップレベルオプションを使用できます。

オプション	説明	例
「管理 LIF 」	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name （ FQDN ；完全修飾ドメイン名）を指定できます。	`10.0.0.1`

オプション	説明	例
「重複排除	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*：NetAppでは、を指定することを推奨しています `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSのマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定すると、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散できます。 • ONTAP SANドライバ*：iSCSIまたはFCには指定しないでください。Tridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCSI LIFまたはFC LIFを検出します。が明示的に定義されている場合は、警告が生成され `dataLIF`ます。	`10.0.0.2`
'VM'	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	`svm_nfs`
「ユーザ名」	ストレージデバイスに接続するユーザ名	`vsadmin`
「 password 」と入力します	ストレージ・デバイスに接続するためのパスワード	`secret`
「集約」	プロビジョニング用のアグリゲート（オプション。設定する場合は SVM に割り当てる必要があります）。ドライバの場合 `ontap-nas-flexgroup`、このオプションは無視されます。SVMに割り当てられたすべてのアグリゲートを使用して、FlexGroupボリュームがプロビジョニングされます。	`aggr1`
「 AggreglimitateUsage 」と入力します	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	`75%`

オプション	説明	例
「 nfsvMountOptions 」のように入力します	NFSマウントオプションをきめ細かく制御します。デフォルトは「 -o nfsvers=3」です。ドライバと`**ontap-nas-economy**`ドライバでのみ使用でき`**ontap-nas**`ます。 "ここでは、 NFS ホストの設定情報を参照してください"です。	`-o nfsvers=4`
「 igroupName 」と入力します	Tridentでは、ノードごとにASを`netappdvp`作成および管理し`igroups`ます。 この値は変更したり省略したりすることはできません。 でのみ使用できます **`ontap-san`** ドライバ。	`netappdvp`
「 limitVolumeSize 」と入力します	要求可能な最大ボリュームサイズ。	`300g`
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数は [50 、 300] の範囲で指定する必要があります。デフォルトは 200 です。 *のため `ontap-nas-economy` ドライバ。このオプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	`300`
`sanType`	*ドライバでのみサポートされてい`ontap-san`ます。*`iSCSI`、`nvme`NVMe/TCP、または`fcp`SCSI over Fibre Channel（FC；SCSI over Fibre Channel）に対してを選択します `iscsi`。	`iscsi` 空白の場合
`limitVolumePoolSize`	* `ontap-san-economy`および`ontap-san-economy`ドライバでのみサポートされています。*ONTAP ONTAP NASエコノミードライバおよびONTAP SANエコノミードライバでFlexVolサイズを制限します。	`300g`

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。

オプション	説明	例
「平和のための準備」を参照してください	スペースリザベーションモード `none` （シンプロビジョニング）または `volume` （シック）	「 NONE 」

オプション	説明	例
「ナプショットポリシー」	使用するSnapshotポリシー。デフォルトはです `none`	「 NONE 」
「スナップショット予約」	Snapshotリザーブの割合。デフォルトはONTAPのデフォルトを使用する場合は""です。	`10`
'plitOnClone	作成時に親からクローンをスプリットします。デフォルトはです `false`	「偽」
「暗号化」	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトは「false」です。このオプションを使用するには、クラスタでNVE のライセンスが設定され、有効になっている必要があります。 バックエンドでNAEが有効になっている場合、Tridentでプロビジョニングされたすべてのボリュームでが有効になります。 詳細については、を参照してください"TridentとNVEおよびNAEとの連携"。	正しいです
「 unixPermissions 」	プロビジョニングされたNFSボリュームのNASオプション。デフォルトはです `777`	`777`
「スナップショット方向」	ディレクトリにアクセスするためのNASオプション `.snapshot`。	`.snapshot`ディレクトリへのアクセスを有効または無効にします。 有効な値： `true`、 `false`。デフォルトの動作は、ONTAPおよびTridentのバージョンによって異なる場合があります。一貫した動作を保証するために、明示的に設定してください。
「 exportPolicy 」と入力します	NFSエクスポートポリシーで使用するNASオプション。デフォルトはです `default`	`default`
'ecurityStyle'	プロビジョニングされたNFSボリュームにアクセスするためのNASオプション。 NFSのサポート `mixed` および `unix` セキュリティ形式デフォルトはです `unix`。	`unix`
「 filesystemtype 」です	ファイルシステムタイプを選択するためのSANオプション。デフォルトはです `ext4`	`xfs`
階層ポリシー	使用する階層化ポリシー。デフォルトはです `none`。	「 NONE 」
`skipRecoveryQueue`	ボリュームの削除中は、ストレージ内のリカバリ キューをバイパスし、ボリュームを直ちに削除します。	``

スケーリングオプション

ドライバと `ontap-san`ドライバを使用すると、 `ontap-nas`DockerボリュームごとにONTAP FlexVolが作成されます。ONTAPは、クラスタノードあたり最大1000個のFlexVol、最大12、000個のFlexVolをサポートします。Dockerボリュームの要件がこの制限の範囲内に収まる場合、 `ontap-nas`Dockerボリューム単位のSnapshotやクローニングなど、FlexVolによって提供される追加機能のため、ドライバが推奨されるNASソリューションです。

FlexVol の制限で対応できる容量よりも多くの Docker ボリュームが必要な場合は、「 ONTAP - NAS - エコノミー」または「 ONTAP - SAN - エコノミー」ドライバを選択します。

`ontap-nas-economy`ドライバは、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 の拡張性は、クラスタノードあたり最大 10 、 000 、クラスタあたり最大 2 、 40 、 000 で、一部の機能を犠牲にすることで大幅に向上しています。この `ontap-nas-economy`ドライバは、Dockerボリューム単位のSnapshotやクローニングをサポートしていません。

メモ	Docker Swarmでは複数のノード間でのボリューム作成のオーケストレーションが行われないため、この `ontap-nas-economy`ドライバは現在Docker Swarmでサポートされていません。

`ontap-san-economy`ドライバは、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。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、 SAN ワークロードのスケーラビリティが向上します。ストレージアレイに応じて、 ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、 Docker ボリューム単位の Snapshot とクローニングをサポートします。

ドライバを選択する `ontap-nas-flexgroup`と、数十億個のファイルを含むペタバイト規模まで拡張可能な単一ボリュームへの並列処理を強化できます。FlexGroup のユースケースとしては、 AI / ML / DL 、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Tridentでは、FlexGroupボリュームのプロビジョニング時に、SVMに割り当てられているすべてのアグリゲートが使用されます。Trident での FlexGroup のサポートでは、次の点も考慮する必要があります。

- ONTAP バージョン 9.2 以降が必要です。
- 本ドキュメントの執筆時点では、 FlexGroup は NFS v3 のみをサポートしています。
- SVM で 64 ビットの NFSv3 ID を有効にすることを推奨します。
- 推奨されるFlexGroupメンバー/ボリュームの最小サイズは 100 GiB です。
- FlexGroupボリュームではクローニングはサポートされていません。

FlexGroupsおよびFlexGroupsに適したワークロードの詳細については、 "NetApp FlexGroupボリュームベストプラクティスおよび実装ガイド"を参照してください。

1つの環境で高度な機能と大規模な機能を利用するには、を使用して、別のを使用 `ontap-nas-economy`し

て、Docker Volume Pluginの複数のインスタンスを実行し `ontap-nas`ます。

Trident用のカスタムONTAPロール

Tridentで処理を実行するためにONTAP adminロールを使用する必要がないように、最小Privilegesを持つONTAPクラスタロールを作成できます。Tridentバックエンド構成にユーザ名を含めると、Trident作成したONTAPクラスタロールが使用されて処理が実行されます。

Tridentカスタムロールの作成の詳細については、を参照してください"Tridentカスタムロールジェネレータ"。

ONTAP CLIノシヨウ

1. 次のコマンドを使用して新しいロールを作成します。

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. Tridentユーザのユーザ名を作成します。

```
security login create -username <user_name\> -application ontapi
-authmethod password -role <name_of_role_in_step_1\> -vserver <svm_name\>
-comment "user_description"
security login create -username <user_name\> -application http -authmethod
password -role <name_of_role_in_step_1\> -vserver <svm_name\> -comment
"user_description"
```

3. ユーザにロールをマッピングします。

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

System Managerの使用

ONTAPシステムマネージャで、次の手順を実行します。

1. カスタムロールの作成：
   a. クラスタレベルでカスタムロールを作成するには、*[クラスタ]>[設定]*を選択します。

      （または）SVMレベルでカスタムロールを作成するには、*[ストレージ]>[Storage VM]>[設定]>[ユーザとロール]*を選択し `required SVM`ます。

   b. の横にある矢印アイコン（→*）を選択します。
   c. [Roles]*で[+Add]*を選択します。
   d. ロールのルールを定義し、*[保存]*をクリックします。
2. ロールをTridentユーザにマップする:+[ユーザとロール]ページで次の手順を実行します。
   a. で[アイコンの追加]+*を選択します。
   b. 必要なユーザ名を選択し、* Role *のドロップダウンメニューでロールを選択します。
   c. [保存（ Save ）] をクリックします。

詳細については、次のページを参照してください。

- "ONTAPの管理用のカスタムロール"または"カスタムロールの定義"
- "ロールとユーザを使用する"

ONTAP 構成ファイルの例

▼ <code>ontap-nas</code> ドライバのNFSの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

▼ <code>ontap-nas-flexgroup</code> ドライバのNFSの例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-flexgroup",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

▼ <code>ontap-nas-economy</code> ドライバのNFSの例

```
{
  "version": 1,
```

```
  "storageDriverName": "ontap-nas-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
```

▼ **<code>ontap-san</code>** ドライバの**iSCSI**の例

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

▼ **<code>ontap-san-economy</code>** ドライバの**NFS**の例

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

▼ **<code>ontap-san</code>** ドライバの**NVMe/TCP**の例

```
{
  "version": 1,
  "backendName": "NVMeBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_nvme",
```

```
  "username": "vsadmin",
  "password": "password",
  "sanType": "nvme",
  "useREST": true
}
```

▼ SCSI-SAN <code> ONTAP </code>ドライバの例

```
{
  "version": 1,
  "backendName": "ontap-san-backend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "sanType": "fcp",
  "svm": "trident_svm",
  "username": "vsadmin",
  "password": "password",
  "useREST": true
}
```

Element ソフトウェアの設定

Element ソフトウェア（ NetApp HCI / SolidFire ）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
「エンドポイント」	\ <a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">https://<login>:<password>@<mvip>/json-rpc/<element-version>;	\ https://admin:admin@192.168.160.3/json-rpc/8.0
「 VIP 」	iSCSI の IP アドレスとポート	10.0.0.7 ： 3260
「 tenantname 」	使用する SolidFire テナント（見つからない場合に作成）	`docker`
「 InitiatorIFCace 」	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	`default`
「タイプ」	QoS の仕様	以下の例を参照してください

オプション	説明	例
「 LegacyNamePrefix 」のように入力します	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンのTridentを使用していて、既存のボリュームでアップグレードを実行した場合は、volume-nameメソッドでマッピングされた古いボリュームにアクセスするためにこの値を設定する必要があります。	`netappdvp-`

「 olidfire -san 」ドライバは Docker Swarm をサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
  "SVIP": "10.0.0.7:3260",
  "TenantName": "docker",
  "InitiatorIFace": "default",
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
```

```
    ]
}
```

既知の問題および制限事項

ここでは、TridentでDockerを使用する場合の既知の問題および制限事項について説明します。

Trident Docker Volume Plugin を旧バージョンから 20.10 以降にアップグレードすると、該当するファイルエラーまたはディレクトリエラーなしでアップグレードが失敗します。

回避策

1. プラグインを無効にします。

```
docker plugin disable -f netapp:latest
```

2. プラグインを削除します。

```
docker plugin rm -f netapp:latest
```

3. 追加の 'config' パラメータを指定して ' プラグインを再インストールします

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

ボリューム名は 2 文字以上にする必要があります。

メモ	これは Docker クライアントの制限事項です。クライアントは、 1 文字の名前を Windows パスと解釈します。 "バグ 25773 を参照"。

Docker Swarmには特定の動作があり、ストレージとドライバの組み合わせごとにTridentがDocker Swarmをサポートできないようになっています。

- Docker Swarm は現在、ボリューム ID ではなくボリューム名を一意のボリューム識別子として使用します。
- ボリューム要求は、 Swarm クラスタ内の各ノードに同時に送信されます。
- ボリュームプラグイン（Tridentを含む）は、Swarmクラスタ内の各ノードで個別に実行する必要があります。ONTAPの動作方法とドライバと `ontap-san`ドライバの機能により、 `ontap-nas`これらの制限内で動作できるのはこれらのドライバだけです。

残りのドライバはレースコンディションなどの問題の影響を受け、明確な「勝者」なしで1回の要求で多数の

ボリュームが作成される可能性があります。たとえば、Elementには、同じ名前で複数のIDを使用できる機能があります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。

ベストプラクティスと推奨事項

Trident導入に関する推奨事項

Tridentを導入する際には、ここに記載されている推奨事項に従ってください。

専用のネームスペースに導入します

"ネームスペース"異なるアプリケーション間で管理を分離し、リソース共有の障壁となります。たとえば、あるネームスペースの PVC を別のネームスペースから使用することはできません。Tridentは、Kubernetesクラスタ内のすべてのネームスペースにPVリソースを提供するため、Privilegesを昇格させたサービスアカウントを利用します。

また、 Trident ポッドにアクセスすると、ユーザがストレージシステムのクレデンシャルやその他の機密情報にアクセスできるようになります。アプリケーションユーザと管理アプリケーションが Trident オブジェクト定義またはポッド自体にアクセスできないようにすることが重要です。

クォータと範囲制限を使用してストレージ消費を制御します

Kubernetes には、 2 つの機能があります。これらの機能を組み合わせることで、アプリケーションによるリソース消費を制限する強力なメカニズムが提供されます。。 "ストレージクォータメカニズム" 管理者は、グローバルおよびストレージクラス固有の、容量とオブジェクト数の使用制限をネームスペース単位で実装できます。さらに、を使用します "範囲制限" 要求がプロビジョニングツールに転送される前に、 PVC 要求が最小値と最大値の両方の範囲内にあることを確認します。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "クォータの活用方法"。

Tridentストレージ構成のベストプラクティス

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の **SVM** を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします "SVM-DR" Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください "NetApp Hardware Universe の略" 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、 Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident の「 ONTAP - NAS 」と「 ONTAP - SAN 」ドライバは、作成された Kubernetes 永続ボリューム（ PV ）ごとに FlexVol をプロビジョニングします。「 ONTAP-NAS-エコノミー 」ドライバは、 PVS 200 個につき約 1 つの FlexVol を作成します（ 50 ～ 300 の範囲で構成可能）。「 ONTAP-SAN-エコノミー 」ドライバは、 PVS 100 個につき約 1 つの FlexVol を作成します（ 50 ～ 200 の範囲で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

「 mAX-VOLUMES 」の値は、環境に固有のいくつかの条件によって異なります。

- ONTAP クラスタ内の既存のボリュームの数
- 他のアプリケーション用に Trident 外部でプロビジョニングするボリュームの数
- Kubernetes アプリケーションで消費されると予想される永続ボリュームの数

「 mAX-volumes 」の値は、 ONTAP クラスタ内のすべてのノードでプロビジョニングされたボリュームの合計数であり、個々の ONTAP ノードではプロビジョニングされません。その結果、 ONTAP クラスタノードの

Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、2ノードONTAPクラスタでは、最大2、000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 に設定していると、非常に妥当な結果が得られます。ただし、SVMに1つのノードからしか割り当てられていない場合や、一方のノードから割り当てられたアグリゲートを（容量などの理由で）プロビジョニングできない場合は "アグリゲート"、Tridentでプロビジョニングされるすべてのボリュームのターゲットにもう一方のノードがなります。つまり、の値に達する前にそのノードのボリューム制限に達する可能性があり、その結果、Tridentとそのノードを使用するその他のボリューム処理の両方に影響が及ぶ可能性があり `max-volumes`ます。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

ボリュームのクローンを作成します

NetApp Tridentは、 `ontap-nas` 、 `ontap-san` 、 そして `solidfire-san`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。

警告	異なるストレージクラスに関連付けられたPVCのクローン作成は避けてください。互換性を確保し、予期しない動作を防ぐため、クローン作成操作は同じストレージクラス内で実行してください。

Trident で作成できるボリュームの最大サイズを制限

Trident で作成できるボリュームの最大サイズを設定するには、「 backend.json 」の定義で「 limitVolumeSize 」パラメータを使用します。

ストレージアレイでボリュームサイズを制御するだけでなく、 Kubernetes の機能も利用する必要があります。

Tridentで作成される**FlexVol**の最大サイズを制限する

ONTAPドライバSAN-EconomyドライバおよびONTAP NAS-Economyドライバのプールとして使用されるFlexVolの最大サイズを設定するには、 `limitVolumePoolSize` `backend.json`定義でパラメータを使用します。

双方向 **CHAP** を使用するように **Trident** を設定します

バックエンド定義で CHAP イニシエータとターゲットのユーザ名とパスワードを指定し、 Trident を使用して SVM で CHAP を有効にすることができます。を使用する `useCHAP` バックエンド構成のパラメータであるTridentは、CHAPを使用してONTAP バックエンドのiSCSI接続を認証します。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、 Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。これはに役立ちます "Bully を防止します" Trident SVM 外のワークロードに影響を及ぼす、制御不能なコンテナ。

SVM の QoS ポリシーはいくつかの手順で作成します。正確な情報については、ご使用の ONTAP バージョンのマニュアルを参照してください。次の例は、 SVM で使用可能な合計 IOPS を 5000 に制限する QoS ポリシーを作成します。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

また、使用しているバージョンの ONTAP でサポートされている場合は、最小 QoS を使用してコンテナ化されたワークロードへのスループットを保証することもできます。アダプティブ QoS は SVM レベルのポリシーには対応していません。

コンテナ化されたワークロード専用の IOPS は、さまざまな要素によって異なります。その中には、次のようなものがあります。

- ストレージアレイを使用するその他のワークロード。Kubernetes 環境とは関係なく、ストレージリソースを利用するほかのワークロードがある場合は、それらのワークロードが誤って影響を受けないように注意する必要があります。
- 想定されるワークロードはコンテナで実行されます。IOPS 要件が高いワークロードをコンテナで実行する場合は、 QoS ポリシーの値が低いとエクスペリエンスが低下します。

SVM レベルで割り当てた QoS ポリシーを使用すると、 SVM にプロビジョニングされたすべてのボリュームで同じ IOPS プールが共有されることに注意してください。コンテナ化されたアプリケーションの 1 つまたは少数のに高い IOPS が必要な場合、コンテナ化された他のワークロードに対する Bully になる可能性があります。その場合は、外部の自動化を使用したボリュームごとの QoS ポリシーの割り当てを検討してください。

重要	ONTAP バージョン 9.8 より前の場合は、 QoS ポリシーグループを SVM * only * に割り当ててください。

Trident の **QoS** ポリシーグループを作成

Quality of Service （ QoS ；サービス品質）は、競合するワークロードによって重要なワークロードのパフォーマンスが低下しないようにします。ONTAP の QoS ポリシーグループには、ボリュームに対する QoS オプションが用意されており、ユーザは 1 つ以上のワークロードに対するスループットの上限を定義できます。QoSの詳細については、を参照してください。 "QoS によるスループットの保証"。
QoS ポリシーグループはバックエンドまたはストレージプールに指定でき、そのプールまたはバックエンドに作成された各ボリュームに適用されます。

ONTAP には、従来型とアダプティブ型の 2 種類の QoS ポリシーグループがあります。従来のポリシーグループは、最大スループット（以降のバージョンでは最小スループット）がフラットに表示されます。アダプティブ QoS では、ワークロードのサイズの変更に合わせてスループットが自動的に調整され、 TB または GB あたりの IOPS が一定に維持されます。これにより、何百何千という数のワークロードを管理する大規模な環境では大きなメリットが得られます。

QoS ポリシーグループを作成するときは、次の点に注意してください。

- バックエンド構成の「金庫」ブロックに「 QOSPolicy 」キーを設定する必要があります。次のバックエンド設定例を参照してください。

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
  - labels:
      performance: extreme
    defaults:
      adaptiveQosPolicy: extremely-adaptive-pg
  - labels:
      performance: premium
    defaults:
      qosPolicy: premium-pg
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoSポリシーグループの詳細については、を参照してください "ONTAPコマンド リファレンス"。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Tridentで作成された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環境のセキュリティ体制にとって重要な要素です。これにより、 Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がないことです。つまり、 PVS はグローバルオブジェクトですが、 PVC にバインドされている場合は、同じネームスペース内のポッドからのみアクセス可能です。* 適切な場合は、名前空間を使用して分離することが重要です。 *

Kubernetes 環境でデータセキュリティを使用する場合、ほとんどの組織で最も懸念されるのは、コンテナ内のプロセスがホストにマウントされたストレージにアクセスできることですが、コンテナ用ではないためです。 "ネームスペース" この種の妥協を防ぐように設計されています。ただし、特権コンテナという例外が 1 つあります。

権限付きコンテナは、通常よりもホストレベルの権限で実行されるコンテナです。デフォルトでは拒否されないため、を使用してこの機能を無効にしてください "ポッドセキュリティポリシー"。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にある

ターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShiftなど）や、ユーザアプリケーションをスケジュールできない他のノードを導入する場合は、ストレージリソースへのアクセスをさらに制限するために別 々 のエクスポートポリシーを使用する必要があります。これには、これらのインフラノードに導入されているサービス（ OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスタ内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Tridentはエクスポートポリシーを自動的に作成、管理できます。これにより、 Trident はプロビジョニング対象のボリュームへのアクセスを Kubernetes クラスタ内のノードに制限し、ノードの追加や削除を簡易化します。

また、エクスポートポリシーを手動で作成し、各ノードのアクセス要求を処理する 1 つ以上のエクスポートルールを設定することもできます。

- 「 vserver export-policy create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーを作成します。
- 「 vserver export-policy rule create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーにルールを追加します。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします **`showmount`** アプリケーション**SVM**用

この `showmount`機能を使用すると、`NFS`クライアントが`SVM`に照会して使用可能な`NFS`エクスポートのリストを確認できます。`Kubernetes`クラスタに導入されたポッドは、に対してコマンドを実行して、使用可能なマウント（ポッドがアクセスできないマウントを含む）のリストを受け取ることができます `showmount -e`。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

SVM レベルの ONTAP CLI コマンドを使用して、 SVM の howmount を無効にする必要があります。

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire のベストプラクティス

Trident に SolidFire ストレージを設定するためのベストプラクティスをご確認ください。

SolidFire アカウントを作成します

各 SolidFire アカウントは固有のボリューム所有者で、 Challenge Handshake Authentication Protocol （ CHAP ；チャレンジハンドシェイク認証プロトコル）クレデンシャルのセットを受け取ります。アカウントに割り当てられたボリュームには、アカウント名とその CHAP クレデンシャルを使用してアクセスするか、ボリュームアクセスグループを通じてアクセスできます。アカウントには最大 2 、 000 個のボリュームを関連付けることができますが、 1 つのボリュームが属することのできるアカウントは 1 つだけです。

QoS ポリシーを作成する

標準的なサービス品質設定を作成して保存し、複数のボリュームに適用する場合は、 SolidFire のサービス品質（ QoS ）ポリシーを使用します。

QoS パラメータはボリューム単位で設定できます。QoS を定義する 3 つの設定可能なパラメータである Min IOPS 、 Max IOPS 、 Burst IOPS を設定することで、各ボリュームのパフォーマンスが保証されます。

4KB のブロックサイズの最小 IOPS 、最大 IOPS 、バースト IOPS の値を次に示します。

IOPS パラメータ	定義（ **Definition** ）	最小価値	デフォルト値	最大値（ **4KB** ）
最小 IOPS	ボリュームに対して保証されたレベルのパフォーマンス。	50	50	15000
最大 IOPS	パフォーマンスはこの制限を超えません。	50	15000	200,000
バースト IOPS	短時間のバースト時に許容される最大 IOPS 。	50	15000	200,000

メモ	Max IOPS と Burst IOPS は最大 200 、 000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロック サイズと帯域幅は IOPS の数に直接影響します。ブロック サイズが大きくなるにつれて、システムはより大きなブロック サイズを処理するために必要なレベルまで帯域幅を増加させます。帯域幅が増加すると、システムが達成できる IOPS の数は減少します。QoS とパフォーマンスの詳細については、 "SolidFire のサービス品質"を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（ VAG ）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、プロビジョニングするボリュームへのアクセスを制御します。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。

メモ	Trident と強化された CSI プロビジョニングツールは、 CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、 SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、 CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、 CHAP 認証の有無に関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP

認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します "NetApp ライブラリ" 最新バージョンの場合。

- ONTAP *
- "『 NFS Best Practice and Implementation Guide 』を参照してください"
- "SAN の管理"（iSCSIの場合）
- "RHEL 向けの iSCSI のクイック構成"
- Element ソフトウェア *
- "SolidFire for Linux を設定しています"
- NetApp HCI *
- "NetApp HCI 導入の前提条件"
- "NetApp Deployment Engine にアクセスします"
- アプリケーションのベストプラクティス情報 *
- "ONTAP での MySQL に関するベストプラクティスです"
- "SolidFire での MySQL に関するベストプラクティスです"
- "NetApp SolidFire および Cassandra"
- "SolidFire での Oracle のベストプラクティス"
- "SolidFire での PostgreSQL のベストプラクティスです"

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です "NetApp ライブラリ" 最新のドキュメントを検索できます。

TridentをKubernetesと統合する

Tridentを統合するには、ドライバの選択と導入、ストレージクラスの設計、仮想プールの設計、永続的ボリューム要求（PVC）によるストレージプロビジョニングへの影響、ボリューム処理、Tridentを使用したOpenShiftサービスの導入など、設計とアーキテクチャの要素を統合する必要があります。

ドライバの選択と展開

ストレージシステム用のバックエンドドライバを選択して導入します。

ONTAP バックエンドドライバ

ONTAP バックエンドドライバは、使用されるプロトコルと、ストレージシステムでのボリュームのプロビジョニング方法によって異なります。そのため、どのドライバを展開するかを決定する際には、慎重に検討する必要があります。

アプリケーションに共有ストレージを必要とするコンポーネント（同じ PVC にアクセスする複数のポッド）がある場合、 NAS ベースのドライバがデフォルトで選択されますが、ブロックベースの iSCSI ドライバは非共有ストレージのニーズを満たします。アプリケーションの要件と、ストレージチームとインフラチームの快適さレベルに基づいてプロトコルを選択してください。一般的に、ほとんどのアプリケーションでは両者の違いはほとんどないため、共有ストレージ（複数のポッドで同時にアクセスする必要がある場合）が必要かどうかに基づいて判断することがよくあります。

使用可能なONTAP バックエンドドライバは次のとおりです。

- [ONTAP-NAS] ：プロビジョニングされた各 PV は、フル ONTAP FlexVol です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree で、 FlexVol あたりの qtree 数を設定できます（デフォルトは 200 ）。
- 「 ONTAP-NAS-flexgroup 」：フル ONTAP FlexGroup としてプロビジョニングされた各 PV と、 SVM に割り当てられたすべてのアグリゲートが使用されます。
- 「 ONTAP - SAN 」：プロビジョニングされた各 PV は、固有の FlexVol 内の LUN です。
- 「 ONTAP-SAN-エコノミー 」：各 PV がプロビジョニングされた LUN で、 FlexVol あたりの LUN 数を設定できます（デフォルトは 100 ）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、すべての機能がTridentを通じて公開されているわけではないことに注意してください。一部の機能は、プロビジョニング後にストレージ管理者が適用する必要があります。上付き文字の脚注は、機能やドライバごとに機能を区別します。

ONTAP NAS ドライバ	**Snapshot**	クローン	動的なエクスポートポリシー	マルチアタッチ	**QoS**	サイズ変更	レプリケーション
「 ONTAP - NAS 」	はい。	はい。	○脚注： 5[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]
「 ONTAP - NAS - エコノミー」	注：3[]	注：3[]	○脚注： 5[]	はい。	注：3[]	はい。	注：3[]
「 ONTAP-NAS-flexgroup 」	○脚注： 1[]	いいえ	○脚注： 5[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]

Tridentでは、ONTAP向けに2つのSANドライバを提供しています。その機能は次のとおりです。

ONTAP SAN ドライバ	**Snapshot**	クローン	マルチアタッチ	双方向 **CHAP**	**QoS**	サイズ変更	レプリケーション
「 ontap - san 」	はい。	はい。	○脚注： 4[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]
「 ONTAP - SAN - エコノミー」	はい。	はい。	○脚注： 4[]	はい。	注：3[]	はい。	注：3[]

> 上記の表の脚注: Yes[1]: Tridentで管理されないYes[2]: Tridentで管理されるが、PVでは管理されないNO[3]: TridentとPVで管理されないYes[4]: raw-blockボリュームでサポ

> ートYes[5]: Tridentでサポート

PV に細分化されていない機能は FlexVol 全体に適用され、 PVS （共有 FlexVol 内の qtree または LUN ）にはすべて共通のスケジュールが適用されます。

上の表に示すように 'ONTAP-NAS' と「 ONTAP-NAS-エコノミー 」の機能の多くは同じですしかし 'ONTAP-NAS-エコノミー のドライバは ' スケジュールを PV 単位で制御する機能を制限するため ' これは特に災害復旧やバックアップ計画に影響を与える可能性がありますONTAP ストレージ上で PVC クローン機能を活用したい開発チームの場合 ' これは 'ONTAP-NAS' ONTAP -SAN' または ONTAP -SAN' のいずれかのドライバを使用する場合にのみ可能です

メモ	「 olidfire -san 」ドライバは PVC のクローン作成にも対応しています。

Cloud Volumes ONTAP バックエンドドライバ

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（ NFS 、 SMB / CIFS 、 iSCSI ）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAP の互換性のあるドライバは、「 ONTAP-NAS' 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-SAN' 」、「 ONTAP-SAN-エコノミー 」です。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

ONTAP バックエンドドライバ用の**Amazon FSX**

Amazon FSx for NetApp ONTAPを使用すると、AWSにデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用しながら、使い慣れたNetAppの機能、パフォーマンス、管理機能を活用できます。FSx for ONTAPは、多くのONTAPファイルシステム機能と管理APIをサポートしています。Cloud Volume ONTAP の互換性のあるドライバはです `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san` および `ontap-san-economy`。

NetApp HCI / SolidFireバックエンドドライバ

NetApp HCI / SolidFire プラットフォームで使用される「 olidfire -SAN」 ドライバは、管理者が QoS 制限に基づいて Trident の Element バックエンドを構成するのに役立ちます。Trident によってプロビジョニングされるボリュームに特定の QoS 制限を設定するためにバックエンドを設計する場合は、バックエンドファイルの「 type 」パラメータを使用します。管理者は 'limitVolumeSize' パラメータを使用して ' ストレージ上に作成できるボリューム・サイズを制限することもできます現在、ボリュームのサイズ変更やボリュームのレプリケーションなどの Element ストレージ機能は、 'olidfire-san' ドライバではサポートされていません。これらの処理は、 Element ソフトウェアの Web UI から手動で実行する必要があります。

SolidFire ドライバ	**Snapshot**	クローン	マルチアタッチ	**CHAP**	**QoS**	サイズ変更	レプリケーション
「 olidfire -san 」	はい。	はい。	○脚注： 2 []	はい。	はい。	はい。	○脚注： 1[]

> 脚注:はい脚注: 1[]: Tridentで管理されていません脚注: 2[]: raw-blockボリュームでサポートされています

Azure NetApp Files バックエンドドライバ

Tridentは `azure-netapp-files`ドライバーを使用して"Azure NetApp Files の特長"サービスを管理します。

このドライバとその設定方法の詳細については、を参照してください"Azure NetApp Files 向けの Trident バックエンド構成"。

Azure NetApp Files ドライバ	**Snapshot**	クローン	マルチアタッチ	**QoS**	を展開します	レプリケーション
「 azure-NetApp-files 」と入力します	はい。	はい。	はい。	はい。	はい。	○脚注： 1[]

> 脚注:はい脚注: 1[]: Tridentで管理されていません

Google Cloud NetApp Volumesバックエンドドライバ

Tridentは、NASワークロード向けの `google-cloud-netapp-volumes`ドライバと、SAN（iSCSI）ワークロード向けの `google-cloud-netapp-volumes-san`をサポートしています。GCP上で実行されるOpenShiftクラスタでは、Trident 26.10以降、両方のドライバがGoogle Cloud Workload Identity Federationをサポートしています。これには、自己管理型のOpenShift Container Platform（IPI環境を含むOCP）およびOpenShift Dedicated（OSD）が含まれます。

バックエンド構成とWIFの前提条件については、"Google Cloud NetApp Volumes を設定"および"SAN ワークロード用に Google Cloud NetApp Volumes を設定"を参照してください。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは
'`toragePools’additionalStoragePools’excludeStoragePools" の 3 セットのフィルタを設定できます

パラメータを使用 `storagePools`すると、指定した属性に一致するプールだけにストレージを制限できます。パラメータは、 `additionalStoragePools`属性とパラメータで選択された一連のプールとともに、Tridentがプロビジョニングに使用する一連のプールを拡張するために使用し `storagePools`ます。どちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

excludeStoragePools' パラメータを使用して ' 属性に一致するプールの一覧を除外します

QoSポリシーをエミュレートします

ストレージクラスを設計して Quality of Service ポリシーをエミュレートする場合は ' 「メディア」属性を「 hdd 」または「 sd 」として ' ストレージクラスを作成しますストレージクラスで言及されている「メディア」属性に基づいて、 Trident は「 hdd 」アグリゲートまたは「 sd 」アグリゲートにメディア属性と一致させる適切なバックエンドを選択し、ボリュームのプロビジョニングを特定のアグリゲートに誘導します。した

がって、「メディア」属性が「 SD 」に設定されているストレージクラス Premium を作成して、プレミアム QoS ポリシーに分類できます。メディア属性を「 hdd 」に設定し、標準の QoS ポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「 IOPS 」属性を使用して、 QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用する

ストレージクラスは、シンプロビジョニングとシックプロビジョニング、 Snapshot 、クローン、暗号化などの機能が有効になっている特定のバックエンドでボリュームを直接プロビジョニングするように設計できます。使用するストレージを指定するには、必要な機能を有効にしてバックエンドに適したストレージクラスを作成します。

仮想プール

仮想プールは、すべてのTridentバックエンドで使用できます。Tridentが提供する任意のドライバを使用して、任意のバックエンドに仮想プールを定義できます。

仮想プールを使用すると、管理者はストレージクラスで参照可能なバックエンド上に抽象化レベルを作成して、バックエンドにボリュームを柔軟かつ効率的に配置できます。同じサービスクラスを使用して異なるバックエンドを定義できます。さらに、同じバックエンドに異なる特性を持つ複数のストレージプールを作成することもできます。ストレージクラスに特定のラベルを持つセレクタが設定されている場合、Tridentはボリュームを配置するすべてのセレクタラベルに一致するバックエンドを選択します。ストレージクラスセレクタのラベルが複数のストレージプールに一致する場合、Tridentはそのうちの1つをボリュームのプロビジョニング元として選択します。

仮想プールの設計

バックエンドを作成する際には、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、 Azure NetApp Filesが含まれます。

メモ	仮想プールを定義する場合は、バックエンド定義で既存の仮想プールの順序を変更しないことをお勧めします。また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

さまざまなサービスレベル/**QoS**のエミュレート

サービスクラスをエミュレートするための仮想プールを設計できます。Cloud Volume Service for Azure NetApp Files の仮想プール実装を使用して、さまざまなサービスクラスをセットアップする方法を見ていきましょう。Azure NetApp Filesバックエンドには、異なるパフォーマンスレベルを表す複数のラベルを設定します。設定 `servicelevel` 適切なパフォーマンスレベルを考慮し、各ラベルの下にその他の必要な側面を追加します。次に、異なる仮想プールにマッピングするさまざまなKubernetesストレージクラスを作成します。を使用する `parameters.selector` 各StorageClassは、ボリュームのホストに使用できる仮想プールを呼び出します。

特定の一連の側面を割り当てます

特定の側面を持つ複数の仮想プールは、単一のストレージバックエンドから設計できます。そのためには、バ

ックエンドに複数のラベルを設定し、各ラベルに必要な側面を設定します。を使用して、さまざまなKubernetesストレージクラスを作成します `parameters.selector` 異なる仮想プールにマッピングされるフィールド。バックエンドでプロビジョニングされるボリュームには、選択した仮想プールに定義された設定が適用されます。

ストレージプロビジョニングに影響する **PVC** 特性

要求されたストレージクラスを超える一部のパラメータは、PVCの作成時にTridentプロビジョニングの決定プロセスに影響する可能性があります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つがアクセスモードです。必要なモードは、ストレージ要求をホストするために選択されたバックエンドに影響を与える可能性があります。

Trident は、以下のマトリックスに記載されているアクセス方法で使用されているストレージプロトコルと一致するかどうかを試みます。これは、基盤となるストレージプラットフォームに依存しません。

	ReadWriteOnce コマンドを使用します	**ReadOnlyMany**	**ReadWriteMany**
iSCSI	はい。	はい。	○（ Raw ブロック）
NFS	はい。	はい。	はい。

NFS バックエンドが設定されていない Trident 環境に送信された ReadWriteMany PVC が要求された場合、ボリュームはプロビジョニングされません。このため、リクエスタは、アプリケーションに適したアクセスモードを使用する必要があります。

ボリューム操作

永続ボリュームの変更

永続ボリュームとは、 Kubernetes で変更不可のオブジェクトを 2 つだけ除いてです。再利用ポリシーとサイズは、いったん作成されると変更できます。ただし、これにより、ボリュームの一部の要素がKubernetes以外で変更されることが防止されるわけではありません。特定のアプリケーション用にボリュームをカスタマイズしたり、誤って容量が消費されないようにしたり、何らかの理由でボリュームを別のストレージコントローラに移動したりする場合に便利です。

メモ	Kubernetesのツリー内プロビジョニングツールは、現時点ではNFS、iSCSI、またはFC PVSのボリュームサイズ変更処理をサポートしていません。Tridentでは、NFS、iSCSI、FCの両方のボリュームの拡張がサポートされています。

作成後に PV の接続の詳細を変更することはできません。

オンデマンドのボリューム **Snapshot** を作成

Trident では、 CSI フレームワークを使用して、ボリュームスナップショットのオンデマンド作成とスナップショットからの PVC の作成がサポートされます。Snapshot は、データのポイントインタイムコピーを管理し、 Kubernetes のソース PV とは無関係にライフサイクルを管理する便利な方法です。これらの Snapshot を使用して、 PVC をクローニングできます。

Snapshot からボリュームを作成します

Trident では、ボリューム Snapshot から PersistentVolumes を作成することもできます。そのためには、PersistentVolumeClaimを作成し、ボリュームの作成元となるSnapshotとしてを指定します `datasource`。Trident は、 Snapshot にデータが存在するボリュームを作成することで、この PVC を処理します。この機能を使用すると、複数のリージョン間でデータを複製したり、テスト環境を作成したり、破損した本番ボリューム全体を交換したり、特定のファイルとディレクトリを取得して別の接続ボリュームに転送したりできます。

クラスタ内でボリュームを移動します

ストレージ管理者は、 ONTAP クラスタ内のアグリゲート間およびコントローラ間で、ストレージ利用者への無停止でボリュームを移動できます。この処理は、Tridentが使用しているSVMからアクセスできるデスティネーションアグリゲートであるかぎり、TridentまたはKubernetesクラスタには影響しません。重要なことは、アグリゲートがSVMに新しく追加されている場合は、バックエンドをTridentに再追加してリフレッシュする必要があることです。これにより、 Trident が SVM のインベントリを再設定し、新しいアグリゲートが認識されます。

ただし、バックエンド間でのボリュームの移動は Trident では自動でサポートされていません。これには、同じクラスタ内の SVM 間、クラスタ間、または別のストレージプラットフォームへの SVM の間も含まれます（ Trident に接続されているストレージシステムの場合も含む）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Trident にインポートできます。

ボリュームを展開します

Trident 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` 、 そして `azure-netapp-files`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ドライバとドライバを使用する場合 `solidfire-san`は、コマンドを使用し `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`て、Tridentで管理するKubernetesに既存のボリュームをインポートします。import volume コマンドで使用した PVC YAML または JSON ファイルは、 Trident をプロビジョニングツールとして識別するストレージクラスを指定します。NetApp HCI / SolidFire バックエンドを使用する場合は、ボリューム名が一意であることを確認してください。ボリューム名が重複している場合は、ボリュームインポート機能で区別できるように、ボリュームを一意の名前にクローニングします。

もし `azure-netapp-files`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。

上記のコマンドが実行されると、 Trident はバックエンド上のボリュームを検出してサイズを確認します。設定されたPVCのボリュームサイズを自動的に追加（および必要に応じて上書き）します。Trident が新しい PV を作成し、 Kubernetes が PVC を PV にバインド

特定のインポートされた PVC を必要とするようにコンテナを導入した場合、ボリュームインポートプロセスによって PVC/PV ペアがバインドされるまで、コンテナは保留状態のままになります。PVC/PV ペアがバインドされると、他に問題がなければコンテナが起動します。

レジストリサービス

レジストリのストレージの導入と管理については、に記載されています "netapp.io のコマンドです" を参照してください "ブログ"。

ロギングサービス

他の OpenShift サービスと同様に、ログ記録サービスは、 Ansible と、インベントリファイル（別名）で提供される構成パラメータを使用して導入されますホスト。プレイブックに含まれています。ここでは、 OpenShift の初期インストール時にロギングを導入し、 OpenShift のインストール後にロギングを導入するという、 2 つのインストール方法について説明します。

注意	Red Hat OpenShift バージョン 3.9 以降、データ破損に関する懸念があるため、記録サービスに NFS を使用しないことを公式のドキュメントで推奨しています。これは、 Red Hat 製品のテストに基づいています。ONTAP NFSサーバにはこのような問題がないため、ロギング環境を簡単にバックアップできます。ロギングサービスには最終的にどちらかのプロトコルを選択する必要がありますが、両方のプロトコルがネットアッププラットフォームを使用する場合に適していることと、 NFS を使用する理由がないことを確認してください。

ロギング・サービスで NFS を使用する場合は、インストーラが失敗しないように、 Ansible 変数「 OpenShift 」の「 OpenShift 」 enable_unsupported _configurations 」を「 true 」に設定する必要があります。

はじめに

ロギングサービスは、必要に応じて、両方のアプリケーションに導入することも、 OpenShift クラスタ自体のコア動作に導入することもできます。オペレーション・ログを配置する場合 ' 変数 OpenShift の logging_use_ops を true として指定すると ' サービスの 2 つのインスタンスが作成されます操作のロギングインスタンスを制御する変数には「 ops 」が含まれ、アプリケーションのインスタンスには含まれません。

基盤となるサービスで正しいストレージが使用されるようにするには、導入方法に応じてAnsible変数を設定することが重要です。それぞれの導入方法のオプションを見てみましょう。

メモ	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のロギングに関するドキュメント"を参照してください。これらは、導入に応じて確認、設定、および使用する必要があります。

次の表の変数では、入力した詳細を使用してロギングサービスの PV と PVC を作成する Ansible プレイブックが作成されます。この方法は、 OpenShift インストール後にコンポーネントインストールプレイブックを使用するよりもはるかに柔軟性に劣るが、既存のボリュームがある場合はオプションとなります。

変数（ **Variable** ）	詳細
「 OpenShift 」ロギング・ストレージ・タイプ	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift 」ロギング・ストレージ・ホスト	NFS ホストのホスト名または IP アドレス。この値は、仮想マシンのdataLIFに設定する必要があります。
「 OpenShift 」ロギング・ストレージ・ NFS_DIRECT'	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift _logging 」としてジャンクションされている場合、この変数にそのパスを使用します。
「 OpenShift 」ロギング・ストレージ・ボリューム名	作成する PV の名前（「 pv_ose_logs 」など）。
「 OpenShift 」ロギング・ストレージ・ボリューム・サイズ	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
'OpenShift の logging_es_vpc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
「 OpenShift logging _es_vpc_storage_class_name 」	PVC で使用されるストレージクラスの名前。
「 OpenShift logging _es_vpc_size 」を参照してください	PVC で要求されたボリュームのサイズ。
「 OpenShift logging _es_vpc_prefix 」を参照してください	ロギングサービスで使用される PVC のプレフィックス。
'OpenShift の logging_es_ops_pvc_dynamic	動的にプロビジョニングされたボリュームを ops ロギングインスタンスに使用するには、「 true 」に設定します。
「 OpenShift logging _es_ops_pvc_storage_class_name 」を参照してください	処理ロギングインスタンスのストレージクラスの名前。
'OpenShift logging _es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
「 OpenShift logging _es_ops_pvc_prefix 」を参照してください	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、初期インストール後にデプロイする場合は、Ansible でコンポーネント プレイブックを使用する必要があります。このプロセスは、OpenShift のバージョンによって若干異なる場合があるため、お使いのバージョンの"Red Hat OpenShift Container Platform 3.11のドキュメント"を必ずお読みになり、従ってください。

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動拡張機能にも必要であり、多くの組織では、チャージバックやショーバックのアプリケーションに指標サービスのデータを使用しています。

ロギングサービスや OpenShift 全体と同様に、 Ansible を使用して指標サービスを導入します。また、ロギングサービスと同様に、メトリクスサービスは、クラスタの初期セットアップ中、またはコンポーネントのインストール方法を使用して運用後に導入できます。次の表に、指標サービスに永続的ストレージを設定する際に重要となる変数を示します。

メモ	以下の表には、指標サービスに関連するストレージ構成に関連する変数のみが含まれています。このドキュメントには、他にも導入環境に応じて確認、設定、使用できるオプションが多数あります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ storage _kind 」	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift _ metrics _storage_host 」というようになります	NFS ホストのホスト名または IP アドレス。この値は、SVMのdataLIFに設定する必要があります。
「 OpenShift _ metrics _storage_nfs_directory 」というエラーが表示されます	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift メトリック」としてジャンクションされている場合は、この変数にそのパスを使用します。
「 OpenShift _ metrics _storage_volume_name 」という形式で指定します	作成する PV の名前（「 pv_ose_metrics 」など）。
「 OpenShift _ metrics _storage_volume_size 」というようになります	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ cassandra_vpc_prefix 」という形式で指定します	メトリック PVC に使用するプレフィックス。
「 OpenShift _ metrics _ cassandra_vp_size' 」のようになります	要求するボリュームのサイズ。
「 OpenShift _ metrics _ cassandra_storage_type 」のようになります	指標に使用するストレージのタイプ。適切なストレージクラスを使用して PVC を作成するには、 Ansible に対してこれを dynamic に設定する必要があります。
「 OpenShift _ metrics _cassanda_pvc_storage_class_name 」という形式で指定します	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、 Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、 PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合は、OpenShiftのインストール後にAnsibleによって必要なPVCが作成され、Tridentによってストレージがプロビジョニングされたらサービスが導入されます。

上記の変数と導入プロセスは、OpenShiftのバージョンごとに変更される可能性があります。お使いの環境に合わせて構成されるように、バージョンに応じた"Red Hat OpenShift導入ガイド"を必ず確認して従ってください。

Tridentデータ保護と災害復旧

TridentとTridentを使用して作成されたボリュームの保護とリカバリのオプションについて説明します。永続性に関する要件があるアプリケーションごとに、データ保護とリカバリの戦略を用意しておく必要があります。

Tridentのレプリケーションとリカバリ

災害発生時にTridentをリストアするバックアップを作成できます。

Tridentレプリケーション

Tridentは、Kubernetes CRDを使用して独自の状態を格納および管理し、Kubernetesクラスタetcdを使用してメタデータを格納します。

手順

1. を使用してKubernetesクラスタetcdをバックアップします "Kubernetes：etcdクラスタのバックアップ"。
2. FlexVol volumeへのバックアップアーティファクトの配置

メモ	NetAppでは、FlexVolが配置されている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リカバリ

Kubernetes CRDとKubernetesクラスタetcdスナップショットを使用して、Tridentをリカバリできます。

手順

1. デスティネーションSVMから、Kubernetes etcdデータファイルと証明書が格納されているボリュームを、マスターノードとしてセットアップするホストにマウントします。
2. Kubernetesクラスタに関連する必要な証明書をすべてのにコピーします `/etc/kubernetes/pki` およびの下のetcdメンバーファイル `/var/lib/etcd`。
3. を使用して、etcdバックアップからKubernetesクラスタをリストアします "Kubernetes：etcdクラスタをリストアします"。
4. を実行します `kubectl get crd` Tridentのカスタムリソースがすべて稼働していることを確認し、Tridentオブジェクトを読み出してすべてのデータが利用可能であることを確認します。

SVMのレプリケーションとリカバリ

Tridentではレプリケーション関係を設定できませんが、ストレージ管理者はを使用してSVMをレプリケートできます "ONTAP SnapMirrorの略"。

災害が発生した場合は、 SnapMirror デスティネーション SVM をアクティブ化してデータの提供を開始できます。システムがリストアされたら、プライマリに戻すことができます。

このタスクについて

SnapMirror SVMレプリケーション機能を使用する場合は、次の点を考慮してください。

- SVM-DRを有効にしたSVMごとに、個別のバックエンドを作成する必要があります。
- SVM-DRをサポートするバックエンドにレプリケーション不要のボリュームをプロビジョニングしないように、必要な場合にのみレプリケートされたバックエンドを選択するようにストレージクラスを設定します。
- アプリケーション管理者は、レプリケーションに伴う追加コストと複雑さを理解し、このプロセスを開始する前にリカバリプランを慎重に検討する必要があります。

SVMレプリケーション

を使用できます "ONTAP：SnapMirrorレプリケーション" をクリックしてSVMレプリケーション関係を作成します。

SnapMirrorでは、レプリケートする対象を制御するオプションを設定できます。プリフォーム時に選択したオプションを知っておく必要がTridentを使用したSVMのリカバリあります。

- "-identity-preserve trueを指定します" SVMの設定全体をレプリケートします。
- "-discard-configs network" LIFと関連ネットワークの設定を除外します。
- "-identity-preserve false" ボリュームとセキュリティ設定のみをレプリケートします。

Tridentを使用したSVMのリカバリ

Tridentでは、SVMの障害は自動的に検出されません。災害が発生した場合、管理者は新しいSVMへのTridentフェイルオーバーを手動で開始できます。

手順

1. スケジュールされた実行中のSnapMirror転送をキャンセルし、レプリケーション関係を解除し、ソースSVMを停止してからSnapMirrorデスティネーションSVMをアクティブ化します。
2. を指定した場合 `-identity-preserve false` または `-discard-config network` SVMレプリケーションを設定する場合は、を更新します `managementLIF` および `dataLIF` をTridentバックエンド定義ファイルに追加します。
3. 確認します `storagePrefix` は、Tridentバックエンド定義ファイルに含まれています。このパラメータは変更できません。省略しています `storagePrefix` バックエンドの更新が失敗するように原因します。
4. 次のコマンドを使用して、必要なすべてのバックエンドを更新して新しいデスティネーションSVM名を反映します。

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
```

```
<namespace>
```

5. を指定した場合 `-identity-preserve false` または `discard-config network`、すべてのアプリケーションポッドをバウンスする必要があります。

メモ	を指定する `-identity-preserve true`と、デスティネーションSVMがアクティブ化されたときに、Tridentによってプロビジョニングされたすべてのボリュームからデータの提供が開始されます。

ボリュームのレプリケーションとリカバリ

TridentではSnapMirrorレプリケーション関係を設定できませんが、ストレージ管理者はを使用して、Tridentで作成されたボリュームをレプリケートできます"ONTAP SnapMirrorのレプリケーションとリカバリ"。

その後、を使用して、リカバリしたボリュームをTridentにインポートできます"tridentctlボリュームインポート"。

メモ	ボリュームインポートは `ontap-san-economy`ドライバでサポートされています。 `ontap-nas-economy`ドライバを使用したQtreeのインポートはテクニカルプレビュー機能であり、本番環境での使用はサポートされていません。 `ontap-flexgroup-economy`ドライバでのインポートはサポートされていません。詳細については、"Tridentボリュームのインポート"を参照してください。

Snapshotデータの保護

次のコマンドを使用してデータを保護およびリストアできます。

- 永続ボリューム（PV）のKubernetesボリュームSnapshotを作成するための外部のSnapshotコントローラとCRD。

  "ボリューム Snapshot"

- ONTAP Snapshot：ボリュームの内容全体のリストア、または個 々 のファイルまたはLUNのリカバリに使用します。

  "ONTAPスナップショット"

Tridentによるステートフル アプリケーションのフェイルオーバーの自動化

Trident の強制デタッチ機能を使用すると、Kubernetes クラスター内の異常なノードからボリュームを自動的にデタッチして、データの破損を防ぎ、アプリケーションの可用性を確保できます。この機能は、ノードが応答しなくなったり、メンテナンスのためにオフラインになったりするシナリオで特に役立ちます。

フォースデタッチの詳細

強制デタッチは、 `ontap-san` 、 `ontap-san-economy` 、 `ontap-nas` 、 そして `ontap-nas-economy`のみ。強制デタッチを有効にする前に、Kubernetes クラスターで非正常なノード シャットダウン (NGNS) を有効にする必要があります。Kubernetes 1.28 以降では、NGNS はデフォルトで有効になっています。詳細については、 "Kubernetes：正常なノードシャットダウンではありません" 。

メモ	ドライバまたは `ontap-nas-economy`ドライバを使用する場合 `ontap-nas`は、管理対象のエクスポートポリシーを使用してtaintが適用されたKubernetesノードからのアクセスをTridentが制限できるように、バックエンド構成のパラメータをに `true`設定する必要 `autoExportPolicy`があります。
警告	TridentはKubernetes NGNに依存しているため、許容できないすべてのワークロードのスケジュールを再設定するまで、正常でないノードからテイントを削除しないで `out-of-service`ください。汚染を無謀に適用または削除すると、バックエンドのデータ保護が危険にさらされる可能性があります。

Kubernetesクラスタ管理者がtaintをノードに適用し、 `enableForceDetach`をに設定する `true`と `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute`、Tridentはノードのステータスを確認し、次の処理を行います。

1. そのノードにマウントされたボリュームのバックエンド I/O アクセスを停止します。
2. Tridentノードオブジェクトを（新しいパブリケーションに対しては安全ではない）としてマークします `dirty`。

メモ	Tridentコントローラは、Tridentノードポッドによって（とマークされた後で）ノードが再修飾されるまで、新しいパブリッシュボリューム要求を拒否し `dirty`ます。マウントされたPVCを使用してスケジュールされたワークロード（クラスタノードが正常で準備が完了したあとも）は、 `Trident`がそのノードを検証できるようになるまで受け入れられません `clean`（新しいパブリケーションに対して安全）。

ノードの健常性が回復してtaintが削除されると、Tridentは次の処理を実行します。

1. ノード上の古い公開パスを特定してクリーンアップします。
2. ノードが状態（アウトオブサービス状態が削除され、ノードが `Ready`状態）で、古い公開パスがすべてクリーンである場合、 `cleanable`Tridentはノードをとして再登録し、新しい公開ボリュームをそのノードに許可します `clean`。

自動フェイルオーバーの詳細

統合により強制デタッチプロセスを自動化できます。"ノードヘルスチェック（NHC）オペレーター" 。ノード障害が発生すると、NHC は、障害が発生したノードを定義する Trident の名前空間に TridentNodeRemediation CR を作成して、 Tridentノード修復 (TNR) をトリガーし、自動的に強制的にデタッチします。TNR はノード障害時にのみ作成され、ノードがオンラインに戻るかノードが削除されると NHC によって削除されます。

ノードポッドの削除プロセスが失敗しました

自動フェイルオーバーは、障害が発生したノードから削除するワークロードを選択します。TNR が作成され

ると、TNR コントローラーはノードをダーティとしてマークし、新しいボリュームの公開を防止し、強制デタッチがサポートされているポッドとそのボリューム アタッチメントの削除を開始します。

強制デタッチでサポートされているすべてのボリューム/PVC は、自動フェイルオーバーでもサポートされます。

- 自動エクスポート ポリシーを使用する NAS および NAS エコノミー ボリューム (SMB はまだサポートされていません)。
- SAN および SAN エコノミー ボリューム。

参照[フォースデタッチの詳細]。

デフォルトの動作:

- 強制デタッチがサポートされているボリュームを使用しているポッドは、障害が発生したノードから削除されます。Kubernetes はこれらを正常なノードに再スケジュールします。
- 強制デタッチでサポートされていないボリューム (Trident 以外のボリュームを含む) を使用するポッドは、障害が発生したノードから削除されません。
- ステートレスポッド（PVCではない）は、ポッドアノテーションがない限り、障害が発生したノードから削除されません。 `trident.netapp.io/podRemediationPolicy: delete`が設定されています。

ポッド削除動作のオーバーライド:

ポッドの削除動作は、ポッド アノテーションを使用してカスタマイズできます。`trident.netapp.io/podRemediationPolicy[retain, delete]` 。これらの注釈は、フェイルオーバーが発生したときに検査され、使用されます。フェイルオーバー後に注釈が消えないように、Kubernetes デプロイメント/レプリカセット ポッド仕様に注釈を適用します。

- `retain`- 自動フェイルオーバー中に、障害が発生したノードからポッドは削除されません。
- `delete`- 自動フェイルオーバー中に、障害が発生したノードからポッドが削除されます。

これらの注釈はどのポッドにも適用できます。

警告	• 強制デタッチをサポートするボリュームの場合、障害が発生したノードでのみ I/O 操作がブロックされます。 • 強制デタッチをサポートしていないボリュームの場合、データ破損やマルチアタッチの問題が発生するリスクがあります。

トライデントノード修復 **CR**

TridentNodeRemediation (TNR) CR は、障害が発生したノードを定義します。TNR の名前は、障害が発生したノードの名前です。

TNRの例:

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediation
metadata:
```

```
  name: <K8s-node-name>
spec: {}
```

TNR の状態: TNR のステータスを表示するには、次のコマンドを使用します。
`kubectl get tnr <name> -n <trident-namespace>`

TNR は次のいずれかの状態になります。

- 修復中:
  - そのノードにマウントされた、強制デタッチでサポートされているボリュームへのバックエンド I/O アクセスを停止します。
  - Tridentノード オブジェクトはダーティ (新規発行には安全ではない) としてマークされています。
  - ノードからポッドとボリュームアタッチメントを削除します
- ノード回復保留中:
  - コントローラーはノードがオンラインに戻るのを待機しています。
  - ノードがオンラインになると、パブリッシュ強制により、ノードがクリーンであり、新しいボリュームのパブリケーションの準備ができていることが確認されます。
- ノードが K8s から削除されると、TNR コントローラーは TNR を削除し、調整を停止します。
- 成功:
  - すべての修復およびノード回復手順が正常に完了しました。ノードはクリーンであり、新しいボリュームの公開の準備ができています。
- 失敗した：
  - 回復不能なエラーです。エラー理由は、CR の status.message フィールドに設定されます。

自動フェイルオーバーの有効化

前提条件:

- 自動フェイルオーバーを有効にする前に、強制デタッチが有効になっていることを確認してください。詳細については、[フォースデタッチの詳細] 。
- Kubernetes クラスターにノード ヘルス チェック (NHC) をインストールします。
  - "オペレーターSDKをインストールする"。
  - まだインストールされていない場合は、クラスターに Operator Lifecycle Manager (OLM) をインストールします。 `operator-sdk olm install` 。
  - ノードヘルスチェックオペレーターをインストールします。 `kubectl create -f https://operatorhub.io/install/node-healthcheck-operator.yaml` 。

メモ	以下の[カスタムノードヘルスチェックソリューションの統合]セクションで指定されている別の方法を使用して、ノード障害を検出することもできます。

見る"ノードヘルスチェックオペレーター"詳細についてはこちらをご覧ください。

手順

1. クラスター内のワーカーノードを監視するには、 Trident名前空間に NodeHealthCheck (NHC) CR を作成します。例：

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: <CR name>
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  remediationTemplate:
    apiVersion: trident.netapp.io/v1
    kind: TridentNodeRemediationTemplate
    namespace: <Trident installation namespace>
    name: trident-node-remediation-template
  minHealthy: 0 # Trigger force-detach upon one or more node failures
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 0s
    - type: Ready
      status: Unknown
      duration: 0s
```

2. ノードヘルスチェックCRを `trident`名前空間。

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

上記の CR は、K8s ワーカーノードのノード状態 Ready: false および Unknown を監視するように構成されています。自動フェイルオーバーは、ノードが Ready: false または Ready: Unknown 状態になるとトリガーされます。

その `unhealthyConditions`CR では 0 秒の猶予期間が使用されます。これにより、K8s がノードからのハートビートを失った後に設定されるノード条件 Ready: false を K8s が設定すると、自動フェイルオーバーが直ちにトリガーされます。K8s では、最後のハートビートの後に Ready: false を設定するまで、デフォルトで 40 秒間待機します。この猶予期間は、K8s デプロイメント オプションでカスタマイズできます。

追加の設定オプションについては、"Node-Healthcheck-Operator ドキュメント" 。

追加のセットアップ情報

強制デタッチを有効にしてTrident をインストールすると、NHC との統合を容易にするために、 Trident名前空間に 2 つの追加リソース (TridentNodeRemediationTemplate (TNRT) と ClusterRole) が自動的に作成されます。

TridentNodeRemediationTemplate (TNRT):

TNRT は NHC コントローラーのテンプレートとして機能し、NHC コントローラーは TNRT を使用して必要に応じて TNR リソースを生成します。

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

クラスターロール:

強制デタッチが有効になっている場合、インストール中にクラスター ロールも追加されます。これにより、Trident名前空間内の TNR に NHC 権限が付与されます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.ext-remediation/aggregate-to-ext-remediation: "true"
  name: tridentnoderemediation-access
rules:
- apiGroups:
  - trident.netapp.io
  resources:
  - tridentnoderemediationtemplates
  - tridentnoderemediations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

K8s クラスターのアップグレードとメンテナンス

フェイルオーバーを防ぐには、ノードがダウンしたり再起動することが予想される K8s メンテナンスまたはアップグレード中は、自動フェイルオーバーを一時停止します。NHC CR (上記で説明) を一時停止するには、その CR にパッチを適用します。

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

これにより、自動フェイルオーバーが一時停止されます。自動フェイルオーバーを再度有効にするには、メンテナンスが完了した後、仕様から pauseRequests を削除します。

制限

- 強制デタッチでサポートされているボリュームの場合、障害が発生したノード上でのみ I/O 操作が防止されます。強制デタッチでサポートされているボリューム/PVC を使用しているポッドのみが自動的に削除されます。
- 自動フェイルオーバーと強制デタッチは、trident-controller ポッド内で実行されます。trident-controller をホストしているノードに障害が発生した場合、K8s がポッドを正常なノードに移動するまで、自動フェイルオーバーは遅延されます。

カスタムノードヘルスチェックソリューションの統合

自動フェイルオーバーをトリガーするために、Node Healthcheck Operator を代替のノード障害検出ツールに置き換えることができます。自動フェイルオーバー メカニズムとの互換性を確保するには、カスタム ソリューションで次のことを行う必要があります。

- ノード障害が検出されると、障害が発生したノードの名前を TNR CR 名として使用して TNR を作成します。
- ノードが回復し、TNR が成功状態になったら、TNR を削除します。

セキュリティ

Tridentセキュリティに関する推奨事項

ここに記載されている推奨事項を使用して、Tridentのインストールが安全であることを確認します。

独自のネームスペースで**Trident**を実行

信頼性の高いストレージを確保し、潜在的な悪意のあるアクティビティをブロックするためには、アプリケーション、アプリケーション管理者、ユーザ、管理アプリケーションがTridentオブジェクト定義やポッドにアクセスできないようにすることが重要です。

他のアプリケーションとユーザをTridentから分離するには、必ずTridentを独自のKubernetesネームスペースにインストールし(`trident`ます）。Tridentを独自のネームスペースに配置すると、Kubernetes管理者のみがTridentポッドと、名前空間CRDオブジェクトに格納されているアーティファクト（該当する場合はバックエンドやCHAPシークレットなど）にアクセスできるようになります。Tridentネームスペースへのアクセスを管理者のみに許可し、アプリケーションへのアクセスを許可する必要があり `tridentctl`ます。

ONTAP SAN バックエンドで **CHAP** 認証を使用します

Tridentでは、ONTAP SANワークロードに対してCHAPベースの認証がサポートされます（ドライバと `ontap-san-economy`ドライバを使用 `ontap-san`）。NetAppでは、ホストとストレージバックエンド間の認証にTridentで双方向CHAPを使用することを推奨しています。

SANストレージドライバを使用するONTAPバックエンドの場合、Tridentは双方向CHAPを設定し、でCHAPユーザ名とシークレットを管理できます `tridentctl`。TridentがONTAPバックエンドでCHAPを構成する方法については、を参照してください"バックエンドにONTAP SANドライバを設定する準備をします"。

NetApp HCI および **SolidFire** バックエンドで **CHAP** 認証を使用します

ホストと NetApp HCI バックエンドと SolidFire バックエンドの間の認証を確保するために、双方向の CHAP を導入することを推奨します。Tridentは、テナントごとに2つのCHAPパスワードを含むシークレットオブジェクトを使用します。Tridentをインストールすると、CHAPシークレットが管理され、それぞれのPVのCRオブジェクトに格納され `tridentvolume`ます。PVを作成すると、TridentはCHAPシークレットを使用してiSCSIセッションを開始し、CHAPを介してNetApp HCIおよびSolidFireシステムと通信します。

メモ	Tridentで作成されるボリュームは、どのボリュームアクセスグループにも関連付けられません。

NVEおよび**NAE**での**Trident**の使用

NetApp ONTAP は、保管データの暗号化を提供し、ディスクが盗難、返却、転用された場合に機密データを保護します。詳細については、を参照してください "NetApp Volume Encryption の設定の概要"。

- バックエンドでNAEが有効になっている場合、TridentでプロビジョニングされたすべてのボリュームでNAEが有効になります。
  - NVE暗号化フラグをに設定すると、NAE対応ボリュームを作成できます `""`。
- バックエンドでNAEが有効になっていない場合、バックエンド構成でNVE暗号化フラグが（デフォルト値）に設定されていないかぎり、Tridentでプロビジョニングされたボリュームは`NVE`対応になり `false`ます。

メモ	NAE対応バックエンドのTridentで作成されたボリュームは、NVEまたはNAEで暗号化する必要があります。 • Tridentバックエンド構成でNVE暗号化フラグを「true」に設定すると、NAE暗号化をオーバーライドし、ボリューム単位で特定の暗号化キーを使用できます。 • NAE対応バックエンドでNVE暗号化フラグをに設定する `false`と、`NAE`対応ボリュームが作成されます。`NVE`暗号化フラグをに設定して`NAE`暗号化を無効にすることはできません `false`。

- TridentでNVEボリュームを手動で作成するには、NVE暗号化フラグを明示的にに設定し `true`ます。

バックエンド構成オプションの詳細については、以下を参照してください。

- "ONTAP のSAN構成オプション"
- "ONTAP NASの構成オプション"

Trident Linux統合キー設定（LUKS）

Linux Unified Key Setup (LUKS) は、ディスク暗号化の仕様です。TridentはLUKSを使用して、ONTAP SANおよびONTAP SAN ECONOMYボリュームに対してボリューム単位のホスト側暗号化を提供し、パスフレーズのローテーションとボリュームの拡張をサポ

ートします。

Tridentでは、LUKSで暗号化されたボリュームでAES-XTS-plain64暗号化およびモードが使用されます（の推奨）"NIST"。

メモ	LUKS暗号化はASA r2システムではサポートされていません。ASAr2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の詳細" 。

作業を開始する前に

- ワーカーノードにはcryptsetup 2.1以上（3.0よりも下位）がインストールされている必要があります。詳細については、を参照してください "Gitlab: cryptsetup"。
- パフォーマンス上の理由から、NetAppでは、ワーカーノードでAdvanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨しています。AES-NIサポートを確認するには、次のコマンドを実行します。

```
grep "aes" /proc/cpuinfo
```

何も返されない場合、お使いのプロセッサはAES-NIをサポートしていません。AES-NIの詳細については、以下を参照してください。 "Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にします

ONTAP SANおよびONTAP SANエコノミーボリュームでは、Linux Unified Key Setup（LUKS；Linux統合キーセットアップ）を使用して、ボリューム単位のホスト側暗号化を有効にできます。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。ONTAP SANのバックエンド構成オプションの詳細については、を参照してください "ONTAP のSAN構成オプション"。

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
```

```
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. 使用 `parameters.selector` LUKS暗号化を使用してストレージプールを定義する方法。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKSパスフレーズを含むシークレットを作成します。例：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

RWX rawブロックボリュームで**LUKS**を使用する

Trident 26.10以降では、 `ReadWriteMany`（RWX）アクセスモードを使用して複数のKubernetesノードに接続されたiSCSIベースのrawブロックボリュームで、LUKS暗号化を使用できます。これは、ライブマイグレーション中にソースノードに接続されたままの状態で、宛先ノードにも同じ暗号化ボリュームを接続する必要があるKubeVirt仮想マシンなどのワークロードに役立ちます。

RWX LUKSのサポートは、rawブロックボリュームに限定されます。ファイルシステムモードのRWX LUKSボリュームはサポートされていません。アプリケーションは、復号化されたデータへの同時アクセスを調整する役割を担います。TridentはLUKS処理の調整のみを行います。

LUKS暗号化を有効にしますで説明されているように、バックエンド、StorageClass、およびパスフレーズSecretでLUKSを設定します。次に、 `volumeMode: Block`および `ReadWriteMany`を使用してPVCを作成します：

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: my-rwx-pvc
  namespace: my-app
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  storageClassName: san-luks-rwx
  resources:
    requests:
      storage: 10Gi
```

複数のノード上のポッドは、このPVCをrawブロックデバイスとして接続できます。Tridentは、最初の接続時にデバイスをLUKS2でフォーマットし、後続のノードでは既存の暗号化ボリュームを開きます。パスフレーズのローテーションやボリュームの拡張など、既存のLUKS操作がサポートされています。

TridentがLUKS操作をシリアル化する理由

LUKS2は、暗号化ヘッダーの冗長コピーをブロックデバイスの先頭に格納します。 `cryptsetup`は、読み取り専用のように見えるコマンドの実行中も含め、コピーが異なることを検出した場合にヘッダーを修復できます。そのロック機能はノード固有のものであり、他のノードでの操作を調整することはできません。1つのノードがヘッダーを更新している間に、別のノードが `cryptsetup`を実行すると、2番目のノードがヘッダー修復をトリガーして更新を中断し、ヘッダーを破損させる可能性があります。

同時ヘッダーアクセスを防止するため、Tridentは各RWX LUKSボリュームに対してKubernetesリースを使用します。ノードがボリュームに対して任意の `cryptsetup`コマンドを実行する前に、リースを保持している必要があります。Tridentは、操作が成功したあとにリースを解放します。リースが保持されている間に別のノードが操作を要求した場合、Tridentは再試行可能なエラーを返し、Kubernetesは後で再試行します。異なるボリュームに対する操作は独立しています。

> `cryptsetup`操作が失敗した場合、Tridentは意図的にリースを保持し、同じノードのみが再試行できるようにします。理由が `LeaseRetained`の警告イベントに失敗が記録されます。リースは自動的に期限切れにはなりません。管理者は、その保持ノードで `cryptsetup`操作が実行されていないことを確認したあとにのみ、リースを削除する必要があります。復旧手順については、<<rwx-luks-stuck-lease,RWX Linux Unified Key Setup (LUKS) ボリュームのリースが停止した状態を修復する>>を参照してください。

ノードが順番にボリュームを開くため、多くの異なるノードでワークロードを開始するには、暗号化されていないボリュームの場合よりも時間がかかる場合があります。Kubernetesは、現在の操作が完了したあとに、一時的なロックエラーを再試行します。ポッドは大規模なスケールアウト中、 `ContainerCreating`に長く留まる場合があります。同一ノード上のワークロードは、単一のオープンデバイスマッピングを共有できます。

制限

LUKSで暗号化されたボリュームは、ONTAP の重複排除と圧縮を利用できません。

LUKSボリュームをインポートするためのバックエンド構成

LUKSボリュームをインポートするには、バックエンドでをに(`true`設定する必要があります `luksEncryption`。このオプションを指定する `luksEncryption`と、(`false`次の例に示すように、ボリュームがLUKS準拠である(`true`かどうかがTridentに通知されます。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

LUKSボリュームをインポートするためのPVC設定

LUKSボリュームを動的にインポートするには、 `trident.netapp.io/luksEncryption` `true`次の例に示すように、アノテーションをに設定し、LUKS対応のストレージクラスをPVCに含めます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

LUKSパスフレーズをローテーションします

LUKSのパスフレーズをローテーションしてローテーションを確認できます。

警告	パスフレーズは、ボリューム、Snapshot、シークレットで参照されなくなることを確認するまで忘れないでください。参照されているパスフレーズが失われた場合、ボリュームをマウント

できず、データが暗号化されたままアクセスできなくなることがあります。

このタスクについて

LUKSパスフレーズのローテーションは、ボリュームをマウントするポッドが、新しいLUKSパスフレーズの指定後に作成されたときに行われます。新しいPODが作成されると、Tridentはボリューム上のLUKSパスフレーズをシークレット内のアクティブなパスフレーズと比較します。

- ボリュームのパスフレーズがシークレットでアクティブなパスフレーズと一致しない場合、ローテーションが実行されます。
- ボリュームのパスフレーズがシークレットのアクティブなパスフレーズと一致する場合は、を参照してください `previous-luks-passphrase` パラメータは無視されます。

手順

1. を追加します `node-publish-secret-name` および `node-publish-secret-namespace` StorageClassパラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. ボリュームまたはSnapshotの既存のパスフレーズを特定します。

   ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

   スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. ボリュームのLUKSシークレットを更新して、新しいパスフレーズと前のパスフレーズを指定します。確

認します `previous-luke-passphrase-name` および `previous-luks-passphrase` 前のパスフレーズと同じにします。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. ボリュームをマウントする新しいポッドを作成します。これはローテーションを開始するために必要です。
5. パスフレーズがローテーションされたことを確認します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

パスフレーズは、ボリュームとSnapshotに新しいパスフレーズのみが返されたときにローテーションされました。

メモ	たとえば、2つのパスフレーズが返された場合などです `luksPassphraseNames: ["B", "A"]`回転が不完全です。回転を完了するために、新しいポッドをトリガできます。

ボリュームの拡張を有効にします

LUKS暗号化ボリューム上でボリューム拡張を有効にできます。

手順

1. を有効にします `CSINodeExpandSecret` 機能ゲート（ベータ1.25+）。を参照してください "Kubernetes 1.25：CSIボリュームのノードベースの拡張にシークレットを使用します" を参照してください。

2. を追加します `node-expand-secret-name` および `node-expand-secret-namespace` StorageClass パラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

結果

ストレージのオンライン拡張を開始すると、ドライバに適切なクレデンシャルが渡されます。

RWX Linux Unified Key Setup (LUKS) ボリュームのリースが停止した状態を修復する

ReadWriteMany（RWX）Linux Unified Key Setup（LUKS）ボリュームは、ノードがKubernetesリースを保持している間に障害が発生した場合、ブロックされる可能性があります。リースを削除する前に、 `cryptsetup`操作が実行されていないことを確認してください。操作中にリースを削除すると、LUKSヘッダーが危険にさらされる可能性があります。

リースを理解する

TridentはKubernetesリースを使用して、一度に1つのノードのみがRWX LUKSボリュームで `cryptsetup`コマンドを実行するようにします。リースはTridentネームスペースで作成され、リース名はPersistentVolume（PV）名になります。ホルダーIDは、操作を実行しているノードを識別します。

Tridentは、操作が正常に完了したあとにリースを削除します。操作が失敗した場合、Tridentはリースを保持し、同じノードのみが再試行できるようにします。リースは自動的に期限切れになったり更新されたりすることはありません。reason `LeaseRetained`を伴う警告イベントには、失敗した操作に関する詳細が含まれます。

スタックした可能性のあるリースを特定する

症状としては、ポッドが `ContainerCreating`や `Terminating`でスタックする、ボリューム拡張の失敗、または一時的なロックエラーなどが挙げられます。

1. 影響を受けたポッドで使用されているPVCを特定します：

```
kubectl get pod -n <app-namespace> <pod-name> \
```

```
  -o jsonpath='{.spec.volumes[*].persistentVolumeClaim.claimName}'
```

2. PVCにバインドされているPVを確認します：

```
kubectl get pvc -n <app-namespace> <pvc-name> \
  -o jsonpath='{.spec.volumeName}'
```

3. Tridentネームスペース内のリースを一覧表示し、PV名を持つリースを特定します：

```
kubectl get lease -n <trident-namespace>
```

4. 該当するLeaseを、その保持者、経過時間、イベントを含めて確認してください：

```
kubectl describe lease -n <trident-namespace> <pv-name>
```

`cryptsetup`の処理は通常、数秒以内に完了します。5〜10分以上経過したリースは、デッドロックを示している可能性があります。 `LeaseRetained`イベントで繰り返し発生するエラーを確認し、ホルダーノードを特定して、ノードの健常性を確認します：

```
kubectl get nodes
kubectl get pods -A -o wide --field-selector spec.nodeName=<lease-holder-
node>
```

ノードが正常で、リースに警告イベントがない場合は、リースが停止していると判断する前に、ホルダーノード上のTridentノードポッドのログを確認してください：

```
kubectl get pods -n <trident-namespace> -o wide \
  --field-selector spec.nodeName=<lease-holder-node>
kubectl logs -n <trident-namespace> <trident-node-pod> -c trident-main
```

根本的なノードまたはネットワークの問題を解決できる場合は、まずそれを実行してください。Tridentが処理を正常に完了すると、リースは解決する可能性があります。

スタックしたリースを安全に解決します

重要	リースの保持者がまだ `cryptsetup`処理を実行している可能性がある間は、リースを削除しないでください。リースによって調整されていない異なるノードからの同時 `cryptsetup`コマンドは、LUKSヘッダーを破損させる可能性があります。

ノードの状態に合った修復方法を使用してください：

- ホルダーノードに障害が発生し、電源がオフになっている場合： そのノードが保持しているリースを特定

し、影響を受けるリースを削除します。ワークロードが再スケジュールされる前にノードが復旧した場合は、Tridentに処理を完了させ、リースを解放します。ワークロードがすでに移動している場合は、復旧したノード上のポッドが完全に終了するまで待機し、デッドロックが残っていることを確認してから、古いリースを削除します。

- ホルダーノードがコントロールプレーンへの接続を失いました： ノードはまだワークロードを実行している可能性があります。リースを削除する前にノードをシャットダウンすることを推奨します。ノードに引き続きアクセスできる場合は、そのノード上のTridentノードコンテナを停止するか、ログからリースを取得または削除できず、実行されていないことを確認します `cryptsetup`。Tridentは、各操作の前にKubernetes APIにアクセスしてリースを取得する必要があります。接続が失われたあとは、別の操作を開始することはできません。進行中のコマンドが完了するように、リースの作成から少なくとも5分待ってから削除を検討してください。
- ホルダーノードとネットワークは正常です： リース警告イベントとTridentノードログを調べて、失敗した操作を特定します。影響を受けるワークロードを別のノードに移動する必要がある場合は、ホルダーノードを隔離し、影響を受けるアプリケーションポッドを強制削除せずに削除し、終了するまで待ってから、ノードの隔離を解除します。元の操作が実行されなくなったことを確認してから、リースを削除してください。

安全であることを確認したあとにのみ、リースを削除してください：

```
kubectl delete lease -n <trident-namespace> <pv-name>
```

修復後、影響を受けたポッドとボリューム操作を監視してください。リースが再度保持される場合は、リースイベントで報告されたcryptsetupエラーを調査し、根本的な問題を解決してください。

Trident Kerberosによる通信中の暗号化

Kerberos転送中暗号化を使用すると、管理対象クラスタとストレージバックエンドの間のトラフィックの暗号化を有効にすることで、データアクセスセキュリティを強化できます。

Tridentは、ストレージバックエンドとしてONTAPのKerberos暗号化をサポートしています。

- オンプレミス **ONTAP** - Trident は、Red Hat OpenShift および上流の Kubernetes クラスタからオンプレミス ONTAP ボリュームへの NFSv3 および NFSv4 接続での Kerberos 暗号化をサポートします。

作成、削除、サイズ変更、スナップショット、クローン、 読み取り専用のクローンを作成し、NFS暗号化を使用するボリュームをインポートします。

オンプレミスの**ONTAP**ボリュームでの転送中**Kerberos**暗号化の設定

管理対象クラスタとオンプレミスのONTAPストレージバックエンドの間のストレージトラフィックに対してKerberos暗号化を有効にすることができます。

メモ	オンプレミスのONTAPストレージバックエンドを使用するNFSトラフィックのKerberos暗号化は、ストレージドライバを使用した場合にのみサポートされ `ontap-nas`ます。

作業を開始する前に

- ユーティリティにアクセスできることを確認し `tridentctl` ます。

- 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確認しておきます。
- NFSボリュームのKerberos暗号化をサポートするようにONTAP Storage VMを準備しておく必要があります。手順については、を参照してください "データLIFでKerberosを有効にする" 。
- Kerberos 暗号化で使用する NFSv4 ボリュームが正しく構成されていることを確認します。NetApp NFSv4 ドメイン設定セクション（13 ページ） "NetApp NFSv4 の機能強化とベストプラクティスガイド" を参照してください。

ONTAPエクスポートポリシー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 Storage VMのルートボリュームおよびアップストリームのKubernetesクラスタと共有する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新規に作成するエクスポートポリシーでは、次のアクセスプロトコルとアクセス権限がサポートされている必要があります。

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。

詳細を確認

ボリュームのニーズに応じて、次の3つのバージョンのいずれかを設定できます。

- * Kerberos 5 *-（認証と暗号化）
- * Kerberos 5i *-（ID保護による認証と暗号化）
- * Kerberos 5p *-（IDおよびプライバシー保護による認証および暗号化）

適切なアクセス権限を指定してONTAPエクスポートポリシールールを設定します。たとえば、Kerberos 5i暗号化とKerberos 5p暗号化が混在しているNFSボリュームをクラスタにマウントする場合は、次のアクセス設定を使用します。

を入力します	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およびエクスポートポリシールールの作成方法については、次のドキュメントを参照してください。

- "エクスポートポリシーを作成する"
- "エクスポートポリシーにルールを追加する"

ストレージバックエンドの作成

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。

このタスクについて

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場合は、パラメータを使用して次の3つのバージョンのKerberos暗号化のいずれかを指定でき `spec.nfsMountOptions` ます。

- `spec.nfsMountOptions: sec=krb5` （認証と暗号化）
- `spec.nfsMountOptions: sec=krb5i` （ID保護による認証と暗号化）
- `spec.nfsMountOptions: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

Kerberosレベルを1つだけ指定してください。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は、最初のオプションのみが使用されます。

手順

1. 管理対象クラスタで、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<>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は、バックエンドの設定に何か問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs
```

構成ファイルで問題を特定して修正したら、 create コマンドを再度実行できます。

ストレージクラスを作成する。

ストレージクラスを作成して、Kerberos暗号化を使用してボリュームをプロビジョニングできます。

このタスクについて

ストレージクラスオブジェクトを作成するときは、パラメータを使用して、次の3つのバージョンのKerberos暗号化のいずれかを指定できます `mountOptions` 。

- `mountOptions: sec=krb5` （認証と暗号化）
- `mountOptions: sec=krb5i` （ID保護による認証と暗号化）
- `mountOptions: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

Kerberosレベルを1つだけ指定してください。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は、最初のオプションのみが使用されます。ストレージバックエンド構成で指定した暗号化レベルがストレージクラスオブジェクトで指定したレベルと異なる場合は、ストレージクラスオブジェクトが優先されます。

手順

1. 次の例を使用して、StorageClass Kubernetesオブジェクトを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. ストレージクラスを作成します。

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. ストレージクラスが作成されていることを確認します。

```
kubectl get sc ontap-nas-sc
```

次のような出力が表示されます。

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになりました。手順については、を参照してください "ボリュームをプロビジョニングする"。

Azure NetApp Filesボリュームでの転送中Kerberos暗号化の設定

管理対象クラスタと単一のAzure NetApp Filesストレージバックエンドまたはazure NetApp Filesストレージバックエンドの仮想プールの間のストレージトラフィックに対してKerberos暗号化を有効にすることができます。

作業を開始する前に

- 管理対象のRed Hat OpenShiftクラスタでTridentが有効になっていることを確認します。
- ユーティリティにアクセスできることを確認し `tridentctl` ます。
- 要件を確認し、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の準備が完了していることを確認します。 "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 ドメイン設定セクション（13 ページ） "NetApp NFSv4 の機能強化とベストプラクティスガイド" を参照してください。

ストレージバックエンドの作成

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。

このタスクについて

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場合は、次の2つのレベルのいずれかで適用するように定義できます。

- フィールドを使用した* storage backend level * `spec.kerberos`
- フィールドを使用した*仮想プールレベル* `spec.storage.kerberos`

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、次の3つのバージョンのKerberos暗号化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保護による認証と暗号化）
- `kerberos: sec=krb5p` （IDおよびプライバシー保護による認証および暗号化）

手順

1. 管理対象クラスタで、ストレージバックエンドを定義する必要がある場所（ストレージバックエンドレベルまたは仮想プールレベル）に応じて、次のいずれかの例を使用してストレージバックエンド構成ファイルを作成します。括弧<>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は、バックエンドの設定に何か問題があります。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs
```

構成ファイルで問題を特定して修正したら、 create コマンドを再度実行できます。

ストレージクラスを作成する。

ストレージクラスを作成して、Kerberos暗号化を使用してボリュームをプロビジョニングできます。

手順

1. 次の例を使用して、StorageClass Kubernetesオブジェクトを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. ストレージクラスを作成します。

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. ストレージクラスが作成されていることを確認します。

```
kubectl get sc -sc-nfs
```

次のような出力が表示されます。

```
NAME          PROVISIONER             AGE
sc-nfs        csi.trident.netapp.io   15h
```

ボリュームのプロビジョニング

ストレージバックエンドとストレージクラスを作成したら、ボリュームをプロビジョニングできるようになりました。手順については、を参照してください "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 Service（EKS）
- Google Kubernetes Engine （ GKE ）
- Microsoft Azure Kubernetes Service （ AKS ）
- Red Hat OpenShift のサービスです
- SUSE Harvester 1.7.0（ONTAP iSCSI）
- SUSE Rancher
- VMware Tanzuポートフォリオ
- アップストリームKubernetes

メモ	• 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、 "以下の手順を参照して" 。
- 少なくとも1つのVolumeSnapshotClassが存在することを確認してください。詳細については、"ボリュームSnapshotクラス"を参照してください。
- Trident Protect をインストールするには Helm 4.x 以降が必要です。

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。

- NetApp ONTAP 対応の Amazon FSX
- Cloud Volumes ONTAP
- ONTAPストレエシアレイ
- Google Cloud NetAppボリューム
- Azure NetApp Files の特長

ストレージバックエンドが次の要件を満たしていることを確認します。

- クラスタ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。
- バックアップを格納するオブジェクトストレージバケットを設定しておきます。
- アプリケーションまたはアプリケーション データ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。

メモ

一部のアプリケーションは、Snapshotディレクトリを使用するボリュームと互換性がありません。これらのアプリケーションでは、ONTAPストレージシステムで次のコマンドを実行して、snapshotディレクトリを非表示にする必要があります。

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

snapshotディレクトリを有効にするには、NASエコノミーボリュームごとに次のコマンドを実行し、を変更するボリュームのUUIDに置き換え `<volume-UUID>`ます。

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

メモ	新しいボリュームに対してSnapshotディレクトリをデフォルトで有効にするには、Tridentバックエンド構成オプションをに `true`設定し `snapshotDir`ます。既存のボリュームには影響しません。

KubeVirt VMによるデータ保護

Trident Protect は、データ保護操作中に KubeVirt 仮想マシンのファイルシステムのフリーズおよびアンフリーズ機能を提供して、データの一貫性を確保します。 VM フリーズ操作の構成方法とデフォルトの動作はTrident Protect のバージョンによって異なり、新しいリリースでは Helm チャート パラメータによる簡素化された構成が提供されています。

メモ	復元操作中は、 `VirtualMachineSnapshots`仮想マシン (VM) 用に作成されたものは復元されません。

▼ **Trident Protect 25.10**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て一貫性を確保します。 Trident Protect 25.10以降では、 `vm.freeze` Helm チャート インストール時のパラメーター。このパラメータはデフォルトで有効になっています。

```
helm install ... --set vm.freeze=false ...
```

▼ **Trident**プロテクト **24.10.1** から **25.06**

Trident Protect 24.10.1 以降、 Trident Protect はデータ保護操作中に KubeVirt ファイルシステムを自動的にフリーズおよびアンフリーズします。オプションで、次のコマンドを使用してこの自動動作を無効にすることができます。

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

▼ **Trident**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、"仮想化の設定"を使用し、次のコマンドを実行することで、データ保護操作中にVMファイルシステムの凍結と解凍を管理できます：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

Trident Protect を使用してアプリケーションを定義し、VM をリストアする方法については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

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システム。Trident保護を使用したSnapMirrorレプリケーションは、現在ASA r2システムではサポートされていません。
- NetApp ONTAP Select の略
- NetApp Cloud Volumes ONTAP の略
- NetApp ONTAP 対応の Amazon FSX

SnapMirrorレプリケーション用のONTAPクラスタの要件

SnapMirrorレプリケーションを使用する場合は、ONTAPクラスタが次の要件を満たしていることを確認します。

- * 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：Data Protection Bundleを使用するONTAP SnapMirror非同期ライセンスが、ソースとデスティネーションの両方のONTAPクラスタで有効になっている必要があります。詳細については、を参照してください "ONTAP のSnapMirrorライセンスの概要" 。

  ONTAP 9.10.1 以降、すべてのライセンスは、複数の機能を有効にする単一のファイルである NetApp ライセンス ファイル (NLF) として提供されます。詳細については、を参照してください "ONTAP Oneに含まれるライセンス" 。

メモ	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場合は、環境が次の要件を満たしていることを確認してください。

- *クラスタとSVM *：ONTAPストレージバックエンドにピア関係が設定されている必要があります。詳細については、を参照してください "クラスタと SVM のピアリングの概要" 。

メモ	2つのONTAPクラスタ間のレプリケーション関係で使用されるSVM名が一意であることを確認してください。

- **NetApp Trident** 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レプリケーション用の**Trident / ONTAP**の設定

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。

メモ	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 100.2610.0 --create
-namespace --namespace trident-protect
```

3. オプションで、デバッグ ログを有効にするには (トラブルシューティングに推奨)、次のコマンドを使用します。

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2610.0 --create-namespace --namespace trident-protect
```

デバッグ ログにより、 NetAppサポートは、ログ レベルの変更や問題の再現を必要とせずに、問題のトラブルシューティングを行うことができます。

プライベートレジストリから**Trident Protect**をインストールする

Kubernetes クラスタ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。

```
docker.io/netapp/controller:26.10.0
docker.io/netapp/restic:26.10.0
docker.io/netapp/kopia:26.10.0
docker.io/netapp/kopiablockrestore:26.10.0
docker.io/netapp/trident-autosupport:26.10.0
docker.io/netapp/exechook:26.10.0
docker.io/netapp/resourcebackup:26.10.0
docker.io/netapp/resourcerestore:26.10.0
docker.io/netapp/resourcedelete:26.10.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

例：

```
docker pull docker.io/netapp/controller:26.10.0
```

```
docker tag docker.io/netapp/controller:26.10.0 <private-registry-
url>/controller:26.10.0
```

```
docker push <private-registry-url>/controller:26.10.0
```

メモ

Helmチャートを取得するには、まず `helm pull trident-protect --version 100.2610.0 --repo https://netapp.github.io/trident-protect-helm-chart`を使用してインターネットにアクセスできるマシンにHelmチャートをダウンロードし、その結果の `trident-protect-100.2610.0.tgz`ファイルをオフライン環境にコピーして、最後のステップでリポジトリ参照の代わりに `helm install trident-protect ./trident-protect-100.2610.0.tgz`を使用してインストールします。

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。

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

メモ

その `imageRegistry`そして `imagePullSecrets`値は、以下のすべてのコンポーネント画像に適用されます。 `resourcebackup`そして `resourcerestore`。レジストリ内の特定のリポジトリパスにイメージをプッシュする場合（例：`example.com:443/my-repo`）の場合は、レジストリ フィールドに完全なパスを含

めます。これにより、すべての画像が <private-registry-url>/<image-name>:<tag>。

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 100.2610.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

8. オプションで、デバッグ ログを有効にするには (トラブルシューティングに推奨)、次のコマンドを使用します。

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name-of-cluster> --set logLevel=debug --version 100.2610.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

デバッグ ログにより、 NetAppサポートは、ログ レベルの変更や問題の再現を必要とせずに、問題のトラブルシューティングを行うことができます。

メモ

AutoSupport設定や名前空間フィルタリングなどのHelmチャートの追加設定オプションについては、以下を参照してください。 "Trident Protectのインストールをカスタマイズする"。

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、クラスタへのアクセスに使用するマシンにインストールする必要があります。マシンがx64またはARM CPUを使用しているかどうかに応じて、次の手順を実行します。

Linux AMD64 CPU用プラグインのダウンロード

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/v26.10.0/tridentctl-protect-linux-amd64
```

Linux ARM64 CPU用プラグインのダウンロード

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/v26.10.0/tridentctl-protect-linux-arm64
```

Mac AMD64 CPU用プラグインのダウンロード

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/v26.10.0/tridentctl-protect-macos-amd64
```

Mac ARM64 CPU用プラグインのダウンロード

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/v26.10.0/tridentctl-protect-macos-arm64
```

1. プラグインバイナリの実行権限を有効にします。

```
chmod +x tridentctl-protect
```

2. プラグインバイナリをPATH変数で定義されている場所にコピーします。たとえば、 `/usr/bin`または `/usr/local/bin`（昇格されたPrivilegesが必要な場合があります）。

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 必要に応じて、プラグインバイナリをホームディレクトリ内の場所にコピーできます。この場合、location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/
```

メモ	プラグインをPATH変数の場所にコピーすると、任意の場所からまたはを `tridentctl protect`入力してプラグインを使用でき `tridentctl-protect`ます。

Trident CLIプラグインのヘルプを表示

組み込みプラグインヘルプ機能を使用して、プラグインの機能に関する詳細なヘルプを表示できます。

手順

1. ヘルプ機能を使用して、使用方法に関するガイダンスを表示します。

```
tridentctl-protect help
```

コマンドの自動補完を有効にする

Trident 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. 完了スクリプトを作成します。

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

```
mkdir -p ~/.bash/completions
```

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.bash/completions`ます。

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. ホームディレクトリ内のファイルに次の行を追加し `~/.bashrc`ます。

```
source ~/.bash/completions/tridentctl-completion.bash
```

Zシェルの自動補完を有効にする

手順

1. 完了スクリプトを作成します。

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. ホームディレクトリにスクリプトを格納する新しいディレクトリを作成します。

```
mkdir -p ~/.zsh/completions
```

3. ダウンロードしたスクリプトをディレクトリに移動し `~/.zsh/completions`ます。

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. ホームディレクトリ内のファイルに次の行を追加し `~/.zprofile`ます。

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

結果

次のシェルログイン時に、tridentctl-protectプラグインで自動補完コマンドを使用できます。

Trident 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。

   次の構成ファイルの例は、使用可能な設定を示しています。このファイルには、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. ファイルから値を適用し `resourceLimits.yaml`ます。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

セキュリティコンテキスト制約のカスタマイズ

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. SCCオプションをファイルに追加し、環境のニーズに応じてパラメータを変更します。

   次に、SCCオプションのパラメータのデフォルト値の例を示します。

   ```
   scc:
     create: true
     name: trident-protect-job
     priority: 1
   ```

   次の表では、SCCオプションのパラメータについて説明します。

パラメータ	説明	デフォルト
作成	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、がに設定され `true`、Helmのインストールプロセスでの場合にのみ作成されます。`scc.create` OpenShift環境が指定されている `scc.create`。OpenShiftで動作していない場合、またはがに設定されている `false`場合、SCCリソースは作成されません。	正しいです
名前	SCCの名前を指定します。	Trident - protect-job
優先度	SCCのプライオリティを定義します。優先度の高いSCCSは、低い値のSCCSよりも先に評価されます。	1

3. ファイルから値を適用し `sccconfig.yaml`ます。

   ```
   helm upgrade trident-protect -n trident-protect netapp-trident-
   protect/trident-protect -f sccconfig.yaml --reuse-values
   ```

   これにより、デフォルト値がファイルで指定された値に置き換えられ `sccconfig.yaml`ます。

追加の**Trident Protect**ヘルムチャート設定を構成する

AutoSupport 設定、名前空間フィルタリング、および OpenShift Webhook 証明書管理をカスタマイズして、特定の要件を満たすことができます。次の表に、使用可能な設定パラメータを示します：

パラメータ	を入力します	説明
自動サポートプロキシ	文字列	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` 。
autoSupport.insecure	ブール値	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	ブール値	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	文字列	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	文字列	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。
useOpenshiftServiceCA	ブール値	OpenShiftのネイティブ Service CA Operator との統合を有効にし、Trident Protect アドミッション Webhook の TLS 証明書の自動作成と挿入を行います。 `true`に設定すると、`Service CA Operator` が `Webhook TLS` 証明書を自動的に管理します。`OpenShift` クラスターでのみ使用してください。（デフォルト： `false`）

スナップショットとバックアップのメタデータを暗号化するには、"Trident Protectのスナップショットとバックアップ用のAppVaultメタデータを暗号化する"を参照してください。

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"
useOpenshiftServiceCA: true
```

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-string
restoreSkipNamespaceAnnotations="{annotation1,annotation2}" \
  --set-string restoreSkipNamespaceLabels="{label1,label2}" \
  --set useOpenshiftServiceCA=true \
  --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. nodeSelectorオプションをファイルに追加し、ファイルを変更してノードラベルを追加または変更して、環境のニーズに応じて制限します。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対象としています。

```
nodeSelector:
```

```
    kubernetes.io/os: linux
    region: us-west
    app.kubernetes.io/name: mysql
```

3. ファイルから値を適用し `nodeSelectorConfig.yaml`ます。

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

これにより、デフォルトの制限がファイルで指定した制限に置き換えられます `nodeSelectorConfig.yaml`。

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、常にデフォルトのネームスペース内のリソースにアクセスできます `trident-protect`。また、他のすべてのネームスペース内のリソースにもアクセスできます。リソースとアプリケーションへのアクセスを制御するには、追加の名前空間を作成し、それらの名前空間にリソースとアプリケーションを追加する必要があります。

デフォルトの名前空間にアプリケーションデータ管理CRSを作成することはできないことに注意して `trident-protect`ください。アプリケーションデータ管理CRSは、アプリケーションネームスペース内に作成する必要があります（ベストプラクティスとして、アプリケーションデータ管理CRSは、関連付けられているアプリケーションと同じネームスペースに作成します）。

メモ

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 *：バケット資格情報データが必要です。
- **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。

RBACを使用して、権限付きオブジェクトへのアクセスを管理者に制限することを推奨します。

RBACでリソースおよびネームスペースへのアクセスを制御する方法の詳細については、を参照して"Kubernetes RBACのドキュメント"ください。

サービスアカウントの詳細については、を参照して "Kubernetesサービスアカウントのドキュメント"ください。

例：2つのユーザグループのアクセスを管理する

たとえば、ある組織に、クラスタ管理者、エンジニアリングユーザのグループ、およびマーケティングユーザのグループがあるとします。クラスタ管理者は次のタスクを実行して、engineeringグループとmarketingグループがそれぞれのネームスペースに割り当てられたリソースのみにアクセスできる環境を作成します。

手順1：各グループのリソースを含むネームスペースを作成する

ネームスペースを作成すると、リソースを論理的に分離し、それらのリソースにアクセスできるユーザをより細かく制御できます。

手順

1. engineeringグループの名前空間を作成します。

```
kubectl create ns engineering-ns
```

2. marketingグループの名前空間を作成します。

```
kubectl create ns marketing-ns
```

ステップ2：各ネームスペースのリソースとやり取りするための新しいサービスアカウントを作成する

作成する新しい名前空間にはそれぞれデフォルトのサービスアカウントが付属していますが、将来必要に応じてPrivilegesをグループ間でさらに分割できるように、ユーザーのグループごとにサービスアカウントを作成する必要があります。

手順

1. engineeringグループのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. マーケティンググループのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
```

```
  namespace: marketing-ns
```

ステップ**3**：新しいサービスアカウントごとにシークレットを作成する

サービスアカウントシークレットは、サービスアカウントでの認証に使用され、侵害された場合は簡単に削除および再作成できます。

手順

1. エンジニアリングサービスアカウントのシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. マーケティングサービスアカウントのシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

手順**4**：**RoleBinding**オブジェクトを作成して、**ClusterRole**オブジェクトを新しい各サービスアカウントにバインドする

Trident 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
```

手順**5**：権限のテスト

権限が正しいことをテストします。

手順

1. エンジニアリングユーザーがエンジニアリングリソースにアクセスできることを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. エンジニアリングユーザーがマーケティングリソースにアクセスできないことを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

手順**6**：**AppVault**オブジェクトへのアクセスを許可する

バックアップやスナップショットなどのデータ管理タスクを実行するには、クラスタ管理者が個 々 のユーザーにAppVaultオブジェクトへのアクセスを許可する必要があります。

手順

1. AppVaultへのユーザーアクセスを許可するAppVaultとシークレットの組み合わせYAMLファイルを作成して適用します。たとえば、次のCRは、AppVaultへのアクセスをユーザーに許可し `eng-user`ます。

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 役割CRを作成して適用し、クラスタ管理者がネームスペース内の特定のリソースへのアクセスを許可で

きるようにします。例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. RoleBinding CRを作成して適用し、権限をeng-userというユーザにバインドします。例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 権限が正しいことを確認します。

   a. すべての名前空間のAppVaultオブジェクト情報の取得を試みます。

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   次のような出力が表示されます。

   ```
   Error from server (Forbidden): appvaults.protect.trident.netapp.io is
   ```

```
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. ユーザがAppVault情報を取得できるかどうかをテストして、アクセス許可を得ているかどうかを確認します。

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

次のような出力が表示されます。

```
yes
```

結果

AppVault権限を付与したユーザーは、アプリケーションデータ管理操作に承認されたAppVaultオブジェクトを使用できる必要があります。また、割り当てられた名前空間以外のリソースにアクセスしたり、アクセスできない新しいリソースを作成したりすることはできません。

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などのツールから送信されたアラートを取り込み、設定した送信先にルーティングするサービスです。

メモ

これらの手順に記載されている構成とガイダンスは一例にすぎません。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ノトキユメント"
- "AlertManagerのドキュメント"

手順**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。

インストール**kube-state-metrics**

kube-state-metricsはHelmを使用してインストールできます。

手順

1. kube-state-metrics Helmチャートを追加します。例：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. Prometheus ServiceMonitor CRD をクラスターに適用します。

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. Helmチャートの構成ファイルを作成します（例： `metrics-config.yaml`）。次の設定例は、環境に合わせてカスタマイズできます。

metrics-config.yaml：kube-state-metrics Helmチャート構成

```
---
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チャートを展開してkube-state-metricsをインストールします。例：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. kube-state-metrics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カスタムリソースドキュメント" 。

Prometheus をインストールする

の手順に従ってPrometheusをインストールできます。 "Prometheusノトキユメント"

AlertManagerのインストール

の手順に従って、AlertManagerをインストールできます "AlertManagerのドキュメント"。

ステップ2：監視ツールが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ら、それらが連携するように設定する必要があります。

手順

1. kube-state-metricsとPrometheusを統合Prometheus構成ファイル(`prometheus.yaml`を編集）、kube-state-metricsサービス情報を追加します。例：

prometheus.yaml: kube-state-metrics サービスと Prometheus の統合

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: trident-protect
data:
  prometheus.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.trident-protect.svc:8080']
```

2. アラートをAlertManagerにルーティングするようにPrometheusを設定します。Prometheus構成ファイル(`prometheus.yaml`を編集）、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では、kube-state-metricsから指標を収集し、アラートをAlertmanagerに送信できるようになりました。これで、アラートをトリガーする条件とアラートの送信先を設定する準備ができました。

手順**3**：アラートとアラートの送信先を設定する

ツールが連携して動作するように設定したら、アラートをトリガーする情報の種類とアラートの送信先を設定する必要があります。

アラートの例：バックアップの失敗

次の例は、バックアップカスタムリソースのステータスが5秒以上に設定された場合にトリガーされるCriticalアラートを定義します `Error`。この例を環境に合わせてカスタマイズし、このYAMLスニペットを構成ファイルに含めることができます `prometheus.yaml`。

rules.yaml: 失敗したバックアップに関する **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."
```

アラートを他のチャネルに送信するように**AlertManager**を設定する

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AlertManagerを設定するには、ファイルでそれぞれの設定を指定し `alertmanager.yaml`ます。

次の例では、Slackチャンネルに通知を送信するようにAlertManagerを設定します。この例を環境に合わせてカスタマイズするには、キーの値を環境で使用されているSlack Webhook URLに置き換え `api_url`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を使用したサポートバンドルの作成

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-`

support-bundle.yaml）。

2. 次の属性を設定します。

   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.triggerType *：（*required*）サポートバンドルをすぐに生成するかスケジュールするかを指定します。スケジュールされたバンドル生成は12AM UTCに行われます。有効な値：
     - スケジュール済み
     - 手動
   - * spec.uploadEnabled *：（_Optional _）サポートバンドルの生成後にNetAppサポートサイトにアップロードするかどうかを制御します。指定しない場合、デフォルトはになります `false`。有効な値：
     - 正しいです
     - false（デフォルト）
   - **spec.dataWindowStart**:(*Optional*)サポートバンドルに含まれるデータのウィンドウを開始する日時を指定する、RFC 3339形式の日付文字列。指定しない場合は、デフォルトで24時間前になります。指定できる最も早い期間の日付は7日前です。

     YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. 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を使用したサポートバンドルの作成

手順

1. サポートバンドルを作成し、角かっこ内の値を環境からの情報に置き換えます。は `trigger-type`、バンドルをすぐに作成するか、スケジュールによって作成時間が指定されているかを決定し、または `Scheduled`を指定できます `Manual`。デフォルト設定はです `Manual`。

   例：

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -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。

メモ

- バージョン24.10からアップグレードする場合、アップグレード中に実行されているスナップショットが失敗する可能性があります。この失敗によって、手動またはスケジュールされたスナップショットの今後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、アプリケーションを保護するために、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、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ます。

- プライベート レジストリのインストールでは、ターゲット バージョンに必要な Helm チャートとイメージがプライベート レジストリで使用できることを確認し、カスタム Helm 値が新しいチャート バージョンと互換性があることを確認します。詳細については、"プライベートレジストリからTrident Protectをインストールする" 。

ステップ**1**：バージョンを選択します

Trident Protectのバージョンは日付ベースの `YY.MM`命名規則に従っており、「YY」は年の最後の2桁、「MM」は月を表します。ドットリリースは `YY.MM.X`規則に従っており、「X」はパッチレベルを表します。アップグレード元のバージョンに基づいて、アップグレード先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、24.10（または任意の24.10ドットリリース）から25.10に直接アップグレードできます。
- 4つの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、複数ステップのアップグレードを実行してください。"以前のバージョン"からアップグレードする場合は、そのバージョンのアップグレード手順を使用して、4リリース期間に収まる最新のリリースにアップグレードします。例えば、24.10 を実行していて、26.10 にアップグレードしたい場合：
  a. 最初に 24.10 から 25.06 にアップグレードします。
  b. 次に、25.06から26.10にアップグレードします。

ステップ**2**：**Trident Protect**のアップグレード

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。

メモ	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：

メモ	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 100.2610.0 --namespace trident-protect
```

メモ

アップグレード中にログレベルを設定するには、 `--set logLevel=debug`アップグレード コマンドに追加します。デフォルトのログレベルは `warn。デバッグ ログは、ログレベルの変更や問題の再現を必要とせずにNetAppサポートが問題を診断するのに役立つため、トラブルシューティングには推奨されます。

アプリケーションの管理と保護

Trident Protect AppVault オブジェクトを使用してバケットを管理する

Trident Protect のバケット カスタム リソース (CR) は、AppVault と呼ばれます。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。

メモ

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認証とパスワードの設定

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。

Data Mover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。

メモ

スナップショットとバックアップのメタデータを暗号化したり、保管庫のパスワードを変更したりするには、 `spec.vaultPasswordSecretRef`フィールドを使用します。"Trident Protectのス

ナップショットとバックアップ用のAppVaultメタデータを暗号化する"を参照してください。

Data Moverリポジトリパスワードシークレットの作成

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。

メモ

- 使用しているData Moverに応じて、そのData Moverに対応するパスワードだけを含める必要があります。たとえば、Resticを使用していて、今後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の使用

```
---
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 を使用します

```
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、 "StorageGRID 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の**AppVault**キー生成例

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
```

StorageGRID S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<storagegrid-s3-trident-protect-src
-bucket-secret> \
-n trident-protect
```

AppVaultの作成例

各プロバイダのAppVault定義の例を次に示します。

AppVault CRの例

次のCR例を使用して、クラウドプロバイダごとにAppVaultオブジェクトを作成できます。

メモ

- 必要に応じて、ResticおよびKopiaリポジトリ暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。詳細については、を参照してください Data Moverリポジトリのパスワード 。
- Amazon S3（AWS）AppVaultオブジェクトの場合、必要に応じて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、でプロバイダのキーを生成するときに作成されクラウドプロバイダのAppVaultキー生成例ます。
- S3 AppVaultオブジェクトの場合、必要に応じて、キーを使用して発信S3トラフィックの出力プロキシURLを指定できます `spec.providerConfig.S3.proxyURL`。

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
```

メモ	KopiaデータムーバーでPod Identityを使用するEKS環境では、 `providerCredentials`セクションを追加して `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

```
apiVersion: protect.trident.netapp.io/v1
```

```
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGRID S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
```

```
        name: s3-secret
```

Trident Protect CLI を使用した AppVault 作成例

次のCLIコマンド例を使用して、プロバイダごとにAppVault CRSを作成できます。

メモ

- 必要に応じて、ResticおよびKopiaリポジトリ暗号化用のカスタムパスワードを含むKubernetesシークレットを指定できます。詳細については、を参照してください Data Moverリポジトリのパスワード 。
- S3 AppVaultオブジェクトの場合は、引数を使用して送信S3トラフィックの出力プロキシURLをオプションで指定できます `--proxy-url <ip_address:port>`。

Google Cloud

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Amazon S3（AWS）

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

汎用 S3

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
```

```
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

StorageGRID S3

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

サポートされている `**providerConfig.s3**`設定オプション

S3 プロバイダー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	例
`providerConfig.s3.skipCertValidation`	SSL/TLS 証明書の検証を無効にします。	いいえ	「真」、「偽」
`providerConfig.s3.secure`	S3 エンドポイントとの安全な HTTPS 通信を有効にします。	正しいです	「真」、「偽」
`providerConfig.s3.proxyURL`	S3 への接続に使用するプロキシ サーバーの URL を指定します。	なし	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	SSL/TLS 検証用のカスタム ルート CA 証明書を提供します。	なし	「CN=MyCustomCA」
`providerConfig.s3.useIAM`	S3 バケットにアクセスするための IAM 認証を有効にします。 EKS ポッド ID に適用可能です。	いいえ	true false

AppVault情報の表示

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. 必要に応じて、各リソースのAppVaultPathを表示するには、フラグを使用し `--show-paths`ます。

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の削除

AppVaultオブジェクトはいつでも削除できます。

メモ	AppVaultオブジェクトを削除する前に、AppVault CRのキーを削除しないで `finalizers`ください。これを行うと、AppVaultバケット内のデータが残り、クラスタ内のリソースが孤立する可能性があります。

作業を開始する前に

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CRSが削除されていることを確認します。

Kubernetes CLIを使用したAppVaultの削除

1. AppVaultオブジェクトを削除し、削除するAppVaultオブジェクトの名前に置き換え `appvault-name` ます。

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

Trident 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、削除するAppVaultオブジェクトの名前に置き換え `appvault-name` ます。

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

Trident Protectのスナップショットとバックアップ用のAppVaultメタデータを暗号化する

Trident ProtectがAppVaultに保存するスナップショットおよびバックアップのメタデータ（Kubernetes Secretsの機密情報を含む）を暗号化できます。暗号化されたデータを保護するパスワードを変更することもできます。パスワードを1回変更するだけで、メタデータの暗号化キーが再暗号化され、AppVault内のすべてのKopiaリポジトリとすべてのResticリポジトリが新しいパスワードに移行されます。

暗号化はデフォルトでは無効になっています。有効にすると、Trident Protectは、メタデータをAppVaultに保存する前に、新しいスナップショットとバックアップのメタデータを暗号化します。これにより、アプリケーションのスナップショットやバックアップからこれらのリソースを明示的に除外しない限り、Secrets内の機密情報を含むアプリケーションリソースが保護されます。

保管庫のパスワードは、データを暗号化するキーを保護します。以下の動作に注意してください：

- vaultのパスワードを変更すると、Trident Protectは同じキーを新しいパスワードで保護します。すでに暗号化されているデータは書き換えません。

- 暗号化を有効にする前に保存されたデータは、引き続き読み取り可能です。
- 暗号化されたデータは、後で新しいデータの暗号化を無効にした場合でも、読み取り可能な状態のままです。

このタスクについて

暗号化を有効にすると、Trident Protectは、カスタムパスワードを指定しない限り、安全性の低いデフォルトパスワードを使用します。各AppVaultについて、カスタムvaultパスワードを作成し、`spec.vaultPasswordSecretRef`で参照します。

デフォルトのパスワードで暗号化を有効にした後でカスタムパスワードを追加した場合は、既存のデータとリポジトリがカスタムパスワードに移行するように、パスワードを変更する必要があります。Vaultパスワードをローテーションする を参照してください。

AppVaultごとに固有のカスタムパスワードを使用してください。一度に1つのAppVaultのパスワードをローテーションし、ローテーションが完了したことを確認してから、別のパスワード変更を開始してください。

作業を開始する前に

以下が用意されていることを確認してください：

- `trident-protect`ネームスペースへのアクセス。
- Kubernetes Secretsを作成し、AppVaultカスタムリソース（CR）にパッチを適用する権限。
- 各AppVault用に個別のパスワードシークレットが計画されています。

メタデータ暗号化を有効にする

Trident ProtectのHelm値を設定して、メタデータ暗号化を有効にします。

手順

1. `values.yaml` ファイルを使用して `resourceBackupEncryptionEnabled` を `true` に設定：
   a. 次の内容を `values.yaml`ファイルに追加します：

```
resourceBackupEncryptionEnabled: true
```

   b. 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
```

2. または、Helm `--set`フラグを使用して `resourceBackupEncryptionEnabled`を `true`に設定します：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set resourceBackupEncryptionEnabled=true \
  --reuse-values
```

結果

Trident Protectは、新しいスナップショットとバックアップのメタデータを暗号化します。

次の手順

各AppVaultにカスタムのvaultパスワードを作成します。作成するまで、Trident Protectは安全性の低いデフォルトパスワードを使用します。

カスタム保管庫パスワードを作成する

独自のパスワードを使用するには、パスワードを含む Kubernetes Secret を作成し、AppVault でその Secret を参照します。

このタスクについて

AppVaultの作成時にカスタムパスワードを設定した場合、データが保存される前に、AppVaultはカスタムパスワードを即座に使用します。パスワードをローテーションする必要はありません。

AppVaultにすでにKopiaまたはResticリポジトリがある場合、新しいパスワードを追加しても、それらのリポジトリのパスワードは変更されません。このセクションの手順は使用しないでください。代わりに、Vaultパスワードをローテーションするの手順に従ってください。

`VAULT_PASSWORD`の値は次の要件を満たす必要があります：

- 空であってはなりません
- 64 KiB未満である必要があります
- NUL文字を含めることはできません
- 改行を含めてはいけません
- スペースまたはその他の空白文字で開始または終了してはいけません

パスワードには、ASCII以外の文字を含む、その他の文字を含めることができます。

手順

1. AppVaultと同じネームスペースにシークレットを作成します。シークレットには、 `VAULT_PASSWORD` という名前のキーが含まれている必要があります：

```
kubectl -n trident-protect create secret generic vault-password \
  --from-literal=VAULT_PASSWORD='<password>'
```

メモ

AppVaultがパスワードが無効であると報告した場合は、 `VAULT_PASSWORD`の値を上記の要件と照らし合わせて確認してください。よくある原因としては、値が空である、末尾に改行がある、パスワードの先頭または末尾にスペースが入っているなどが挙げられます。

2. AppVault CRの `spec.vaultPasswordSecretRef` フィールドでシークレットを参照します：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: my-vault
  namespace: trident-protect
spec:
  vaultPasswordSecretRef: vault-password
  providerType: <provider-type>
  providerConfig:
    <provider-configuration>
```

`spec.vaultPasswordSecretRef` が空の場合、AppVault は安全性の低いデフォルトのパスワードを使用します。

結果

AppVaultは、メタデータの暗号化キーを保護するためにカスタムパスワードを使用します。

Vaultパスワードをローテーションする

すでにデータを保存しているAppVaultのパスワードをローテーションできます。1回のローテーションでメタデータの暗号化キーを再暗号化し、AppVault内のすべてのKopiaリポジトリとすべてのResticリポジトリを新しいパスワードに移行します。

このタスクについて

ローテーション中、Trident Protectは次の処理を実行します：

- 開始する前に、AppVault の実行中のバックアップおよびリストアジョブが完了するのを待機します。
- 新しいパスワードでメタデータ暗号化キー(`metadata-key.json`）を保護します。アーカイブオブジェクトは書き換えません。
- AppVaultプレフィックスの下にある各Kopiaリポジトリと各Resticリポジトリを検索し、パスワードを新しいパスワードに変更します。既に新しいパスワードで開いているリポジトリは変更されません。
- バックアップまたはリストアジョブがまだ使用しているリポジトリをスキップし、後で再試行します。
- AppVaultを参照する有効なスケジュールを一時停止します。各スケジュールについて、Trident Protectは `spec.enabled`を `false`に設定し、次のアノテーションを追加します：

```
protect.trident.netapp.io/vault-password-rotation-suspended:
<namespace>/<appvault-name>
```

ローテーションが完了すると、Trident Protectはこのアノテーションが付いているスケジュールのみを再び有効にします。ローテーションが失敗した場合は、停止した場所から再開されます。すでに新しいパスワードを使用しているリポジトリは、再度変更されません。

作業を開始する前に

以下が用意されていることを確認してください：

- Vaultパスワード要件を満たす新しいパスワード。
- 他のAppVaultはパスワードをローテーションしていません。複数のAppVaultsが同じバケットとプレフィックスを共有している場合、パスワードをローテーションできるのはそのうちの1つだけです。バケットを共有するAppVaultsのローテーションを管理するを参照してください。

手順

1. 現在のシークレットとは異なる名前で新しいシークレットを作成します：

```
kubectl -n trident-protect create secret generic vault-password-new \
  --from-literal=VAULT_PASSWORD='<new-password>'
```

> 重要
>
> ローテーションが終了する前に、現在のシークレットを編集または削除しないでください。Trident Protectは、すべてのリポジトリが新しいパスワードを使用するようになるまで、既存のデータを開くために現在のパスワードを必要とします。現在のシークレットを早すぎるタイミングで削除した場合は、同じ名前と元の `VAULT_PASSWORD`値で再作成してください。

2. ローテーションを有効にし、新しいシークレットを使用するようにAppVaultを設定します。 `my-vault`をAppVaultの名前に置き換えます：

```
kubectl -n trident-protect patch appvault my-vault --type merge --patch
'
spec:
  enableRotation: true
  vaultPasswordSecretRef: vault-password-new
'
```

> メモ
>
> パスワードを変更する前に、 `spec.enableRotation`を `true`に設定する必要があります。 `false`の場合、AppVaultはローテーションが無効であることを報告し、パスワードを変更しません。

3. [パスワードのローテーションを監視する] の手順に従ってローテーションを監視します。
4. ローテーションが完了したら、他に使用されていない限り、以前のシークレットを削除します。

結果

メタデータ暗号化キーとAppVault内のすべてのKopiaおよびResticリポジトリは新しいパスワードで保護され、Trident Protectが一時停止したスケジュールが再び有効になります。

パスワードのローテーションを監視する

パスワードのローテーションの進行状況を確認し、完了したことを確認できます。

手順

1. AppVaultを表示：

```
kubectl -n trident-protect get appvault my-vault --output yaml
```

2. 出力で、 `status.vaultPasswordRotation`セクションを確認します。ローテーションの進行中は、次のようになります：

```
status:
  vaultPasswordRotation:
    state: Rotating
    rotated: 3
    pending: 1
    message: <message>
```

この表は、各項目について説明しています：

フィールド	説明
status.vaultPasswordRotation.state	現在の回転状態。
status.vaultPasswordRotation.rotated	新しいパスワードを使用しているリポジトリの数。
status.vaultPasswordRotation.pending	バックアップまたはリストアで使用されているためスキップされたリポジトリの数。
status.vaultPasswordRotation.message	最新の進捗状況または失敗メッセージ。

結果

ローテーションは、以下のすべての条件が満たされたときに完了します：

- `status.vaultPasswordRotation` は存在しません。
- `status.previousRepositoryPasswordSecretRef`は空です。
- `status.repositoryPasswordSecretRef` が一致します `spec.vaultPasswordSecretRef`。

次の手順

ローテーションが完了しない場合は、以下を確認してください：

- ローテーションが開始されません： `spec.enableRotation`が `false`になっている可能性があります。これを `true`に設定し、 `spec.vaultPasswordSecretRef`を新しいSecretに設定します。
- リポジトリが保留状態のままです：バックアップまたはリストアジョブがまだそれらを使用しています。ジョブが完了するまでお待ちください。Trident Protectは保留中のリポジトリを自動的に再試行します。
- 別の**AppVault**がロックを所有しているため、ローテーションが開始されません：バケットを共有するAppVaultsのローテーションを管理するを参照してください。
- メッセージ欄にエラーが報告されています：メッセージに記載されている問題を解決してください。ローテーションは停止した場所から再開されます。

dataMoverPasswordSecretRefからの移行

既存のAppVaultが古い `spec.dataMoverPasswordSecretRef`フィールドを使用している場合は、`spec.vaultPasswordSecretRef`に移動できます。

このタスクについて

Trident Protectは、後方互換性のために `spec.dataMoverPasswordSecretRef`を維持します。新しいAppVaultsには `spec.vaultPasswordSecretRef`を使用してください。

手順

1. `spec.dataMoverPasswordSecretRef`とそのSecretをそのままにしておきます。
2. `spec.vaultPasswordSecretRef`を使用して、パスワードを新しいシークレットにローテーションします。Vaultパスワードをローテーションするの手順に従います。
3. ローテーションが完了したら、AppVaultから `spec.dataMoverPasswordSecretRef`を削除します。
4. 他に使用されていない古いシークレットは削除してください。

バケットを共有するAppVaultsのローテーションを管理する

複数のAppVaultsが同じバケットとプレフィックスを使用する場合、パスワードをローテーションできるのはそのうちの1つだけです。

このタスクについて

パスワードのローテーションには、 `password-rotation.lock`という名前のロックファイルが使用されます。ロックには、バケットとプレフィックスのパスワードのローテーションを許可されたAppVaultのUIDが記録されます。2つ目のAppVaultに `spec.enableRotation: true`があり、ロックで別のAppVaultが指定されている場合、2つ目のAppVaultはローテーションできません。

例 **1.** 手順

ローテーションを別のAppVaultに転送する

1. ロックを所有しているAppVaultで、クリアします `spec.enableRotation`：

```
kubectl -n trident-protect patch appvault <current-owner> --type
merge --patch '
spec:
  enableRotation: false
'
```

2. ロックが解除されるまでお待ちください。
3. 次にローテーションするAppVaultで、 `spec.enableRotation`を `true`に設定します：

```
kubectl -n trident-protect patch appvault <new-owner> --type merge
--patch '
spec:
  enableRotation: true
```

```
'
```

削除された**AppVault**が残したロックを解除する

クラッシュ後など、ロックを所有しているAppVaultがすでに存在せず、ロックが解除されなかった場合は、次の手順を実行します。

1. バケット内のAppVaultプレフィックスから `password-rotation.lock`ファイルを削除します。
2. ローテーションするAppVaultで、 `spec.enableRotation`を `true`に設定します：

```
kubectl -n trident-protect patch appvault <appvault-name> --type
merge --patch '
spec:
  enableRotation: true
'
```

関連情報

- "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のインストールをカスタマイズする"
- "アプリケーションの保護"
- "Trident Protectを使用してアプリケーションを復元する"

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の作成

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。"

アプリケーションの定義

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。

メモ	名前空間全体ではなく名前で特定のKubeVirt仮想マシン（VM）を保護するVMスコープのアプリケーションを定義するには、Application CRの `spec.includedVirtualMachines`フィールドまたはCLIの `--virtual-machines`フラグを使用します。詳細、フィールドリファレンス、およびYAMLの例については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

CRを使用したアプリケーションの追加

手順

1. デスティネーションアプリケーションのCRファイルを作成します。
   a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `maria-app.yaml` ）。
   b. 次の属性を設定します。
      - * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。保護操作に必要な他のCRファイルがこの値を参照するため、選択した名前をメモします。
      - * spec.includedNamespaces*:(*required*)名前空間とラベルセレクタを使用して、アプリケーションが使用する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に含まれている必要があります。ラベルセレクタはオプションで、指定した各名前空間内のリソースをフィルタリングするために使用できます。
      - * spec.includedClusterScopedResources*:（_Optional _）この属性を使用して、アプリケーション定義に含めるクラスタスコープ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、およびラベルに基づいてこれらのリソースを選択できます。
        - *groupVersionKind *:(*required*)クラスタスコープリソースのAPIグループ、バージョン、および種類を指定します。
        - *labelSelector *:(*Optional*)ラベルに基づいてクラスタスコープリソースをフィルタリングします。
      - **spec.volumeGroups**：（オプション）この属性を使用して、関連するボリュームをまとめて「crash consistent state（障害など予期しないシャットダウン時と同様）」の状態のボリュームグループスナップショットとしてスナップショットを作成します。詳細については、"Trident Protectでボリュームグループのスナップショットを使用する"を参照してください。
      - **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"。

アプリケーションの作成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します。

```
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）必要に応じて、同じCRにリソースフィルタリングを追加して、特定のリソースを含めるか除外することができます：

   - 一般的なフィルタの例：

     - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) resourceMatchersで定義されたリソースを使用 `Include`または `Exclude`除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。

       - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
       - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
       - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
       - * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。

- *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

メモ	両方が `resourceFilter`そして `labelSelector`使用される、 `resourceFilter` 最初に実行し、次に `labelSelector`結果のリソースに適用されます。

例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

- **PVC**のみのフィルターの例：

  PVCのみのアプリケーションを定義するには、 `PersistentVolume`と `VolumeSnapshotClass`もリソースフィルターに含める必要があります。スナップショットとバックアップの処理は、 `PersistentVolume`（各PVCにバインドされたクラスタスコープのボリューム）と `VolumeSnapshotClass`（スナップショットドライバ）に依存しており、これらがないと失敗します。例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-pvc-app
  namespace: my-app-namespace
spec:
  includedNamespaces:
  - namespace: my-app-namespace
  resourceFilter:
    resourceMatchers:
```

```
    - kind: PersistentVolumeClaim
      version: v1
    - kind: PersistentVolume
      version: v1
    - kind: VolumeSnapshotClass
      version: v1
  resourceSelectionCriteria: Include
```

2. 環境に合わせてアプリケーションCRを作成したら、CRを適用します。例：

```
kubectl apply -f maria-app.yaml
```

手順

1. 次のいずれかの例を使用して、アプリケーション定義を作成して適用します。括弧内の値は、環境の情報に置き換えます。アプリケーション定義に名前空間とリソースを含めるには、例に示す引数をカンマで区切ったリストを使用します。

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際に注釈を指定するには、 `--annotation` フラグを使用できます。

+ * アプリケーションを作成し、ファイルシステムのフリーズ動作に関する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に関するローカルアプリケーション設定を構成します：

+

```
tridentctl-protect create application <my_new_app_cr_name> --namespaces
<namespaces_to_include> --csr <cluster_scoped_resources_to_include>
--namespace <my-app-namespace> --annotation
```

```
protect.trident.netapp.io/skip-vm-freeze=<"true"|"false">
```

+ * `--resource-filter-include`および `--resource-filter-exclude`フラグを使用して、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、名前空間などの `resourceSelectionCriteria`に基づいてリソースを含めたり除外したりすることができます。

+

```
tridentctl-protect create application <my_new_app_cr_name> --namespaces
<namespaces_to_include> --csr <cluster_scoped_resources_to_include>
--namespace <my-app-namespace> --resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my-
deployment"],"Namespaces":["my-namespace"],"LabelSelectors":["app=my-
app"]}]'
```

+ * PVCのみのアプリケーションを定義するには、リソースフィルターに `PersistentVolume`および `VolumeSnapshotClass``も含める必要があります。スナップショットとバックアップの操作は、`PersistentVolume`（各PVCにバインドされたクラススコープのボリューム）および `VolumeSnapshotClass`（スナップショットドライバ）に依存しており、これらがないと失敗します。例：

+

```
tridentctl-protect create app my-pvc-app --namespaces <my-app-
namespace> --resource-filter-include
'[{"Kind":"PersistentVolumeClaim","Version":"v1"},{"Kind":"PersistentVo
lume","Version":"v1"},{"Kind":"VolumeSnapshotClass","Version":"v1"}]'
-n <my-app-namespace>
```

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。

メモ

- 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"。
- アプリケーションの関連ボリュームを同じ時点のスナップショットとして取得するには、アプリケーション内でボリュームグループを定義します。"Trident Protectでボリュームグループのスナップショットを使用する"を参照してください。

オンデマンドスナップショットを作成します

オンデマンド Snapshot はいつでも作成できます。

メモ	クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CRを使用したスナップショットの作成

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.applicationRef *：スナップショットを作成するアプリケーションのKubernetes名。
   - * spec.appVaultRef *：（*required*）スナップショットの内容（メタデータ）を格納するAppVaultの名前。
   - * spec.reclaimPolicy *：（_Optional _）スナップショットCRが削除されたときのスナップショットのAppArchiveの動作を定義します。つまり、に設定しても `Retain`Snapshotは削除されます。有効なオプション：
     - `Retain` （デフォルト）
     - `Delete`

```
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  reclaimPolicy: Delete
```

3. ファイルに正しい値を入力したら `trident-protect-snapshot-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-cr.yaml
```

CLIを使用したスナップショットの作成

手順

1. スナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

オンデマンドバックアップの作成

アプリはいつでもバックアップできます。

メモ	クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

作業を開始する前に

長時間実行されるs3バックアップ処理には、AWSセッショントークンの有効期限が十分であることを確認してください。バックアップ処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

CRを使用したバックアップの作成

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.applicationRef *：（*required*）バックアップするアプリケーションのKubernetes名。
   - * spec.appVaultRef *：（*required*）バックアップ内容を格納するAppVaultの名前。
   - *spec.DataMover *:(*Optional*)バックアップ操作に使用するバックアップツールを示す文字列。有効な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia` （デフォルト）
   - * spec.reclaimPolicy *：（_Optional _）要求から解放されたバックアップの処理を定義します。有効な値：
     - `Delete`
     - `Retain` （デフォルト）
   - **spec.snapshotRef**: (オプション): バックアップのソースとして使用するスナップショットの名前。指定しない場合は、一時Snapshotが作成されてバックアップされます。

YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. ファイルに正しい値を入力したら `trident-protect-backup-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-cr.yaml
```

CLIを使用したバックアップの作成

手順

1. バックアップを作成します。角かっこ内の値は、使用している環境の情報に置き換えます。例：

```
tridentctl-protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up> --data-mover
<Kopia_or_Restic> -n <application_namespace>
```

オプションで、フラグを使用して、バックアップを非増分にするかどうかを指定できます `--full-backup`。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です。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。リストアに伴うリスクを最小限に抑えるために、フルバックアップを定期的に実行してから、フルバックアップの間に増分バックアップを実行することを推奨します。

サポートされているバックアップ注釈

次の表は、バックアップ CR を作成するときに使用できる注釈を示しています。

アノテーション	を入力します	説明	デフォルト値
protect.trident.netapp.io/フルバックアップ	文字列	バックアップを非増分にするかどうかを指定します。設定 `true`非増分バックアップを作成します。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。	いいえ

アノテーション	を入力します	説明	デフォルト値
protect.trident.netapp.io/スナップショット完了タイムアウト	文字列	スナップショット操作全体が完了するまでに許容される最大時間。	「60メートル」
protect.trident.netapp.io/ボリュームスナップショットの使用準備完了のタイムアウト	文字列	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	「30メートル」
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	文字列	ボリューム スナップショットの作成に許可される最大時間。	「5メートル」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。	「1200」（20分）

データ保護スケジュールを作成

保護ポリシーは、定義されたスケジュールでスナップショット、バックアップ、またはその両方を作成すること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。

メモ

- 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。
- クラスタ対象のリソースが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ネームスペースへの参照がある場合、バックアップ、Snapshot、またはクローンに含まれます。

CRを使用したスケジュールの作成

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-schedule-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.DataMover *:(*Optional*)バックアップ操作に使用するバックアップツールを示す文字列。有効な値（大文字と小文字が区別されます）：
     - `Restic`
     - `Kopia` （デフォルト）
   - * spec.applicationRef *：バックアップするアプリケーションのKubernetes名。
   - * spec.appVaultRef *：（*required*）バックアップ内容を格納するAppVaultの名前。

- **spec.backupRetention**: (必須) 保持するバックアップの数。ゼロは、バックアップを作成しないことを示します (スナップショットのみ)。
- **spec.backupReclaimPolicy**: (オプション) バックアップ CR が保持期間中に削除された場合にバックアップに何が起こるかを決定します。保持期間が過ぎると、バックアップは常に削除されます。可能な値（大文字と小文字が区別されます）：
  - `Retain` （デフォルト）
  - `Delete`
- **spec.snapshotRetention**: (必須) 保持するスナップショットの数。ゼロはスナップショットを作成しないことを示します。
- **spec.snapshotReclaimPolicy**: (オプション) 保持期間中にスナップショット CR が削除された場合にスナップショットに何が起こるかを決定します。保持期間が過ぎると、スナップショットは常に削除されます。可能な値（大文字と小文字が区別されます）：
  - `Retain`
  - `Delete`（デフォルト）
- * spec.granularity*:スケジュールを実行する頻度。指定可能な値と必須の関連フィールドは次のとおりです。
  - `Hourly`（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。
- **spec.runImmediately**：（オプション） `true`に設定すると、スケジュールの作成時に1回限りの即時ベースライン実行（保持設定に基づくバックアップやスナップショット）がトリガーされます。デフォルトは `false`です。これは以降の繰り返しには影響しません。

  バックアップとスナップショットのスケジュールの YAML の例:

```
---
```

```
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: Daily
  hour: "0"
  minute: "0"
```

スナップショットのみのスケジュールの YAML の例:

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

即時実行のスケジュールの YAML の例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-daily-schedule-run-immediately
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "7"
```

```
  snapshotRetention: "7"
  granularity: Daily
  hour: "3"
  minute: "0"
  runImmediately: true
```

3. ファイルに正しい値を入力したら `trident-protect-schedule-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

CLIを使用してスケジュールを作成する

手順

1. 保護スケジュールを作成し、角かっこ内の値を環境からの情報に置き換えます。例：

メモ	を使用すると、このコマンドの詳細なヘルプ情報を表示できます `tridentctl-protect create schedule --help`。

```
tridentctl-protect create schedule <my_schedule_name> \
  --appvault <my_appvault_name> \
  --app <name_of_app_to_snapshot> \
  --backup-retention <how_many_backups_to_retain> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_to_run_schedule> \
  --granularity <frequency_to_run> \
  --hour <hour_of_day_to_run> \
  --minute <minute_of_hour_to_run> \
  --recurrence-rule <recurrence> \
  --snapshot-retention <how_many_snapshots_to_retain> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

次のフラグを使用すると、スケジュールをさらに制御できます。

- フルバックアップスケジュール: `--full-backup-rule` 非増分完全バックアップをスケジュールするためのフラグ。このフラグは、 `--granularity Daily`。可能な値:
  - `Always`: 毎日フル バックアップを作成します。
  - 特定の曜日: 1 つ以上の曜日をカンマで区切って指定します (例: `"Monday,Thursday"`）。有効な値: 月曜日、火曜日、水曜日、木曜日、金曜日、土曜日、日曜日。

メモ	その `--full-backup-rule` フラグは、時間単位、週単位、または月単位の粒度では機能しません。

- 即時ベースライン保護： `--run-immediately true`を使用すると、最初のスケジュールされた実行時間を待たずに、スケジュールが作成されるとすぐに初期バックアップまたはスナップショットを作成できます。デフォルトは `false`です。
- スナップショットのみのスケジュール: 設定 `--backup-retention 0` 0より大きい値を指定する `--snapshot-retention`。

サポートされているスケジュール注釈

次の表は、スケジュール CR を作成するときに使用できる注釈を示しています。

アノテーション	を入力します	説明	デフォルト値
protect.trident.netapp.io/フルバックアップルール	文字列	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します。設定できるのは `Always` 継続的なフル 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、日単位の粒度を選択した場合、フルバックアップを実行する曜日を指定できます（例： `"Monday,Thursday"`）。有効な曜日の値は、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日、日曜日です。この注釈は、以下のスケジュールでのみ使用できます。 `granularity` に設定 `Daily`。	未設定（すべてのバックアップは増分です）
protect.trident.netapp.io/スナップショット完了タイムアウト	文字列	スナップショット操作全体が完了するまでに許容される最大時間。	「60メートル」
protect.trident.netapp.io/ボリュームスナップショットの使用準備完了のタイムアウト	文字列	ボリューム スナップショットが使用可能状態になるまでに許容される最大時間。	「30メートル」
protect.trident.netapp.io/ボリュームスナップショット作成タイムアウト	文字列	ボリューム スナップショットの作成に許可される最大時間。	「5メートル」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。	「1200」（20分）

Snapshot を削除します

不要になったスケジュール済みまたはオンデマンドの Snapshot を削除します。

手順

1. Snapshotに関連付けられているSnapshot CRを削除します。

```
kubectl delete snapshot <snapshot_name> -n my-app-namespace
```

バックアップを削除します

不要になったスケジュール済みまたはオンデマンドのバックアップを削除します。

メモ	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。このポリシーのデフォルト設定は `Retain`偶発的なデータ損失を防ぐためです。ポリシーが変更されていない場合は、 `Delete`バックアップ データはオブジェクト ストレージに残り、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
```

バックアップ処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したバックアップ処理のステータスを確認できます。

手順

1. 次のコマンドを使用してバックアップ処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

azure-anf-files NetApp（ANF）処理のバックアップとリストアを実現

Trident Protect をインストールしている場合は、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。

作業を開始する前に

次の点を確認します。

- Trident Protect をインストールしました。
- 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。
- ストレージバックエンドのデフォルトのストレージクラスとしてを選択しまし `azure-netapp-files` た。

▼ 構成手順用に展開

1. Trident 24.10にアップグレードする前にANFボリュームを作成した場合は、Tridentで次の手順を実行します。

    a. アプリケーションに関連付けられているNetAppファイルベースの各PVのSnapshotディレクトリを

有効にします。

```
tridentctl update volume <pv name> --snapshot-dir=true -n trident
```

b. 関連付けられている各PVに対してSnapshotディレクトリが有効になっていることを確認します。

```
tridentctl get volume <pv name> -n trident -o yaml | grep
snapshotDir
```

応答：

```
snapshotDirectory: "true"
```

+
スナップショット ディレクトリが有効になっていない場合、 Trident Protect は通常のバックアップ機能を選択し、バックアップ プロセス中に容量プールのスペースを一時的に消費します。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。

Trident Protectでボリュームグループのスナップショットを使用する

ボリュームグループのスナップショットを使用して、関連する永続ボリューム要求（PVC）を同じ時点でキャプチャします。これは、複数のボリュームに書き込むアプリケーション、例えば複数のディスクを持つ仮想マシン（VM）などに役立ちます。

このタスクについて

ボリューム グループは、 `spec.volumeGroups` カスタム リソース（CR）の `Application` フィールドで定義します。アプリケーションのスナップショットを作成すると、Trident Protect は ONTAP 整合グループを使用して、ボリューム グループ内のすべての PVC を同時にスナップショットします。ボリューム グループ内のネームスペースごとに 1 つの整合グループが作成されます。グループ内の各ボリュームにも個別のスナップショットが作成されるため、ボリュームを個別にリストアできます。

Trident Protectは、ボリュームグループに属していないPVCの個別のスナップショットを作成します。これらのスナップショットは連続して取得されるため、必ずしも同じ時点の状況を反映しているとは限りません。

ボリュームグループのスナップショットは、オンデマンドスナップショット、スケジュールされたスナップショット、およびTrident Protectがバックアップの一部として取得するスナップショットに適用されます。既存のスナップショットをバックアップする場合、Trident Protectはそのスナップショットを使用し、新しいグループスナップショットは作成しません。レプリケーションでも、別のグループスナップショットは作成されません。グループスナップショットがすでに作成したボリュームごとのスナップショットを使用します。

Trident Protectは、以下のすべての条件を満たす場合にのみ、同じ時点のボリュームをキャプチャします：

- それらは同じボリュームグループに属しています。
- これらは同じネームスペースにあります。
- それらは同じストレージ仮想マシン（SVM）上にあります。
- ボリュームグループには、そのネームスペース内に80個以下のボリュームが含まれます。

Trident Protectが、要件を満たしていないなどの理由でグループスナップショットを作成できない場合、`status.warnings`に警告を追加し、そのグループのPVCの個別のスナップショットを作成します。これらのスナップショットは次々に作成されるため、同じ時点の状態を反映していない可能性があります。アプリケーションのスナップショットは引き続き完了します。

以下の表は、一般的な状況で何が起こるかを説明しています。

状況	結果
ボリュームグループは複数のネームスペース内のPVCを選択します	グループスナップショットは1つのネームスペースに属するため、Trident Protect は各ネームスペースに個別のグループスナップショットを作成します。これらのグループスナップショットは同時に作成されるわけではありません。1つが成功する一方で、別のグループスナップショットがまだ実行中であったり、失敗したり、または個別のスナップショットにフォールバックしたりする場合があります。たとえば、`app: payments-db`のラベルが付いた PVC は、 `payments-a`と`payments-b`において2つの個別のグループスナップショットになります。異なるネームスペースのグループスナップショットを1つのリストアポイントとして扱わないでください。
ネームスペース内のPVCが複数のSVM上に存在する	Trident Protectは、SVMやTridentバックエンドごとにボリュームを分割しません。ONTAPの整合性グループには複数のSVMのボリュームを含めることができないため、SVMを共有するボリュームを含め、そのネームスペース全体のグループスナップショットが失敗します。Trident Protectは警告を追加し、グループスナップショットを削除して、各ボリュームの個別のスナップショットを作成します。ボリュームをまとめてキャプチャするには、同じSVMに配置してください。
ネームスペースに80を超える一致するPVCがあります	Trident Protectはそのネームスペースのみをスキップし、そのPVCの個々のスナップショットを作成します。同じボリュームグループ内の他のネームスペースでは、引き続きグループスナップショットが作成されます。
PVCが複数の `volumeGroups`エントリに一致する	Trident Protectは両方のボリュームグループにPVCを追加しますが、グループスナップショットのいずれかが失敗する可能性があります。失敗したグループスナップショットは、個別のスナップショットにフォールバックします。各PVCが1つのエントリのみに一致することを確認してください。
ボリュームグループには、ボリュームグループのスナップショットをサポートしないドライバまたはプロトコルを使用するPVCが含まれます。	グループスナップショットが失敗し、Trident ProtectはそれらのPVCの個別のスナップショットを作成します。

作業を開始する前に

- PVCは、ボリュームグループのスナップショットをサポートするストレージドライバとプロトコルを使用する必要があります。サポートされているドライバとプロトコルについては、"Tridentボリュームグループのスナップショットの操作"を参照してください。

- クラスターはKubernetes 1.36以降で動作している必要があります。
- Kubernetesスナップショットコントローラはバージョン8.5.0以降であり、 `CSIVolumeGroupSnapshot`機能ゲートが有効になっている必要があります。
- `v1`、 `VolumeGroupSnapshot`、 `VolumeGroupSnapshotClass`、および `VolumeGroupSnapshotContent`の CRD をインストールする必要があります。
- オンラインバックエンドには Trident 26.10 以降を使用する必要があります。 `CSIVolumeGroupSnapshot`機能ゲートは、Trident `csi-snapshotter`コンテナでも有効にする必要があります。
- `VolumeGroupSnapshotClass`ドライバーを使用する `csi.trident.netapp.io`を定義する必要があります。クラスが1つだけ定義されている場合、`Trident Protect` はそのクラスを使用します。複数のクラスが定義されている場合は、`Trident Protect` が使用するクラスに `snapshot.storage.kubernetes.io/is-default-class: "true"`で注釈を付けます。それ以外の場合、`Trident Protect` はアルファベット順で最初のクラスを使用します。 `Application`または `Snapshot` CR でクラスを選択することはできません。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"を参照してください。
- TridentバックエンドのONTAPアカウントには、整合性グループの作成、取得、削除、およびスナップショットの作成が許可されている必要があります。"アプリケーションの一貫性グループを管理する"および"アプリケーションの一貫性グループのスナップショットを管理する"を参照してください。デフォルトのSVM管理者ロール(`vsadmin`）には、これらの権限は含まれていません。

ボリュームグループを定義する

Trident Protect は、ボリュームグループに属するボリュームに対してのみグループスナップショットを作成します。グループを定義するには、 `spec.volumeGroups` を `Application` CR に追加します。各エントリは個別のグループです。ラベル、VM、またはその両方でボリュームを選択します。

このタスクについて

- ボリュームグループは、 `Application` CRでのみ定義できます。 `tridentctl-protect` CLIにはボリュームグループオプションはありません。
- Trident Protectは、ボリュームグループ内のバインドされたPVCのみを含め、バインドされていないPVCの個別のスナップショットを作成します。エントリでバインドされたPVCが選択されていない場合、Trident Protectはそのエントリのグループスナップショットを作成しません。
- 各PVCが1つのエントリのみに対応するようにしてください。

作業を開始する前に

保護するアプリケーションを定義します。"Trident Protectで管理するアプリケーションを定義する"を参照してください。

手順

1. ラベルでボリュームを選択する場合は、グループ化したいPVCにラベルを付けてください。ラベルのキーと値は任意に使用できます。このページの例では `volume-group=my-volume-group`を使用しています。

```
kubectl label pvc <pvc_name_1> <pvc_name_2> volume-group=my-volume-group
-n <application_namespace>
```

2. `spec.volumeGroups`を既存の `Application`CRファイルに追加します。各エントリには、`labelSelector`、 `vmSelector`、またはその両方を含める必要があります。エントリに両方のフィー

ルドが含まれている場合、ボリュームグループには、いずれかのフィールドで選択されたすべてのボリュームが含まれます。

- **spec.volumeGroups[].labelSelector**：アプリケーション内のPVCを選択するKubernetesラベルセレクタ(`matchLabels`または `matchExpressions`）です。空のセレクタは無効です。
- **spec.volumeGroups[].vmSelector**：KubeVirt VMのリスト。それぞれに `namespace`と `name`があります。Trident Protectには、各VMがマウントするPVCが含まれます。

  以下の例では、2つのボリュームグループを定義しています。最初のグループには、 `volume-group: my-volume-group`というラベルが付いたPVCが含まれます。2番目のグループには、 `my-vm-1`と `my-vm-2`がマウントするPVCが含まれます。

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-namespace
spec:
  includedNamespaces:
    - namespace: my-namespace
  volumeGroups:
    - labelSelector:
        matchLabels:
          volume-group: my-volume-group
    - vmSelector:
        - namespace: my-namespace
          name: my-vm-1
        - namespace: my-namespace
          name: my-vm-2
```

+

メモ	お客様の `Application` CR にすでに含まれている他のフィールド（ `spec.resourceFilter` など）はそのまま保持してください。適用するファイルからフィールドを省略すると、 `kubectl apply`はそのフィールドを CR から削除します。

3. 更新されたCRを適用します：

```
kubectl apply -f <application_cr_file>.yaml
```

結果

`Application`
CRにはボリュームグループが含まれます。アプリケーションの次のスナップショットでは、それらが使用されます。

Snapshot を作成します

アプリケーションのオンデマンドスナップショット、オンデマンドバックアップ、またはデータ保護スケジュールを作成します。フィールドを追加する必要はありません。Trident Protectは、 `Application` CRで定義したボリュームグループを使用します。手順については、"Trident Protectを使用してアプリケーションを保護する"を参照してください。

メモ	デフォルトでは、Trident Protectは、VMファイルシステムをフリーズする前に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内のボリュームはフリーズせずにキャプチャされるため、VMがまだディスクに書き込んでいないデータは含まれません。ボリュームグループに含まれていないディスクは、Trident ProtectがVMをフリーズした後に個別のスナップショットを作成する際に、後でキャプチャされます。VMのすべてのディスクを同時にキャプチャするには、すべてのディスクをボリュームグループに含めます。VMファイルシステムのフリーズの詳細については、"KubeVirt VMによるデータ保護"を参照してください。

結果を確認する

スナップショットが完了したら、ステータスを確認して、Trident Protectがグループスナップショットを作成したことを確認します。

メモ	アプリケーションのスナップショットが完了したからといって、そのグループのスナップショットが成功したとは限りません。すべてのグループスナップショットがスキップされたり失敗したりした場合でも、 `VolumeGroupSnapshotsCreated`条件は `true`になり、`Trident Protect`はKubernetesイベントやメトリックを作成しません。これを検出するには、`status.warnings`、および`skipped`と`failed`のフィールドを監視します。

手順

1. スナップショットの名前を取得します。アプリケーションのスナップショットを一覧表示するには、次のコマンドを実行します：

```
kubectl get snapshots -n <application_namespace>
```

2. スナップショットの状態を確認してください：

```
kubectl get snapshots <snapshot_name> -n <application_namespace> -o yaml
```

結果

`status.volumeGroupSnapshots` フィールドには、グループスナップショットごとに1つのエントリがあります。`Trident Protect`は、ボリュームグループ内のネームスペースごとに個別のグループスナップショットを作成します。以下の例は省略されています：

```
status:
  volumeGroupSnapshots:
    - name: vgs-<snapshot_uid_prefix>-<group_index>-<namespace>
```

```
      namespace: <namespace>
      ready: true
      memberPVCs:
        - pvcName: <pvc_name_1>
          pvcNamespace: <namespace>
        - pvcName: <pvc_name_2>
          pvcNamespace: <namespace>
```

グループスナップショット名には、 `Snapshot` UIDの最初の8文字、 `spec.volumeGroups` エントリのインデックス（ `0` から開始）、および12文字に切り詰められたネームスペースが含まれます。

- `ready: true` は、グループスナップショットが成功したことを意味します。グループを構成するスナップショットは、 `status.volumeSnapshots` にも表示されます。これらはグループのメンバースナップショットであり、同じボリュームの追加スナップショットではありません。
- `skipped: true` は、ボリュームグループのそのネームスペース内の PVC が 80 個を超えているため、Trident Protect がグループスナップショットの作成を試みなかったことを意味します。Trident Protect はグループのボリュームの個別スナップショットを作成し、 `skipReason` テキストを `status.warnings` に追加します。
- `failed: true` は、Trident Protect がグループスナップショットの作成を試みたが失敗したことを意味します。Trident Protect はグループのボリュームの個別のスナップショットを代わりに作成し、 `error` テキストを `status.warnings`に追加します。

一時的な**PVC**ラベルとクリーンアップ

Trident Protectは、グループスナップショット内の各PVCにラベルを追加します（例： `protect.trident.netapp.io/vgs-group-673c0189-0=true`）。お客様独自のラベルは変更されません。アプリケーションスナップショットを削除するまで、ラベルはPVCに残ります。アプリケーションスナップショットを削除すると、Trident Protectはボリュームグループスナップショットとメンバースナップショットも削除し、PVCからラベルを削除します。これは、スナップショットの `reclaimPolicy`が `Delete`であるか `Retain`であるかにかかわらず発生します。

次の手順

他のスナップショットと同様の方法で、スナップショットからアプリケーションをリストアします。ボリュームグループのスナップショットには、個別のリストアはありません。Trident Protectはメンバースナップショットからリストアするため、グループ内のすべてのPVCをリストアすることなく、個々のPVCまたはVMをリストアできます。"Snapshotから別のネームスペースへのリストア" または "Snapshotから元のネームスペースへのリストア" を参照してください。

関連情報

- "Trident Protectで管理するアプリケーションを定義する"
- "Trident Protectを使用してアプリケーションを保護する"
- "KubeVirt 仮想マシンを Trident Protect で保護する"

リストアアプリケーション

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。

メモ

- アプリケーションを復元すると、そのアプリケーションに設定されているすべての実行フックがアプリケーションとともに復元されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。
- qtreeボリュームでは、バックアップから別のネームスペースまたは元のネームスペースへの復元がサポートされています。ただし、スナップショットから別のネームスペースまたは元のネームスペース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"。
- アーカイブからすべての名前空間リソースを復元せずに特定のKubeVirt仮想マシン（VM）を復元するには、復元CR の `includedVirtualMachines`フィールドを使用します。サポートされているCR、フィールドリファレンス、およびYAMLの例については、"KubeVirt 仮想マシンを Trident Protect で保護する"を参照してください。

バックアップから別のネームスペースへのリストア

BackupRestore CR を使用して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。

メモ

- 既存のリソースがある別のネームスペースにバックアップをリストアしても、バックアップ内のリソースと名前を共有するリソースは変更されません。バックアップ内のすべてのリソースをリストアするには、ターゲットネームスペースを削除して再作成するか、新しいネームスペースにバックアップをリストアします。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

メモ

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の使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.appVaultRef *：（*required*）バックアップコンテンツが格納されているAppVaultの名前。
   - **spec.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       destinationApplicationName: my-new-app-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。

メモ	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要) resourceMatchersで定義されたリソースを使用 `Include`または `Exclude`除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、

種類、バージョン）はAND演算として照合されます。

- *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
- *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
- **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
- * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
- *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
- *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

  例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-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. ファイルに正しい値を入力したら `trident-protect-backup-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

CLI を使用します

手順

1. バックアップを別のネームスペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。 `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2`ます。例：

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

バックアップから元のネームスペースへのリストア

いつでもバックアップを元の名前空間に復元できます。In Placeリストアを実行すると、Trident Protectは、保護スケジュールと進行中の操作を自動的に管理し、無効な回復ポイントを防止します。

- 復元が開始される前に、アプリケーションの有効なすべての保護スケジュールが無効になります。これにより、アプリケーション リソースの復元中に、スケジュールされたバックアップまたはSnapshotが実行されなくなります。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。すでに無効になっているスケジュールは無効のままになります。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元は続行され、復元CRステータスに警告が記録されます。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

メモ	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の使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-ipr-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

```
kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

- * spec.appVaultRef *：（*required*）バックアップコンテンツが格納されているAppVaultの名前。

  例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appArchivePath: my-backup-path
  appVaultRef: appvault-name
```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。

メモ	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要) resourceMatchersで定義されたリソースを使用 `Include`または `Exclude`除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
  - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
    - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
    - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
    - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
    - * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
    - *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
    - *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

      例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-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. ファイルに正しい値を入力したら `trident-protect-backup-ipr-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

CLI を使用します

手順

1. バックアップを元のネームスペースにリストアします。角かっこ内の値は、使用している環境の情報に置き換えてください。この `backup`引数では、という形式のネームスペースとバックアップ名を使用し `<namespace>/<name>`ます。例：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

バックアップから別のクラスタへのリストア

元のクラスタで問題が発生した場合は、バックアップを別のクラスタにリストアできます。

メモ

- 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 を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自

動的に作成します。

作業を開始する前に

次の前提条件が満たされていることを確認します。

- 宛先クラスターにはTrident Protect がインストールされています。
- デスティネーションクラスタは、バックアップが格納されているソースクラスタと同じAppVaultのバケットパスにアクセスできます。
- 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。
- 長時間実行されるリストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。
  - 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント"ください。
  - AWSリソースのクレデンシャルの詳細については、を参照してください "AWSのドキュメント"。

手順

1. Trident Protect CLIプラグインを使用して、デスティネーション クラスタにAppVault CRが存在することを確認します：

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

メモ

AppVault CR がデスティネーション クラスタに存在しない場合は、"Trident Protect AppVault オブジェクトを使用してバケットを管理する"の手順に従って作成します。

2. デスティネーション クラスタで利用可能なAppVaultのバックアップコンテンツを表示し、復元するバックアップの `appArchivePath`をメモします：

```
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で使用可能なバックアップが表示されます。これには、元のクラスタ、対応するアプリケーション名、タイムスタンプ、アーカイブパスが含まれます。

出力例：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |         TIMESTAMP
|    PATH     |
```

```
+------------+----------+--------+----------------
+-------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+------------+----------+--------+----------------
+-------------------------+-------------+
```

3. AppVault名とアーカイブパスを使用して、アプリケーションをデスティネーションクラスタにリストアします。

メモ	CRを使用する場合は、アプリケーションのリストアに使用する名前空間がデスティネーションクラスタに存在することを確認してください。

CRの使用

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-backup-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appVaultRef * :（*required*）バックアップコンテンツが格納されているAppVaultの名前。
   - **spec.appArchivePath**：（必須）AppVault内のバックアップコンテンツが保存されるパス。ステップ2のコマンドを使用してバックアップの内容を表示し、復元するバックアップの `appArchivePath`を確認します。
   - **spec.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     例：

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-backup-path
  destinationApplicationName: my-new-app-name
```

```
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. ファイルに正しい値を入力したら `trident-protect-backup-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

CLI を使用します

1. 次のコマンドを使用してアプリケーションをリストアし、括弧内の値を環境の情報に置き換えます。namespace-mapping引数では、コロンで区切られた名前空間を使用して、ソース名前空間をsource1:dest1、source2:dest2の形式で正しいデスティネーション名前空間にマッピングします。例：

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--destination-app-name <custom_app_name> \
--context <destination_cluster_name> \
-n <application_namespace>
```

Snapshotから別のネームスペースへのリストア

カスタム リソース (CR) ファイルを使用して、スナップショットから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。

メモ

- 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。
- CR を使用して新しい名前空間に復元する場合は、CR を適用する前に、宛先名前空間を手動で作成する必要があります。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。

- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

CRの使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appVaultRef * :（*required*）スナップショットコンテンツが格納されているAppVaultの名前。
   - * spec.appArchivePath * :スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.destinationApplicationName**：（オプション）リストアされたアプリケーションの名前。指定されている場合、リストアされたアプリケーションはこの名前を使用します。指定しない場合、リストアされたアプリケーションはソース アプリケーション名を使用します。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （オプション）リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。

メモ	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要) resourceMatchersで定義されたリソースを使用 `Include`または `Exclude`除外します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
  - **resourceFilter.resourceMatchers**: resourceMatcherオブジェクトの配列。この配列に複数の要素を定義した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
    - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
    - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
    - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
    - * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
    - *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
    - *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

      例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-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. ファイルに正しい値を入力したら `trident-protect-snapshot-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

CLI を使用します

手順

1. スナップショットを別のネームスペースにリストアし、括弧内の値を環境の情報に置き換えます。

   - `snapshot`引数では、という形式のネームスペースとSnapshot名を使用し `<namespace>/<name>`ます。
   - `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2`ます。

     例：

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name <custom_app_name> \
-n <application_namespace>
```

Snapshotから元のネームスペースへのリストア

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。In Placeリストアを実行すると、Trident Protectは、保護スケジュールと進行中の操作を自動的に管理し、無効な回復ポイントを防止します：

- 復元が開始される前に、アプリケーションの有効なすべての保護スケジュールが無効になります。これにより、アプリケーション リソースの復元中に、スケジュールされたバックアップまたはSnapshotが実行されなくなります。
- 復元が正常に完了すると、復元前に有効になっていたスケジュールのみが再度有効になります。すでに無効になっているスケジュールは無効のままになります。
- 復元が開始される前に、進行中のバックアップまたはスナップショット操作はすべてキャンセルされます。操作が5分以内にキャンセルされない場合、復元は続行され、復元CRステータスに警告が記録されます。

作業を開始する前に

長時間実行されるs3リストア処理には、AWSセッショントークンの有効期限が十分であることを確認してください。リストア処理中にトークンの有効期限が切れた場合、処理が失敗することがあります。

- 現在のセッショントークンの有効期限を確認する方法については、を参照して "AWS APIのドキュメント" ください。
- AWSリソースのクレデンシャルの詳細については、を参照してください "AWS IAMのドキュメント"。

CRの使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-ipr-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。

   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appVaultRef * :（*required*）スナップショットコンテンツが格納されているAppVaultの名

前。

- * spec.appArchivePath *：スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

```
kubectl get snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

```
---
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-snapshot-path
```

+ .（オプション）アプリケーションの特定のリソースのみを選択して復元する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するかを指定するフィルタリングを追加します。

+ 注記：Trident Protect は、ユーザーが選択したリソースとの関連性に基づいて、一部のリソースを自動的に選択します。例えば、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） `Include`または`Exclude`を使用して、`resourceMatchers`で定義されたリソースを含めるか除外します。含めるリソースまたは除外するリソースを定義するには、以下の`resourceMatchers`パラメータを追加します：**`resourceFilter.resourceMatchers`**：**`resourceMatcher`**オブジェクトの配列。この配列に複数の要素を定義した場合、それらは**OR**演算で一致し、各要素内のフィールド（**`group`**、**`kind`**、**`version`**）は**AND**演算で一致します。* **`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`フィールドにある名前空間。* **`resourceMatchers[].labelSelectors`**：（オプション） "Kubernetes のドキュメント"で定義されたリソースの`Kubernetes metadata.name`フィールド内のラベルセレクタ文字列。例：`"trident.netapp.io/os=linux"`。

+ 例：

+

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
```

```
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

+ . `trident-protect-snapshot-ipr-cr.yaml`ファイルに正しい値を入力したら、CRを適用します。

+

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

CLI を使用します

手順

1. Snapshotを元のネームスペースにリストアします。括弧内の値は、環境の情報に置き換えてください。例：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
-n <application_namespace>
```

リストア処理のステータスの確認

コマンドラインを使用して、実行中、完了、または失敗したリストア処理のステータスを確認できます。

手順

1. 次のコマンドを使用してリストア処理のステータスを取得し、角かっこ内の値を環境の情報に置き換えます。

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

注釈、名前空間設定、ストレージ オプションなどの詳細設定を使用して、特定の要件を満たすように復元操作をカスタマイズできます。

リストア処理とフェイルオーバー処理時のネームスペースのアノテーションとラベル

リストア処理とフェイルオーバー処理では、デスティネーションネームスペースのラベルとアノテーションがソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネームスペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、すでに存在するラベルまたはアノテーションがソースネームスペースの値に一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルやアノテーションは変更されません。

メモ	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に関するドキュメント" 。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数を設定することで、デスティネーションネームスペースの特定のアノテーションが上書きされないようにすることができます `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。例：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

メモ	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、 "追加のTrident Protectヘルムチャート設定を構成する"。

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、ソースとデスティネーションのネームスペースを示しています。それぞれにアノテーションとラベルが設定されています。処理の前後のデスティネーションネームスペースの状態、およびデスティネーションネームスペースでアノテーションやラベルを組み合わせたり上書きしたりする方法を確認できます。

リストアまたはフェイルオーバー処理の前

次の表に、リストアまたはフェイルオーバー処理を実行する前のソースネームスペースとデスティネーションネームスペースの状態を示します。

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true"	• 環境=本番 • コンプライアンス= HIPAA • 名前= ns-1
ネームスペースns-2（デスティネーション）	• Annotation.one/key："true" • annotation.three/key："false"	• ロール=データベース

リストア処理後

次の表に、リストアまたはフェイルオーバー処理後の例のデスティネーションネームスペースの状態を示します。一部のキーが追加され、一部のキーが上書きされ、 `name`デスティネーションネームスペースに一致するようにラベルが更新されました。

ネームスペース	アノテーション	ラベル
ネームスペースns-2（デスティネーション）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true" • annotation.three/key："false"	• 名前= ns-2 • コンプライアンス= HIPAA • 環境=本番 • ロール=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、システムのさまざまな動作を設定するためにサポートされているアノテーションの一覧を示します。ユーザーがアノテーションを明示的に設定しない場合、システムはデフォルト値を使用します。

アノテーション	を入力します	説明	デフォルト値
保護.trident.netapp.io/データムーバータイムアウト秒	文字列	データ ムーバー操作を停止できる最大時間 (秒単位)。	「300」
保護.trident.netapp.io/kopia-content-cache-size-limit-mb	文字列	Kopia コンテンツ キャッシュの最大サイズ制限 (メガバイト単位)。	「1000」
protect.trident.netapp.io/pvc-bind-timeout-sec	文字列	新しく作成されたPersistentVolumeClaims（PVC）が到達するまでの最大待機時間（秒）。 `Bound`操作が失敗する前のフェーズ。すべての復元 CR タイプ (BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore) に適用されます。ストレージ バックエンドまたはクラスターでより多くの時間が必要になることが多い場合は、より高い値を使用します。	「1200」（20分）
protect.trident.netapp.io/retain-mac-address	文字列	KubeVirt VMの場合は、リストア時に最初のネットワークインターフェイスのMACアドレスを保持する（アプリケーションライセンスがMACアドレスにバインドされている場合など）ように `"true"`に設定します。環境：すべてのリストアCRタイプおよびAppMirrorRelationship（関係をプロモートする前に設定します）。デフォルトでは、Trident Protectは競合を防ぐためにこのMACアドレスを削除します。ソースVMが同じネットワーク上でまだ実行されている場合は、MACアドレスの重複を避けるために、まずシャットダウンするか、隔離されたネットワークにリストアしてください。	設定されていません（MACアドレスは削除されています）

カスタムリソースを使用して復元する際にリソースを変更します

リソース変換機能を使用すると、リソースの復元中にリソースを変更できます。これは、復元されたバージョンがオリジナルと異なる必要がある場合に役立ちます。たとえば、アプリケーションを別のクラスタに復元する際にConfigMapのホスト名を更新する場合などです。

リソース変更の仕組み

`SnapshotRestore`、 `BackupRestore`、 `AppMirrorRelationship`、およびその他のリストア リソースの `transformations`フィールドを使用すると、リストア プロセス中に Kubernetes

リソースを変更できます。これは、リストアされたアプリケーションをオリジナルとは異なる方法で構成する必要がある場合（ホスト名、レジストリ
URL、リソース制限、環境変数の変更など）に便利です。

リソース変換では、 "RFC 6902 JSON Patch"操作と "RFC 6901 JSONポインタ"パスを使用して、Kubernetes リソース内の特定のフィールドをターゲットにして変更します。

以下に、リソース変換を含む復元オブジェクトの基本的な構造を示します。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: my-restore
  namespace: target-namespace
spec:
  appVaultRef: my-vault
  appArchivePath: /path/to/snapshot
  namespaceMapping:
    - source: source-ns
      destination: target-ns
  transformations:
    - resource:
        kind: ConfigMap       # Required: resource kind
        group: ""             # Optional: API group (empty for core
resources)
        version: ""           # Optional: API version
        name: ""              # Optional: specific resource name
      operations:
        - op: replace         # Operation type
          path: "/data/key"   # JSON Pointer path
          value: "new-value"  # New value (for add/replace/test)
```

サポートされているリソース

以下の条件に一致するリソースに対して、リソース変換を使用できます：

- **kind** （必須）：Kubernetes リソースの種類（例： `ConfigMap`、 `Deployment`、 `Pod`）
- グループ（オプション）：APIグループ（例： `apps`、 `route.openshift.io`）—コアリソースの場合は省略
- **version** (オプション)：API バージョン （例： `v1`、 `v1beta1`）
- 名前（オプション）：名前で指定した特定のリソースにのみ適用します

メモ	PersistentVolumeClaimsと名前空間の変更は現在サポートされていません。

サポートされている操作

リソースを変更するには、以下の操作を使用できます： `add`、 `copy`、 `move`、 `remove`、 `replace`、および `test`。

リソースに値を追加する

`add`操作を使用して、指定されたパスに新しいフィールドまたは値を追加します。オブジェクトや配列にデータを追加できます。

CRの使用

次の例では、ノードセレクタを `Deployment`リソースに追加します：

```
transformations:
  - resource:
      kind: Deployment
    operations:
      - op: add
        path: "/spec/template/spec/nodeSelector"
        value:
          "topology.kubernetes.io/zone": "us-east-1a"
          disktype: "ssd"
```

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'apps,v1,Deployment:add{"path":"/spec/template/spec/nodeSelector","valu
e":{"topology.kubernetes.io/zone":"us-east-1a","disktype":"ssd"}}'
```

リソース内の値をコピーする

`copy`操作を使用して、同じリソース内のあるパスから別のパスに値をコピーします。ソースは変更されません。

CRの使用

次の例では、 `ConfigMap`オブジェクトのデータキーを複製します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: copy
        from: "/data/source-key"
        path: "/data/backup-key"
```

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation ',v1,ConfigMap:copy{"from":"/data/source-
key","path":"/data/backup-key"}'
```

リソース内で値を移動する

`move`操作を使用して、同じリソース内であるパスから別のパスに値を移動します。ソースが削除され、値が宛先に配置されます。

CRの使用

次の例では、 `ConfigMap`オブジェクトのデータキーの名前を変更します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: move
        from: "/data/OLD_KEY"
        path: "/data/NEW_KEY"
```

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
```

```
  --transformation
',v1,ConfigMap:move{"from":"/data/OLD_KEY","path":"/data/NEW_KEY"}'
```

リソースから値を削除する

`remove`操作を使用して、指定されたパスにあるフィールドまたは値を削除します。

CRの使用

次の例では、`ConfigMap`リソースから注釈を削除します：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: remove
        path: "/metadata/annotations/kubectl.kubernetes.io~1last-
applied-configuration"
```

メモ　上記の例のパスでは、`~1`は`/`のJSONポインターのエスケープシーケンスです。

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:remove{"path":"/metadata/annotations/kubectl.kubernetes.
io~1last-applied-configuration"}'
```

リソース内の値を置き換える

`replace`操作を使用して、指定されたパスにあるリソース内の既存の値を置き換えます。JSONパスは既に存在している必要があります。

CRの使用

次の例では、`Route`オブジェクトのホスト名を変更します：

```
transformations:
  - resource:
      kind: Route
      group: route.openshift.io
    operations:
      - op: replace
        path: "/spec/host"
        value: "prod.example.com"
```

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'route.openshift.io,v1,Route:replace{"path":"/spec/host","value":"prod.
example.com"}'
```

リソースの変更をテストする

`test`操作を使用して、他の操作が適用される前に、パス上の値が期待値と一致することを確認します。テストが失敗した場合、変更全体がロールバックされます。

CRの使用

次の例では、 `environment`が `staging`に設定されている場合にのみ `database-host`を更新します。

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: test
        path: "/data/environment"
        value: "staging"
      - op: replace
        path: "/data/database-host"
        value: "prod-db.example.com"
```

CLI を使用します

コマンドラインを使用してこの変換を実行するには、次のコマンドを使用します：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:test{"path":"/data/environment","value":"staging"},repla
ce{"path":"/data/database-host","value":"prod-db.example.com"}'
```

NetApp SnapMirrorでアプリケーションをレプリケート

Trident Protectを使用してアプリケーションのレプリケーションを設定および管理する

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。

リストア処理とフェイルオーバー処理時のネームスペースのアノテーションとラベル

リストア処理とフェイルオーバー処理では、デスティネーションネームスペースのラベルとアノテーションがソースネームスペースのラベルとアノテーションと一致するように作成されます。デスティネーションネームスペースに存在しないソースネームスペースのラベルまたはアノテーションが追加され、すでに存在するラベルまたはアノテーションがソースネームスペースの値に一致するように上書きされます。デスティネーションネームスペースにのみ存在するラベルやアノテーションは変更されません。

メモ	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に関するドキュメント" 。

リストアまたはフェイルオーバー処理を実行する前にKubernetes環境変数を設定することで、デスティネーションネームスペースの特定のアノテーションが上書きされないようにすることができます `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。例：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

メモ	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、 "追加のTrident Protectヘルムチャート設定を構成する"。

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、ソースとデスティネーションのネームスペースを示しています。それぞれにアノテーションとラベルが設定されています。処理の前後のデスティネーションネームスペースの状態、およびデスティネーションネームスペースでアノテーションやラベルを組み合わせたり上書きしたりする方法を確認できます。

リストアまたはフェイルオーバー処理の前

次の表に、リストアまたはフェイルオーバー処理を実行する前のソースネームスペースとデスティネーションネームスペースの状態を示します。

ネームスペース	アノテーション	ラベル
ネームスペースns-1（ソース）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true"	• 環境=本番 • コンプライアンス= HIPAA • 名前= ns-1
ネームスペースns-2（デスティネーション）	• Annotation.one/key："true" • annotation.three/key："false"	• ロール=データベース

リストア処理後

次の表に、リストアまたはフェイルオーバー処理後の例のデスティネーションネームスペースの状態を示します。一部のキーが追加され、一部のキーが上書きされ、 `name`デスティネーションネームスペースに一致するようにラベルが更新されました。

ネームスペース	アノテーション	ラベル
ネームスペースns-2（デスティネーション）	• Annotation.one/key："UpdatedValue" • Annotation.Two/key："true" • annotation.three/key："false"	• 名前= ns-2 • コンプライアンス= HIPAA • 環境=本番 • ロール=データベース

メモ	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、実行フックはアプリケーション上に存在し、関

連するアクションの実行中に実行されます。

- リバース同期またはリバース再同期中、アプリケーション上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ターゲットアプリケーションになると、これらの実行フックは無効となり、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

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）
- Snapshotの作成時間の設定

手順

1. ソースクラスタで、ソースアプリケーションのAppVaultを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します"AppVaultカスタムリソース"。

CRを使用した**AppVault**の作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-appvault-primary-source.yaml`）。

b. 次の属性を設定します。

- * metadata.name*:（*required*）AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
- * spec.providerConfig*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。bucketNameとプロバイダーに必要なその他の詳細を選択します。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"AppVaultカスタムリソース"。
- * spec.providerCredentials*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
  - * spec.providerCredentials.valueFromSecret*:（*required*）は、クレデンシャル値がシークレットから取得される必要があることを示します。
    - * key * :（*required*）選択するシークレットの有効なキー。
    - * name * :（*required*）このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
  - * spec.providerCredentials.secretAccessKey*:（*required*）プロバイダへのアクセスに使用するアクセスキー。name は spec.providerCredentials.valueFromSecret.name*と一致している必要があります。
- * spec.providerType*:（*required*）は、バックアップの提供元（NetApp ONTAP S3、汎用S3、Google Cloud、Microsoft Azureなど）を決定します。有効な値：

- AWS
- Azure
- GCP
- 汎用- s3
- ONTAP - s3
- StorageGRID - s3

c. ファイルに正しい値を入力したら `trident-protect-appvault-primary-source.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

CLIを使用した**AppVault**の作成

a. AppVaultを作成し、括弧内の値を環境からの情報に置き換えます。

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. ソースクラスタで、ソースアプリケーションCRを作成します。

CRを使用したソースアプリケーションの作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-app-source.yaml` ）。

b. 次の属性を設定します。

- * metadata.name*:（*required*）アプリケーションカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
- * spec.includedNamespaces*:（*required*）名前空間と関連ラベルの配列。名前空間名を使用し、必要に応じてラベルを使用して名前空間のスコープを絞り込み、ここにリストされている名前空間に存在するリソースを指定します。アプリケーション名前空間は、この配列の一部である必要があります。
  - YAMLの例*：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-app-namespace
```

```
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. ファイルに正しい値を入力したら `trident-protect-app-source.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-namespace
```

CLIを使用したソースアプリケーションの作成

a. ソースアプリケーションを作成します。例：

```
tridentctl-protect create app <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します"。
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。

メモ

下記の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、7日間の保持期間を持つ日次スナップショットスケジュールを別途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日間利用可能になりますが、保持期間はユーザーの要件に応じ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、逆操作に使用できます。このアプローチにより、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、逆操作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、という名前を付けます（例： `trident-protect-schedule.yaml` ）。

ii. 次の属性を設定します。

- * metadata.name*:（*required*）スケジュールカスタムリソースの名前。
- **spec.appVaultRef**: (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 *：（*required*）このフィールドは必須であり、値は0に設定する必要があります。
- * spec.enabled *：trueに設定する必要があります。
- * spec.granularity*:はに設定する必要があります `Custom`。
- *spec.recurrenceRule *:開始日をUTC時間と繰り返し間隔で定義します。
- * spec.snapshotRetention *：を2に設定する必要があります。

YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule
  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. ファイルに正しい値を入力したら `trident-protect-schedule.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-
namespace
```

CLIを使用してレプリケーションスケジュールを作成する

a. 括弧内の値を環境の情報に置き換えて、レプリケーション スケジュールを作成します。

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
```

```
<snapshot_retention_count> -n <my_app_namespace>
```

- 例： *

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. デスティネーションクラスタで、ソースクラスタに適用したAppVault CRと同じソースアプリケーションAppVault CRを作成し、という名前を付けます（例： `trident-protect-appvault-primary-destination.yaml`）。
6. CRを適用します。

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
trident-protect
```

7. デスティネーションクラスタに、デスティネーションアプリケーション用のデスティネーションAppVault CRを作成します。ストレージプロバイダに応じて、の例を環境に合わせて変更します"AppVaultカスタムリソース"。
   a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-appvault-secondary-destination.yaml`）。
   b. 次の属性を設定します。
      - * metadata.name*:（*required*）AppVaultカスタムリソースの名前。レプリケーション関係に必要な他のCRファイルがこの値を参照しているため、選択した名前をメモしておいてください。
      - * spec.providerConfig*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要な設定を保存します。およびプロバイダに必要なその他の詳細情報を選択します `bucketName`。レプリケーション関係に必要な他のCRファイルはこれらの値を参照しているため、選択した値をメモしておいてください。他のプロバイダでのAppVault CRSの例については、を参照してください"AppVaultカスタムリソース"。
      - * spec.providerCredentials*:（*required*）は、指定されたプロバイダを使用してAppVaultにアクセスするために必要なすべての資格情報への参照を保存します。
        - * spec.providerCredentials.valueFromSecret*:（*required*）は、クレデンシャル値がシークレットから取得される必要があることを示します。
          - * key * :（*required*）選択するシークレットの有効なキー。
          - * name * :（*required*）このフィールドの値を含むシークレットの名前。同じネームスペースになければなりません。
        - * spec.providerCredentials.secretAccessKey*:（*required*）プロバイダへのアクセスに使用するアクセスキー。name は spec.providerCredentials.valueFromSecret.name*と一致している必要があります。
      - * spec.providerType*:（*required*）は、バックアップの提供元（NetApp ONTAP S3、汎用S3

、Google Cloud、Microsoft Azureなど）を決定します。有効な値：

- AWS
- Azure
- GCP
- 汎用- s3
- ONTAP - s3
- StorageGRID - s3

c. ファイルに正しい値を入力したら `trident-protect-appvault-secondary-destination.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n trident-protect
```

8. デスティネーション クラスタで、AppMirrorRelationship CR ファイルを作成します。

メモ	CR を使用する場合は、CR を適用する前に、宛先名前空間を手動で作成します。Trident Protect は、CLI を使用する場合にのみ名前空間を自動的に作成します。

CRを使用した**AppMirrorRelationship**の作成

a. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-relationship.yaml`）。

b. 次の属性を設定します。

- * metadata.name:*（必須）AppMirrorRelationshipカスタムリソースの名前。
- **metadata.annotations.protect.trident.netapp.io/retain-mac-address**：（オプション）デスティネーション クラスタ上のKubeVirt VMのMACアドレスを保持するには、 `'"true"'`に設定します。"高度なTrident Protect復元設定を使用する"を参照してください。
- * spec.destinationAppVaultRef*:（*required*）この値は、デスティネーションクラスタ上のデスティネーションアプリケーションのAppVaultの名前と一致する必要があります。
- * spec.namespaceMapping*:(*required*)宛先およびソースの名前空間は、それぞれのアプリケーションCRで定義されているアプリケーション名前空間と一致している必要があります。
- *spec.sourceAppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVaultの名前と一致する必要があります。
- **spec.sourceApplicationName**:(*required*)この値は、ソースアプリケーションCRで定義したソースアプリケーションの名前と一致する必要があります。
- **spec.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.storageClassMapping**: （オプション） 同じ AMR 内の個々の PVC のソースストレージクラスをデスティネーションストレージクラスにマッピングします。各エントリには `source`および `destination`ストレージクラス名が必要です。設定すると、 `storageClassMapping`は `storageClassName`を完全にオーバーライドします。一致するソースエントリのない PVC はクラスタのデフォルトのストレージクラスを使用し、 `storageClassName`は使用しません。

  AppMirrorRelationship CR で `storageClassMapping`を使用する際は、以下の制限事項に注意してください。

  - `storageClassMapping`は `storageClassName`を完全にオーバーライドします。両方のフィールドが設定されている場合、 `storageClassName`は無視されます。
  - `storageClassMapping`に一致するエントリがないソースストレージクラスを持つ PVC は、 `storageClassName`ではなく、クラスターのデフォルトストレージクラスを使用します。
  - 各エントリーには、 `source`と `destination`の値が必要です。
  - デスティネーション ストレージクラスを変更すると、既存のデスティネーション PVC の削除と再作成が強制される場合があります。
- *spec.recurrenceRule *:開始日をUTC時間と繰り返し間隔で定義します。

  YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. ファイルに正しい値を入力したら `trident-protect-relationship.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

CLIを使用した**AppMirrorRelationship**の作成

a. AppMirrorRelationship オブジェクトを作成して適用し、括弧内の値を環境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> --storageclass-mapping
<src_class:dst_class> -n <application_namespace>
```

メモ

`--storageclass-mapping`を使用して、個々のソースストレージクラスをデスティネーションストレージクラスにマッピングします。カンマで区切られた1つ以上の `source:destination`ペアを指定します（例： `"class-a:class-b,class-c:class-d"）。 `*:destination-class`をワイルドカードとして使用して、マッピングされていないすべてのPVCを単一のデスティネーションストレージクラスにマッピングします。 `--storageclass-mapping`が設定されている場合、`--storage-class`は無視されます。

単一のデスティネーション ストレージクラスを使用した例(**`--storage-class`**）：

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. （オプション）デスティネーションクラスタで、レプリケーション関係の状態とステータスを確認します。

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

デスティネーションクラスタへのフェイルオーバー

Trident Protect を使用すると、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。

この手順では、最新のスナップショットにフェイルオーバーします。古いスナップショットにフェイルオーバーするには、"Trident Protectを使用して以前のスナップショットにフェイルオーバーする"を参照してください。

手順

1. （オプション）KubeVirt VMのMACアドレスを保持するには、昇格させる前にAppMirrorRelationshipにアノテーションを追加します。"高度なTrident Protect復元設定を使用する"を参照してください。

```
kubectl annotate appmirrorrelationship <relationship_name> -n
<application_namespace> protect.trident.netapp.io/retain-mac-address
="true"
```

2. デスティネーションクラスタで、AppMirrorRelationship CRファイル（など）を編集し `trident-protect-relationship.yaml`、* spec.desiredState*の値をに変更します `Promoted`。
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. （オプション）フェイルオーバーされたアプリケーションで必要な保護スケジュールを作成します。
6. （オプション）レプリケーション関係の状態とステータスを確認します。

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

フェイルオーバーされたレプリケーション関係を再同期します。

再同期処理によってレプリケーション関係が再確立されます。再同期処理を実行すると、元のソースアプリケーションが実行中のアプリケーションになり、デスティネーションクラスタで実行中のアプリケーションに加えた変更は破棄されます。

このプロセスは、レプリケーションを再確立する前に、デスティネーションクラスタ上のアプリケーションを停止します。

重要	フェイルオーバー中にデスティネーションアプリケーションに書き込まれたデータはすべて失われます。

手順

1. オプション：ソースクラスタで、ソースアプリケーションのSnapshotを作成します。これにより、ソース

クラスタからの最新の変更がキャプチャされます。

2. デスティネーションクラスタで、AppMirrorRelationship CRファイル（など）を編集し `trident-protect-relationship.yaml`、spec.desiredStateの値をに変更します。 `Established`
3. CR ファイルを保存します。
4. CRを適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. フェイルオーバーされたアプリケーションを保護するためにデスティネーションクラスタで保護スケジュールを作成した場合は削除します。スケジュールが残っていると、ボリュームSnapshotが失敗します。

フェイルオーバーされたレプリケーション関係の逆再同期

フェイルオーバーされたレプリケーション関係を逆再同期すると、デスティネーションアプリケーションがソースアプリケーションになり、ソースがデスティネーションになります。フェイルオーバー中にデスティネーションアプリケーションに加えられた変更は保持されます。

手順

1. 元のデスティネーションクラスタで、AppMirrorRelationship CRを削除します。これにより、デスティネーションがソースになります。新しいデスティネーションクラスタに保護スケジュールが残っている場合は削除します。
2. レプリケーション関係を設定するには、元 々 その関係を反対側のクラスタに設定するために使用したCRファイルを適用します。
3. 新しいデスティネーション（元のソースクラスタ）に両方のAppVault CRSが設定されていることを確認します。
4. 反対側のクラスタにレプリケーション関係を設定し、逆方向の値を設定します。

アプリケーションのレプリケーション方向を反転

レプリケーション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、ソースとデスティネーションを交換しようとしています。

手順

1. ソースクラスタで、シャットダウンSnapshotを作成します。

CRを使用したシャットダウンスナップショットの作成

a. ソースアプリケーションの保護ポリシースケジュールを無効にします。

b. ShutdownSnapshot CRファイルを作成します。

i. カスタムリソース（CR）ファイルを作成し、という名前を付けます（例： `trident-protect-shutdownsnapshot.yaml`）。

ii. 次の属性を設定します。

- * metadata.name*:（*required*）カスタムリソースの名前。
- *spec.AppVaultRef *:(*required*)この値は、ソースアプリケーションのAppVaultのmetadata.nameフィールドと一致する必要があります。
- *spec.ApplicationRef *:(*required*)この値は、ソースアプリケーションCRファイルのmetadata.nameフィールドと一致する必要があります。

YAMLの例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. ファイルに正しい値を入力したら `trident-protect-shutdownsnapshot.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-namespace
```

CLIを使用したシャットダウンスナップショットの作成

a. シャットダウンスナップショットを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot> -n <application_namespace>
```

2. ソースクラスタで、シャットダウンSnapshotが完了したら、シャットダウンSnapshotのステータスを取得します。

```
kubectl get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o yaml
```

3. ソースクラスタで、次のコマンドを使用して* shutdownsnapshot.status.appArchivePath *の値を探し、フ

ファイルパスの最後の部分（basenameとも呼ばれます。最後のスラッシュのあとのすべての部分）を記録します。

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 次のように変更して、新しいデスティネーションクラスタから新しいソースクラスタへのフェイルオーバーを実行します。

メモ	フェイルオーバー手順のステップ2では、AppMirrorRelationship CRファイルにフィールドを含め、 `spec.promotedSnapshot`その値を上記の手順3で記録したベースネームに設定します。

5. の逆再同期の手順を実行し[フェイルオーバーされたレプリケーション関係の逆再同期]ます。
6. 新しいソースクラスタで保護スケジュールを有効にします。

結果

リバースレプリケーションが実行されると、次の処理が実行されます。

- 元のソースアプリのKubernetesリソースのスナップショットが作成されます。
- 元のソースアプリケーションのポッドは、アプリケーションのKubernetesリソースを削除することで正常に停止されます（PVCとPVはそのまま維持されます）。
- ポッドがシャットダウンされると、アプリのボリュームのスナップショットが取得され、レプリケートされます。
- SnapMirror関係が解除され、デスティネーションボリュームが読み取り/書き込み可能な状態になります。
- アプリのKubernetesリソースは、元のソースアプリがシャットダウンされた後に複製されたボリュームデータを使用して、シャットダウン前のスナップショットから復元されます。
- 逆方向にレプリケーションが再確立されます。

アプリケーションを元のソースクラスタにフェイルバックします

Trident 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、デスティネーションへのフェイルオーバーが完了した関係から開始し、次の手順を実行します。

- フェイルオーバー状態から開始します。
- レプリケーション関係を逆再同期します。

注意	通常の再同期操作は実行しないでください。フェイルオーバー中にデスティネーションクラスタに書き込まれたデータが破棄されます。

- レプリケーションの方向を逆にします。

手順

1. 手順を実行します[フェイルオーバーされたレプリケーション関係の逆再同期]。
2. 手順を実行します[アプリケーションのレプリケーション方向を反転]。

レプリケーション関係を削除する

レプリケーション関係はいつでも削除できます。アプリケーションレプリケーション関係を削除すると、2つの別 々 のアプリケーションが作成され、それらのアプリケーション間に関係がなくなります。

手順

1. 現在のディサイトクラスタで、AppMirrorRelationship CRを削除します。

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

Trident Protectを使用して以前のスナップショットにフェイルオーバーする

Trident Protectを使用して、レプリケートされたアプリケーションを以前のスナップショットにフェイルオーバーすることで、データが破損または侵害される前の時点からリカバリできます。デフォルトでは、フェイルオーバーは最後に転送されたスナップショットを使用します。

注意	古いスナップショットにフェイルオーバーすると、選択したスナップショットよりも新しいデスティネーション クラスタ上のスナップショットは無効になるため、ONTAP SnapMirrorがそれらのスナップショットを削除します。このデータ損失が許容できる場合にのみ、この手順でアノテーションを追加してください。

作業を開始する前に

- AppMirrorRelationshipが存在し、 `Established`状態であること。作成するには、"レプリケーション関係を設定"を参照してください。状態を確認するには、 `kubectl get amr -n <namespace> <relationship_name> -o=jsonpath='{.status}'`を実行します。
- ソースおよびデスティネーション クラスタは、SnapMirrorレプリケーションを使用するONTAPストレージ バックエンドを使用します。Azure NetApp Filesやその他の非ONTAPバックエンドはサポートされていません。"SnapMirrorレプリケーションの要件"を参照してください。
- ソースアプリケーションに、スナップショットスケジュール（ `snapshotRetention` が1より大きい）が存在します。

スナップショットを特定する

各スナップショットには完了時刻が設定されているため、任意の時点を選択できます。

手順

1. ソースクラスタで、スナップショットの一覧を表示します：

```
kubectl get snapshots -n my-app-namespace
```

2. フェイルオーバー先のスナップショットを見つけます。 `completionTimestamp`値を使用して、特定の時点を選択します。
3. そのスナップショットのアーカイブパスを取得します：

```
kubectl get snapshots <snapshot_name> -n my-app-namespace -o jsonpath
='{.status.appArchivePath}'
```

4. パスの最後の部分、つまり最後のスラッシュ以降のすべてを記録します。この部分は_basename_です。例えば：

```
20231114173500_schedule-20231114173500_f8df8bd3-a727-4a14-b954-
9666aa618b55
```

メモ	スナップショットカスタムリソースの名前ではなく、ベース名を使用してください。

結果

選択されたスナップショットのベース名が記録されます。

選択したスナップショットにフェイルオーバーする

昇格させるスナップショットを指定し、注釈を追加してデータ損失を確認します。

手順

1. デスティネーション クラスタで、AppMirrorRelationshipに注釈を追加します。注釈がない場合、フェイルオーバーでは最後に転送されたスナップショットのみを使用できます。

```
kubectl annotate appmirrorrelationship <relationship_name> -n my-app-
namespace protect.trident.netapp.io/allow-promotion-with-older-
snapshots="true"
```

2. AppMirrorRelationship CRファイルを編集します（例： `trident-protect-relationship.yaml`）：
   - *spec.desiredState*を `Promoted`に設定します。
   - [スナップショットを特定する]で記録したベース名に*spec.promotedSnapshot*を設定します。

     例：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: <relationship_name>
  namespace: <namespace>
  annotations:
```

```
    protect.trident.netapp.io/allow-promotion-with-older-snapshots:
"true"
spec:
  desiredState: Promoted
  promotedSnapshot: <basename>
```

3. CRを適用します。

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. 関係のステータスを確認してください：

```
kubectl get amr -n my-app-namespace <relationship_name> -o=jsonpath
='{.status}' | jq
```

結果

関係は `Promoted`状態に移行し、アプリケーションは選択したスナップショットのデータを使用してデスティネーション クラスタ上で実行されます。

関係が `Promoting`状態のままで、 `status.error`に `The snapshot provided is invalid or older than the most recently transferred snapshot`と表示される場合、アノテーションが欠落しているか、 `true`に設定されていません。アノテーションを追加して、もう一度試してください。

次の手順

元のクラスタに戻るには、"逆再同期"を実行し、その後"レプリケーションの方向を反転させる"。

注意	元のクラスタに戻すには、通常の再同期は実行しないでください。通常の再同期では、フェイルオーバーしたアプリケーションが破棄され、関係が最新のソーススナップショットに戻ります。

関連情報

- "Trident Protectを使用してアプリケーションのレプリケーションを設定および管理する"
- "デスティネーションクラスタへのフェイルオーバー"
- "フェイルオーバーされたレプリケーション関係を再同期します。"
- "アプリケーションを元のソースクラスタにフェイルバックします"
- "データ保護スケジュールを作成"

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。

メモ	アプリケーションを移行すると、そのアプリケーション用に構成されたすべての実行フックがアプリケーションとともに移行されます。リストア後の実行フックがある場合は、リストア処理の一環として自動的に実行されます。

バックアップとリストアの処理

次のシナリオでバックアップとリストアの処理を実行するには、特定のバックアップとリストアのタスクを自動化します。

同じクラスタにクローニング

アプリケーションを同じクラスタにクローニングするには、Snapshotまたはバックアップを作成し、同じクラスタにデータをリストアします。

手順

1. 次のいずれかを実行します。
   a. "Snapshot を作成します"です。
   b. "バックアップを作成します"です。
2. Snapshotとバックアップのどちらを作成したかに応じて、同じクラスタで次のいずれかを実行します。
   a. "スナップショットからデータをリストア"です。
   b. "バックアップからデータをリストア"です。

別のクラスタにクローニング

アプリケーションを別のクラスターに複製するには (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。

メモ	を使用して、異なるクラスタ間でアプリケーションをレプリケートできます"SnapMirrorレプリケーション"。

手順

1. "バックアップを作成します"です。
2. バックアップを含むオブジェクトストレージバケットのAppVault CRがデスティネーションクラスタで設定されていることを確認します。
3. デスティネーションクラスタで、"バックアップからデータをリストア"を実行します。

あるストレージクラスから別のストレージクラスへのアプリケーションの移行

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、次のようになります（リストアCRのシークレットを除く）。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
```

```
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    - destination: "${destinationNamespace}"
      source: "${sourceNamespace}"
  storageClassMapping:
    - destination: "${destinationStorageClass}"
      source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

CRを使用したスナップショットのリストア

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-snapshot-restore-cr.yaml`ます。
2. 作成したファイルで、次の属性を設定します。
   - * metadata.name*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - * spec.appArchivePath *：スナップショットの内容が格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o
     jsonpath='{.status.appArchivePath}'
     ```

   - * spec.appVaultRef *：（*required*）スナップショットコンテンツが格納されているAppVaultの名前。
   - * spec.namespaceMapping*:リストア処理のソースネームスペースとデスティネーションネームスペースのマッピング。および `my-destination-namespace`を、使用している環境の情報に置き換え `my-source-namespace`ます。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
     ```

```
    appArchivePath: my-snapshot-path
    appVaultRef: appvault-name
    namespaceMapping: [{"source": "my-source-namespace",
  "destination": "my-destination-namespace"}]
```

3. リストアするアプリケーションの特定のリソースのみを選択する必要がある場合は、特定のラベルが付いたリソースを含めるか除外するフィルタリングを追加します。

   - **resourceFilter.resourceSelectionCriteria**:(フィルタリングに必要) `include or exclude` resourceMatchersで定義されたリソースを含めるか除外するかを指定します。次のresourceMatchersパラメータを追加して、追加または除外するリソースを定義します。
     - **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した場合、それらはOR演算として照合され、各要素内のフィールド（グループ、種類、バージョン）はAND演算として照合されます。
       - *resourceMatchers[].group *:(*Optional*)フィルタリングするリソースのグループ。
       - *resourceMatchers[].kind *:(*optional*)フィルタリングするリソースの種類。
       - **resourceMatchers[].version**:(*Optional*)フィルタリングするリソースのバージョン。
       - * resourceMatchers[].names *：（*optional*）フィルタリングするリソースのKubernetes metadata.nameフィールドの名前。
       - *resourceMatchers[].namespaces *:(*optional*)フィルタリングするリソースのKubernetes metadata.nameフィールドの名前空間。
       - *resourceMatchers[].labelSelectors *:(*Optional*)で定義されているリソースのKubernetes metadata.nameフィールドのラベルセレクタ文字列 "Kubernetes のドキュメント"。例：`"trident.netapp.io/os=linux"`。

         例：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-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. ファイルに正しい値を入力したら `trident-protect-snapshot-restore-cr.yaml` 、CRを適用します。

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

CLIを使用したスナップショットのリストア

手順

1. スナップショットを別のネームスペースにリストアし、括弧内の値を環境の情報に置き換えます。

   - `snapshot`引数では、という形式のネームスペースとSnapshot名を使用し `<namespace>/<name>`ます。
   - `namespace-mapping`引数は、コロンで区切られた名前空間を使用して、ソース名前空間を正しい宛先名前空間に形式でマッピングし `source1:dest1,source2:dest2`ます。

     例：

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

KubeVirt 仮想マシンを Trident Protect で保護する

Trident Protectを使用すると、OpenShift Virtualization上で実行されているKubeVirt仮想マシン（VM）をバックアップおよびリストアできます。1つ以上のネームスペースにまたがるすべてのVMを一度に保護することも、 `includedVirtualMachines`フィールドを使用して特定のVMを名前で指定して保護することもできます。Trident Protectは、バックアップ時に関連するすべてのVMリソースを自動的に収集します。リストア時には、アーカイブ全体を復旧することも、特定のVMを選択して個別にリストアすることもできます。

KubeVirt 保護モード

Trident Protect は、保護する KubeVirt VM を定義する 2 つの方法をサポートしています。アプリケーションを定義する際に、いずれかのモードを選択してください。

モード	フィールド	説明
名前空間ベース	`includedNamespaces`	1つ以上の名前空間内の、一致するすべてのリソースを保護します。名前空間エントリで `labelSelector`を使用して、動的に含める仮想マシンを制御します。
VMスコープ	`includedVirtualMachines`	名前空間と名前で特定の KubeVirt VMを保護します。VMが複数の名前空間に分散している場合、またはすべての名前空間リソースを保護したくない場合は、このモードを使用してください。

必ず1つのフィールド（(`includedNamespaces`または `includedVirtualMachines`）を

`ApplicationSpec`に指定してください。同じアプリケーション内で両方を使用することはできません。

どちらのモードも追加のフィルタリングをサポートしています。

- `resourceFilter`を使用して、特定のリソースを含めるか除外するかを指定します。
- `includedClusterScopedResources`を使用して、 `StorageClass`オブジェクトなど、互換性のあるクラスタスコープのリソースを含めます。

メモ

- Trident Protect は、データ保護操作中に KubeVirt VM ファイルシステムをフリーズおよびアンフリーズして、一貫性を確保できます。設定の詳細については、"KubeVirt VMによるデータ保護"を参照してください。
- 仮想マシンのすべてのディスクを同じ時点のスナップショットとして保存するには、ボリュームグループのスナップショットを使用します。"Trident Protectでボリュームグループのスナップショットを使用する" を参照してください。

バックアップとスナップショット中に収集されたリソース

KubeVirt VM を含むアプリケーションのバックアップまたはスナップショットを作成する場合、Trident Protect は選択された VM リソースを収集し、それらに依存するリソースを自動的に検出します。

コレクションの動作は、次の2つの場合に適用されます。

- VM スコープのアプリケーションで `includedVirtualMachines`を使用するもの。
- 名前空間ベースのアプリケーションで、KubeVirt `VirtualMachine`リソースを含むもの。

VMスコープのコレクション

VMスコープのアプリケーションの場合、Trident Protectは、 `includedVirtualMachines`にリストされている各`VirtualMachine`から開始し、そのVMのリストアに必要な依存リソースを検出します。

リソースタイプ	Scope	発見方法
`VirtualMachine`	名前空間付き	`includedVirtualMachines`リストからの完全一致。
`VirtualMachineInstance`	名前空間付き	VMが実行中の場合は、VMと同じ名前空間と名前を使用します。
`VirtualMachineInstancetype`	名前空間付き	参照元 `spec.instancetype`（ `kind: VirtualMachineInstancetype`の場合）。
`VirtualMachineClusterInstancetype`	クラスタ	`kind`が未設定または `kind: VirtualMachineClusterInstancetype`の場合に、 `spec.instancetype`によって参照されます。
`VirtualMachinePreference`	名前空間付き	参照元 `spec.preference`（ `kind: VirtualMachinePreference`の場合）。
`VirtualMachineClusterPreference`	クラスタ	`kind`が未設定の場合、または `kind: VirtualMachineClusterPreference`の場合に、 `spec.preference`によって参照されます。
`DataVolume`	名前空間付き	参照元 `spec.template.spec.volumes[].dataVolume.name`。

リソースタイプ	Scope	発見方法
`PersistentVolumeClaim`	名前空間付き	DataVolume と同じ名前を使用する DataVolume バッキング PVC、 `persistentVolumeClaim.claimName`内の直接 PVC 参照、 `memoryDump.claimName`内のメモリダンプ PVC、および `ephemeral.persistentVolumeClaim.claimName`内の一時的な PVC から検出されます。
`PersistentVolume`	クラスタ	各収集済みPVCのバッキングPVから、 `spec.volumeName`を使用して検出されます。
`Secret`	名前空間付き	ボリュームシークレットから検出 `secret.secretName` ；containerDiskイメージのプルシークレット；CloudInitNoCloudおよびCloudInitConfigDriveのユーザーデータとネットワークデータのシークレット参照；Sysprepシークレット；KernelBootコンテナイメージのプルシークレット；およびAccessCredentialsのSSHまたはパスワードシークレット。
`ConfigMap`	名前空間付き	ボリューム ConfigMaps および Sysprep ConfigMaps の `configMap.name`から検出されました。
`ServiceAccount`	名前空間付き	`serviceAccount.serviceAccountName`のボリューム参照から検出されました。
`NetworkAttachmentDefinition`	名前空間付き	ネットワーク名に `/`が含まれていない場合、 `spec.template.spec.networks[].multus.networkName`内のMultusネットワークから検出されます。同一ネームスペースのNetworkAttachmentDefinition参照のみが収集されます。ネームスペース間の参照はスキップされます。
`Pod`	名前空間付き	ラベルセレクター `kubevirt.io=virt-launcher,vm.kubevirt.io/name=<vmName>`を使用して、 `virt-launcher` Podに対して検出されました。
ストレージクラス、Tridentボリューム、およびTridentバックエンド	クラスタ	仮想マシンに対して検出された PVC に基づいて、名前空間ベースのアプリケーションと同様の方法で収集されます。

メモ	VM スコープの名前空間リソースがすべて収集された後、Trident Protect は標準 `resourceFilter`を適用します（設定されている場合）。その後、クラスタスコープのリソース収集は通常どおり実行されます。

名前空間ベースのコレクション

名前空間ベースのアプリケーションの場合、Trident Protect は選択された名前空間をスキャンし、一致するすべてのリソースを収集します。KubeVirt `VirtualMachine` オブジェクトが見つかった場合、Trident Protect は、VMスコープのアプリケーションと同様に、インスタンスタイプ、設定、および `virt-launcher` Pod などの依存リソースを自動的に収集します。

これは、アプリケーションが定義されている場合でも、VM 固有のリソースがバックアップまたはスナップショットに含まれることを意味します `includedNamespaces`。

名前空間ベースのアプリケーション定義を使用して仮想マシンを保護する

名前空間ベースのアプリケーションを作成する場合、Trident Protect は選択された名前空間をスキャンし、KubeVirt VM を含むすべてのリソースを収集します。見つかった仮想マシンごとに、Trident Protect はその仮想マシンのバックアップと復元に必要な依存リソースも収集します。

名前空間ベースのアプリケーションCRの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
```

アプリケーションを作成したら、バックアップとスナップショットに標準の Trident Protect ワークフローを使用します：

- スナップショットYAMLの例については、"オンデマンドスナップショットを作成します"を参照してください。
- バックアップYAMLの例については、"オンデマンドバックアップの作成"を参照してください。

名前空間ベースの**VM**アプリケーションのリストア動作

名前空間ベースのアプリケーションは選択された名前空間からリソースを収集するため、バックアップまたはスナップショットにはKubeVirtランタイムオブジェクトとKubeVirtが管理するメタデータが含まれる可能性があります。これらのオブジェクトの中には、ソースクラスタの実行時状態を表しているため、直接復元すべきでないものもあります。

復元中、Trident Protect は KubeVirt 対応の変換を適用して、ソースクラスターのメタデータを削除し、一時的な KubeVirt リソースをスキップします。これにより、KubeVirt は復元された環境で VM をクリーンに再作成できます。

リソース	変換
VirtualMachine	`kubevirt.io/`で始まるラベルと注釈を削除します。
VirtualMachineInstance	`kubevirt.io/`で始まるラベルと注釈を削除します。
VirtualMachineInstanceMigration	無視され、復元もされません。
VirtualMachineSnapshot	無視され、復元もされません。
VirtualMachineSnapshotContent	無視され、復元もされません。
Pod (`virt-launcher`)	ポッドに `kubevirt.io=virt-launcher`ラベルがある場合はスキップされます。

メモ

- Trident Protect は、依存関係の問題を回避するために、KubeVirt リソースを定められた順序で復元します。CDI DataVolumes (`cdi.kubevirt.io`) は、VM オブジェクトが作成さ

れる前に VM ディスクデータが利用可能になるように、最初に復元されます。その後、VirtualMachines (`kubevirt.io`) が VirtualMachineInstances より先に復元されます。

- デフォルトでは、Trident Protectは復元された各VMの最初のネットワークインターフェイスからMACアドレスを削除します。保持するには、 `protect.trident.netapp.io/retain-mac-address`アノテーションを使用します。"高度なTrident Protect復元設定を使用する"を参照してください。

ラベルセレクタを使用した動的**VM**保護

各仮想マシン名を列挙せずに動的に仮想マシンを保護したい場合は、ラベルセレクタを使用した名前空間ベースのアプリケーション定義を使用してください。

VirtualMachine CR のみにラベルが必要です。DataVolumes、PVC、シークレット、ConfigMaps などの依存リソースは自動的に収集されます。

手順

1. 仮想マシン CR に共有ラベルを付ける：

```
kubectl label virtualmachine <vm_name 1> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_2> app=<label_value> -n
<vm_namespace>
```

2. アプリケーション CR を `includedNamespaces`および `labelSelector`で作成します：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
      labelSelector:
        matchLabels:
          app: <label_value>
```

3. 標準の Trident Protect ワークフローを使用してバックアップとリストアを作成します。

追加情報：

- ラベルを変更することで、いつでも仮想マシンを保護対象に追加または削除できます。

```
kubectl label virtualmachine <vm_name_to_add> app=<label_value> -n
<vm_namespace>
```

```
kubectl label virtualmachine <vm_name_to_remove> app
=<different_label_value> -n <vm_namespace>
```

- ラベルセレクタは複数の名前空間にわたって使用することもできます。

```
spec:
  includedNamespaces:
    - namespace: <vm_namespace_1>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
    - namespace: <vm_namespace_2>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
```

特定の仮想マシンを**includedVirtualMachines**で保護する

名前付きセットのVirtualMachinesを1つまたは複数の名前空間にわたって保護し、アーカイブから特定のVMを選択的にリストアする場合は、 `includedVirtualMachines`を使用します。

アプリケーションCRとリストアCRの両方で同じ `includedVirtualMachines`セレクター形式を使用します。

このセレクターは、さまざまな状況で使用できます。

- アプリケーションCRでは、どのVMが保護されるかを定義します。
- リストアCRでは、バックアップまたはスナップショット アーカイブからどのVMをリストアするかを定義します。

メモ	CRの復元時、 `includedVirtualMachines`は任意です。このフィールドを含めない場合、アーカイブ全体が復元されます。指定した場合、リストに記載されている仮想マシンとその依存リソースのみが復元されます。

以下のYAMLは、共有セレクタ構造を示しています。

```
includedVirtualMachines:
  - namespace: <vm_namespace_1>
    names:
      - <vm_name_1>
      - <vm_name_2>
  - namespace: <vm_namespace_2>
    names:
      - <vm_name_3>
```

メモ	複数のエントリー `includedVirtualMachines`は異なる名前空間を参照できるため、マルチ名前

空間のVM対応アプリケーションの定義とリストアが可能になります。

フィールド	を入力します	説明
namespace	文字列	対象となる仮想マシンオブジェクトを含むKubernetesネームスペース。
名前	文字列のリスト	その名前空間内の1つ以上の仮想マシン名。

VMベースのアプリケーションの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace_1>
      names:
        - <vm_name_1>
        - <vm_name_2>
    - namespace: <vm_namespace_2>
      names:
        - <vm_name_3>
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

特定の**KubeVirt VM**をリストアする

リストアCRで `includedVirtualMachines`を使用して、バックアップ、スナップショット、またはレプリケートされたスナップショットアーカイブから選択したKubeVirt VMのみをリストアできます。

`includedVirtualMachines`がリストア CR で指定されている場合、Trident Protect はリストされている VirtualMachines とその依存リソース（ `VirtualMachineInstance`、 `DataVolume`、 `PersistentVolumeClaim`、 `PersistentVolume`、 `Secret`、 `ConfigMap`、 `ServiceAccount`、およびその他の VM 関連オブジェクト）のみをリストアします。

リストア CR で `includedVirtualMachines`を指定しない場合、Trident Protect はリストア CR の設定に従ってアーカイブ全体をリストアします。

サポートされている復元**CR**と**CLI**コマンド

以下の表は、 `includedVirtualMachines`をサポートするリストアCRと、それらに対応するCLIコマンドの一覧です。

CRを復元する	CLIコマンド	説明
`BackupRestore`	`tridentctl-protect create backuprestore`	バックアップから、新規または異なる名前空間マッピングに復元します。サポート `namespaceMapping`および `storageClassMapping`。
`BackupInplaceRestore`	`tridentctl-protect create backupinplacerestore`	バックアップから元の名前空間または複数の名前空間に復元します。
`SnapshotRestore`	`tridentctl-protect create snapshotrestore`	スナップショットから、新規または異なる名前空間マッピングに復元します。サポート `namespaceMapping`および `storageClassMapping`。
`SnapshotInplaceRestore`	`tridentctl-protect create snapshotinplacerestore`	スナップショットから元の名前空間または複数の名前空間に復元します。
`ReplicateSnapshotRestore`	YAMLのみを使用するか、お使いの環境でサポートされているCLIコマンドが利用可能な場合はそれを使用してください	レプリケートされたスナップショットから復元します。`namespaceMapping`、 `storageClassMapping`、および `inPlaceRestore`フラグを使用したIn Placeリストアをサポートします。

メモ	Trident CLIには、リストア コマンドに `includedVirtualMachines`用のフラグはありません。 `tridentctl-protect`特定のVMをリストアするには、 `kubectl apply`を使用してリストアCR YAMLファイルを直接適用する必要があります。YAMLを一から作成するか、CLIで `--dry-run --output yaml`を使用して開始ファイルを生成し、 `spec`の下に `includedVirtualMachines`フィールドを追加してから適用します。

VMスコープのリストア動作

復元CRの `includedVirtualMachines`フィールドは、アプリケーションCRと同じセレクタ形式を使用します。

```
includedVirtualMachines:
  - namespace: <source_vm_namespace>
    names:
      - <vm_name_1>
      - <vm_name_2>
```

CR を復元する場合、 `includedVirtualMachines`の名前空間の値は、バックアップ、スナップショット、またはレプリケートされたスナップショット アーカイブに存在するソース名前空間を参照します。

使用する場合は `namespaceMapping`、Trident Protect はまずソースアーカイブから要求された仮想マシンを選択し、次にリストア中に名前空間マッピングを適用します。使用する場合は `storageClassMapping`、Trident Protect は、リストア変換フェーズ中にストレージクラスのマッピングを適用します。

CLIを使用してリストア用YAMLを生成する

`tridentctl-protect`を使用してベースリストアCRを生成し、YAMLを編集して`includedVirtualMachines`を追加できます。

```
tridentctl-protect create <restore_command> <restore_name> \
  <restore_options> \
  --dry-run --output yaml > <restore_name>.yaml
```

生成されたYAMLを編集し、`spec`の下に`includedVirtualMachines`フィールドを追加してから、ファイルを適用します：

```
kubectl apply -f <restore_name>.yaml
```

例：VMベースのアプリケーションを定義する

以下のアプリケーションは、2つの名前空間にまたがる3つの特定の仮想マシンを保護します。

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: production-vms
  namespace: prod
spec:
  includedVirtualMachines:
    - namespace: prod
      names:
        - app-server
        - database-server
    - namespace: monitoring
      names:
        - prometheus-vm
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

例：バックアップを作成する

VMベースのアプリケーションのバックアップを作成するには、標準のバックアップCRを使用します。バックアップCRには、VM固有のフィールドは必要ありません。

```
apiVersion: protect.trident.netapp.io/v1
```

```
kind: Backup
metadata:
  name: daily-backup
  namespace: prod
spec:
  applicationRef: production-vms
  appVaultRef: s3-vault
```

例：バックアップから単一の仮想マシンをリストアする

次の例では、バックアップ アーカイブから `database-server`のみをリストアします。アーカイブ内の他のVM（ `app-server`や `prometheus-vm`など）はスキップされます。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: restore-db-only
  namespace: prod-vms-restore
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  namespaceMapping:
    - source: prod
      destination: prod-dr
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f restore-db-only.yaml
```

`database-server`とその依存リソース（DataVolumes、PVC、PV、Secrets、ConfigMaps、および関連するKubeVirtリソースなど）のみがリストアされます。

例：バックアップから単一の **VM**を**In Place**リストアする

次の例では、 `database-server`のみを元の名前空間にバックアップ アーカイブからリストアします。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: ipr-db-only
```

```
  namespace: prod
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f ipr-db-only.yaml
```

例：スナップショットから単一の仮想マシンを復元する

次の例では、スナップショット アーカイブから `ubuntu-vm-blue-a627be38`のみをリストアします。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: blue-vm-snap-restore
  namespace: blue
spec:
  appVaultRef: my-appvault
  appArchivePath: snapshots/protectctl-blue-vm/snap-1
  includedVirtualMachines:
    - namespace: blue
      names:
        - ubuntu-vm-blue-a627be38
```

```
kubectl apply -f blue-vm-snap-restore.yaml
```

VMリストアと他のフィルターを組み合わせる

`includedVirtualMachines`を `resourceFilter`、 `namespaceMapping`、および `storageClassMapping`と同じリストアCRで組み合わせて使用することで、リストア対象の内容を細かく調整できます。

追加すると `resourceFilter`、Trident Protect はまず `includedVirtualMachines`にリストされている VM とその依存リソースを選択します。次に、リソースフィルターを適用して、結果に含めるものと除外するものをさらに絞り込みます。

```
spec:
```

```
  includedVirtualMachines:
    - namespace: <vm_namespace>
      names:
        - <vm_name>
  resourceFilter:
    resourceSelectionCriteria: Include
    resourceMatchers:
      - kinds: ["PersistentVolumeClaim"]
        names: ["<pvc_name>"]
```

`BackupRestore`、 `SnapshotRestore`、および `ReplicateSnapshotRestore`の場合、 `namespaceMapping`と `storageClassMapping`は `includedVirtualMachines`と連携して動作します。 `includedVirtualMachines`の名前空間参照は、アーカイブに表示されているソース名前空間に対応します。名前空間とストレージクラスのマッピングは、フィルタリングが完了した後のリストア変換フェーズ中に適用されます。

復元動作の詳細については、"Trident Protectを使用してアプリケーションを復元する"を参照してください。

CLIを使用して**VM**スコープのアプリケーションを作成する

`--virtual-machines`フラグを使用して、名前空間全体ではなく特定のKubeVirt VMにアプリケーションのスコープを絞り込みます。 `--virtual-machines`フラグは `--namespaces`と相互に排他的です。

手順

1. 以下のコマンドのいずれかを実行し、括弧内の値を環境情報に置き換えてアプリケーションを作成してください。 `--virtual-machines`フラグのフォーマットは `<namespace>(<vm_name>)`です。複数の仮想マシンを指定する場合は、括弧内にカンマ区切りの名前を使用してください。
   - 名前空間内の単一の仮想マシン：

     ```
     tridentctl-protect create application <application_name> \
       -n <application_namespace> \
       --virtual-machines "<vm_namespace>(<vm_name>)"
     ```

   - 名前空間をまたがる複数の仮想マシン：

     ```
     tridentctl-protect create application <application_name> \
       -n <application_namespace> \
       --virtual-machines "<vm_namespace_1>(<vm_name_1>,<vm_name_2>
     ),<vm_namespace_2>(<vm_name_3>)"
     ```

関連情報

- "管理用アプリケーションの定義"
- "アプリケーションの保護"
- "Trident Protectを使用してアプリケーションを復元する"
- "詳細リストア設定を使用する"
- "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 CLIプラグインをインストールする"
- "AppVaultオブジェクトを使用してバケットを管理する"

Trident Protect実行フックを管理する

実行フックは、管理対象アプリケーションのデータ保護操作と組み合わせて実行するように構成できるカスタムアクションです。たとえば、データベースアプリケーションがある場合、実行フックを使用して、スナップショットの前にすべてのデータベーストランザクションを一時停止し、スナップショットの完了後にトランザクションを再開できます。これにより、アプリケーションと整合性のある Snapshot を作成できます。

実行フックのタイプ

Trident Protect は、実行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。

- Snapshot前
- Snapshot後
- バックアップ前
- バックアップ後
- リストア後のPOSTコマンドです
- フェイルオーバー後

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

1. 適用可能なカスタムプリオペレーション実行フックは、適切なコンテナで実行されます。カスタムのプリオペレーションフックは必要なだけ作成して実行できますが、操作前のこれらのフックの実行順序は保証も構成もされていません。
2. 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. データ保護処理が実行されます。
4. フリーズされたファイルシステムは、該当する場合はフリーズ解除されます。
5. 適用可能なカスタムポストオペレーション実行フックは、適切なコンテナで実行されます。必要な数のカスタムポストオペレーションフックを作成して実行できますが、操作後のこれらのフックの実行順序は保証されず、設定もできません。

同じ種類の実行フック（スナップショット前など）を複数作成する場合、これらのフックの実行順序は保証されません。ただし、異なるタイプのフックの実行順序は保証されています。たとえば'以下は'すべての異なるタイプのフックを持つ構成の実行順序です

1. スナップショット前フックが実行されます
2. スナップショット後フックが実行されます
3. 予備フックが実行されます
4. バックアップ後のフックが実行されます

メモ	上記の順序の例は、既存のSnapshotを使用しないバックアップを実行する場合にのみ該当します。

メモ	本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にしたら、作成されたSnapshotとバックアップをテストして整合性があることを確認します。これを行うには、アプリケーションを一時的なネームスペースにクローニングし、スナップショットまたはバックアップをリストアしてから、アプリケーションをテストします。

メモ	スナップショット前の実行フックで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、および後続のリストア処理に含まれます。

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

- 実行フックは、スクリプトを使用してアクションを実行する必要があります。多くの実行フックは、同じスクリプトを参照できます。
- Trident Protect では、実行フックが使用するスクリプトを実行可能なシェル スクリプトの形式で記述する必要があります。
- スクリプトのサイズは96KBに制限されています。
- Trident Protect は、実行フックの設定と一致する基準を使用して、スナップショット、バックアップ、または復元操作に適用可能なフックを決定します。

メモ	実行フックは、実行中のアプリケーションの機能を低下させたり、完全に無効にしたりすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。実行フックが関連付けられている状態でバックアップまたはスナップショット操作を開始した後'キャンセルした場合でも'バックアップまたはスナップショット操作がすでに開始されていればフックは実行できますつまり、バックアップ後の実行フックで使用されるロジックは、バックアップが完了したとは見なされません。

実行フックフィルタ

アプリケーションの実行フックを追加または編集するときに、実行フックにフィルタを追加して、フックが一致するコンテナを管理できます。フィルタは、すべてのコンテナで同じコンテナイメージを使用し、各イメージを別の目的（Elasticsearchなど）に使用するアプリケーションに便利です。フィルタを使用すると、一部の同一コンテナで実行フックが実行されるシナリオを作成できます。1つの実行フックに対して複数のフィルタ

を作成すると、それらは論理AND演算子と結合されます。実行フックごとに最大10個のアクティブフィルタを使用できます。

実行フックに追加する各フィルターは、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" 。

メモ	リストアまたはクローン処理のあとに実行される実行フックにネームスペースフィルタを追加し、リストアまたはクローンのソースとデスティネーションが異なるネームスペースにある場合、ネームスペースフィルタはデスティネーションネームスペースにのみ適用されます。

実行フックの例

にアクセスして "NetApp Verda GitHubプロジェクト" 、Apache CassandraやElasticsearchなどの一般的なアプリケーションの実際の実行フックをダウンロードします。また、独自のカスタム実行フックを構築するための例やアイデアを得ることもできます。

メモ	Verdaリポジトリのスクリプトは現状のまま提供されており、NetAppによる公式サポートの対象外です。NetAppのサポート対象は、Trident Protect内の実行フックフレームワークのみです。

実行フックの作成

Trident Protect を使用して、アプリのカスタム実行フックを作成できます。実行フックを作成するには、所有者、管理者、またはメンバーの権限が必要です。

CRの使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-hook.yaml`ます。
2. Trident 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*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.applicationRef *:(*required*)実行フックを実行するアプリケーションのKubernetes名。
   - *spec.stage *:(*required*)実行フックが実行されるアクションのステージを示す文字列。有効な値：
     - 前
     - 投稿
   - *spec.action *:(*required*)指定された実行フックフィルタが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値：
     - スナップショット
     - バックアップ
     - リストア

        - フェイルオーバー
    - *spec.enabled *:(*Optional*)この実行フックが有効か無効かを示します。指定しない場合、デフォルト値はtrueです。
    - **spec.hookSource**:(*required*) base64でエンコードされたフックスクリプトを含む文字列。
    - *spec.timeout *:(_Optional _)実行フックの実行を許可する時間を分単位で定義する数値。最小値は1分で、指定しない場合のデフォルト値は25分です。
    - * spec.arguments *:(_Optional _)実行フックに指定できる引数のYAMLリスト。
    - *spec.matchingCriteria *:(*Optional*)実行フックフィルタを構成する各ペアの基準キー値ペアのオプションリスト。実行フックごとに最大10個のフィルタを追加できます。
    - *spec.matchingCriteria.type *:(*Optional*)実行フックフィルタタイプを識別する文字列。有効な値：
        - コンテナイメージ
        - コンテナ名
        - ポッド名
        - PodLabel
        - ネームスペース名
    - *spec.matchingCriteria.value *:(*Optional*)実行フックフィルタ値を識別する文字列または正規表現。

    YAMLの例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
```

```
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. CRファイルに正しい値を入力したら、CRを適用します。

```
kubectl apply -f trident-protect-hook.yaml
```

CLI を使用します

手順

1. 実行フックを作成し、括弧内の値を環境からの情報に置き換えます。例：

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

実行フックを手動で実行する

テスト目的で、または失敗後にフックを手動で再実行する必要がある場合は、実行フックを手動で実行できます。実行フックを手動で実行するには、Owner、Admin、またはMemberの権限が必要です。

実行フックを手動で実行するには、次の2つの基本ステップがあります。

1. リソースのバックアップを作成します。リソースを収集してバックアップを作成し、フックの実行場所を決定します。
2. バックアップに対して実行フックを実行する

▼ 手順**1**：リソースのバックアップを作成する

CRの使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-resource-backup.yaml`ます。
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*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
- * spec.applicationRef *：（*required*）リソースのバックアップを作成するアプリケーションのKubernetes名。
- * spec.appVaultRef *：（*required*）バックアップコンテンツが格納されているAppVaultの名前。
- **spec.appArchivePath**:バックアップコンテンツが格納されているAppVault内のパス。このパスを検索するには、次のコマンドを使用します。

```
kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

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 を使用します

手順

1. バックアップを作成します。角かっこ内の値は、使用している環境の情報に置き換えます。例：

```
tridentctl protect create 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。この例のコマンドは、処理が完了するまで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の使用

手順

1. カスタムリソース（CR）ファイルを作成し、という名前を付け `trident-protect-hook-run.yaml`ます。
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*:（*required*）このカスタムリソースの名前。環境に適した一意の適切な名前を選択します。
   - *spec.applicationRef *:(*required*)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 *:(*required*)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 *:(*required*)指定された実行フックフィルタが一致すると仮定して、実行フックが実行するアクションを示す文字列。有効な値：
     - スナップショット
     - バックアップ
     - リストア
     - フェイルオーバー
   - *spec.stage *:(*required*)実行フックが実行されるアクションのステージを示す文字列。このフックランは、他のステージではフックを実行しません。有効な値：
     - 前
     - 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 を使用します

手順

1. 手動実行フック実行要求を作成します。

```
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. 実行フック実行のステータスを確認します。このコマンドは、処理が完了するまで繰り返し実行できます。

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. exechooksrunオブジェクトについて説明し、最終的な詳細とステータスを確認します。

```
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。

   メモ　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10月16日:"Kubernetes向けの高度なストレージソリューション"
- 2025年8月19日:"データの一貫性の強化: 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: "NetApp ASA r2 システムのパワーを Kubernetes ブロック ストレージ向けに解放"
- 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05日: "『Unlock Seamless iSCSI Storage Integration：A Guide to FSxN on ROSA Clusters for AWS』"
- 2025年2月27日: "Trident、GKE、 Google Cloud NetApp Volumesを使用したクラウド ID の導入"
- 2024年12月12日: "Tridentでのファイバ・チャネル・サポートの導入"
- 2024年11月11日: "NetApp TridentとGoogle Cloud NetApp Volumes"
- 2024年10月29日: "Tridentを使用したAmazon FSx for NetApp ONTAPと Red Hat OpenShift Service on AWS (ROSA) 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08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01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03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仮想化VMのcrash-consistentバックアップおよびリストア処理"
- 2025年3月11日："NetApp Tridentを使用してGitOpsパターンをアプリケーションデータ保護に拡張"
- 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の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:"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

ここでは、Tridentのインストール、設定、アップグレード、トラブルシューティングに関するFAQを紹介します。

一般的な質問

Tridentはどのくらいの頻度でリリースされますか。

24.02リリース以降、Tridentは2月、6月、10月の4カ月ごとにリリースされます。

Tridentは、特定のバージョンの**Kubernetes**でリリースされたすべての機能をサポートしていますか。

Tridentは通常、Kubernetesのアルファ機能をサポートしていません。Trident は、 Kubernetes ベータリリースに続く 2 つの Trident リリースでベータ機能をサポートしています。

Tridentの機能に関して、他の**NetApp**製品との依存関係はありますか。

Tridentは、他のNetAppソフトウェア製品との依存関係はなく、スタンドアロンアプリケーションとして機能します。ただし、ネットアップのバックエンドストレージデバイスが必要です。

Trident構成の完全な詳細情報を取得するにはどうすればよいですか。

Trident構成に関する詳細情報を取得するには、コマンドを使用し `tridentctl get`ます。

Tridentでのストレージのプロビジョニング方法に関する指標は取得できますか。

はい。管理対象のバックエンド数、プロビジョニングされたボリューム数、消費されたバイト数など、Trident処理に関する情報を収集するために使用できるPrometheusエンドポイント。を監視および分析に使用することもできます"Cloud Insights の機能です"。

Tridentを**CSI**プロビジョニングツールとして使用すると、ユーザエクスペリエンスは変化しますか？

いいえ。ユーザーエクスペリエンスと機能に関しては変更はありません。使用されるプロビジョニングツール名はです `csi.trident.netapp.io`。現在および将来のリリースで提供されるすべての新機能を使用する場合は、この方法でTridentをインストールすることを推奨します。

Kubernetesクラスタへの**Trident**のインストールと使用

Tridentはプライベートレジストリからのオフラインインストールをサポートしていますか。

はい、Tridentはオフラインでインストールできます。"Tridentのインストールについて"を参照してください。

Tridentをリモートでインストールできますか。

はい。Trident 18.10以降では、クラスタにアクセスできる任意のマシンからのリモートインストール機能がサポートされます `kubectl`。アクセスが確認されたら `kubectl`（たとえば、リモートマシンからコマンドを実

行し `kubectl get nodes`て確認する）、インストール手順に従います。

Tridentでハイアベイラビリティを構成できますか。

Tridentはインスタンスが1つのKubernetesデプロイメント（ReplicaSet）としてインストールされるため、HAが組み込まれています。デプロイメント内のレプリカの数を増やすことはできません。Tridentがインストールされているノードが失われた場合やポッドにアクセスできない場合、Kubernetesはポッドをクラスタ内の正常なノードに自動的に再導入します。Tridentはコントロールプレーンのみであるため、現在マウントされているポッドはTridentを再導入しても影響を受けません。

Tridentは**kube-system**ネームスペースにアクセスする必要がありますか。

TridentはKubernetes APIサーバから読み取り、アプリケーションが新しいPVCを要求するタイミングを判断するため、kube-systemへのアクセスが必要になります。

Tridentで使用されるロールと**Privileges**を教えてください。

Tridentインストーラは、KubernetesのClusterRoleを作成します。これにより、KubernetesクラスタのPersistentVolume、PersistentVolumeClaim、StorageClass、およびSecretリソースへの特定のアクセス権が付与されます。"tridentctlのインストールをカスタマイズします"を参照してください。

Tridentがインストールに使用するマニフェストファイルをローカルで生成できますか。

必要に応じて、Tridentがインストールに使用する正確なマニフェストファイルをローカルで生成および変更できます。"tridentctlのインストールをカスタマイズします"を参照してください。

2つの独立した**Kubernetes**クラスタで、**2**つの独立した**Trident**インスタンスで同じ**ONTAP**バックエンド**SVM**を共有できますか。

推奨されませんが、2つのTridentインスタンスに同じバックエンドSVMを使用できます。インストール時に各インスタンスに一意のボリューム名を指定するか、ファイルで一意のパラメータを `setup/backend.json`指定し `StoragePrefix`ます。これは、両方のインスタンスに同じFlexVol volumeが使用されないようにするためです。

Tridentを**ContainerLinux(**旧**CoreOS)**にインストールすることはできますか**?**

Tridentは単なるKubernetesポッドであり、Kubernetesが実行されている場所にインストールできます。

NetApp Cloud Volumes ONTAPで**Trident**を使用できますか。

はい。Tridentは、AWS、Google Cloud、Azureでサポートされています。

トラブルシューティングとサポート

NetAppは**Trident**をサポートしていますか。

Tridentはオープンソースで無料で提供されていますが、NetAppバックエンドがサポートされていれば、NetAppは完全にサポートしています。

サポートケースを作成するにはどうすればよいですか？

サポートケースを作成するには、次のいずれかを実行します。

1. サポートアカウントマネージャーに連絡して、チケットの発行に関するサポートを受けてください。
2. 連絡してサポートケースを作成します "ネットアップサポート"。

サポートログバンドルを生成するにはどうすればよいですか？

tridentctl logs-a を実行して ' サポートバンドルを作成できますバンドルでキャプチャされたログに加えて、kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

問題を作成し "Trident Github の利用"、問題の件名と説明に*RFE*を記載します。

不具合を発生させる場所

で問題を作成し "Trident Github の利用"ます。問題に関連する必要なすべての情報とログを記録しておいてください。

Tridentに関する簡単な質問があり、説明が必要な場合はどうなりますか？コミュニティやフォーラムはありますか？

ご質問、問題、ご要望がございましたら、TridentまたはGitHubからお問い合わせ"チャネルを外します"ください。

ストレージシステムのパスワードが変更され、**Trident**が機能しなくなりました。どうすれば回復できますか？

バックエンドのパスワードを `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。交換してください `myBackend` この例では、バックエンド名にとを指定しています `/path/to_new_backend.json` と入力します `backend.json` ファイル。

Tridentで**Kubernetes**ノードが見つかりません。この問題を解決するにはどうすればよいですか

TridentがKubernetesノードを検出できない可能性があるシナリオは2つあります。Kubernetes または DNS 問題内のネットワーク問題が原因の場合もあります。各 Kubernetes ノードで実行される Trident ノードのデーモンが Trident コントローラと通信し、 Trident にノードを登録できる必要があります。この問題は、Tridentのインストール後にネットワークの変更が発生した場合、クラスタに追加された新しいKubernetesノードでのみ発生します。

Trident ポッドが破損すると、データは失われますか？

Trident ポッドが削除されても、データは失われません。TridentのメタデータはCRDオブジェクトに格納されます。Trident によってプロビジョニングされた PVS はすべて正常に機能します。

Tridentのアップグレード

古いバージョンから新しいバージョンに直接アップグレードできますか（いくつかのバージョンはスキップします）？

NetAppでは、Tridentをあるメジャーリリースから次のメジャーリリースにアップグレードできます。バージョン 18.xx から 19.xx 、 19.xx から 20.xx にアップグレードできます。本番環境の導入前に、ラボでアップグレードをテストする必要があります。

Trident を以前のリリースにダウングレードできますか。

アップグレード、依存関係の問題、またはアップグレードの失敗または不完全な実行後に見つかったバグの修正が必要な場合は、そのバージョンに固有の手順を使用して以前のバージョンを再インストールする必要があります"Tridentのアンインストール"。これは、以前のバージョンにダウングレードするための唯一の推奨方法です。

バックエンドとボリュームを管理

ONTAPバックエンド定義ファイルに管理**LIF**とデータ**LIF**の両方を定義する必要がありますか。

管理LIFは必須です。DataLIFの種類：

- ONTAP SAN：iSCSIには指定しないでください。Tridentは、を使用して"ONTAP の選択的LUNマップ"、マルチパスセッションの確立に必要なiSCI LIFを検出します。が明示的に定義されている場合は、警告が生成され `dataLIF`ます。詳細については、を参照してください "ONTAP のSAN構成オプションと例" 。
- ONTAP NAS：NetAppでは指定を推奨しています `dataLIF`。指定しない場合、TridentはSVMからデータLIFをフェッチします。NFSのマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定すると、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散できます。詳細は、を参照してください。"ONTAP NASの設定オプションと例"

Tridentは**ONTAP**バックエンド用に**CHAP**を構成できますか。

はい。Tridentは、ONTAPバックエンドに対して双方向CHAPをサポートしています。これには、バックエンド構成での設定が必要です `useCHAP=true`。

Tridentを使用してエクスポートポリシーを管理するにはどうすればよいですか。

Tridentでは、バージョン20.04以降でエクスポートポリシーを動的に作成および管理できます。これにより、ストレージ管理者はバックエンド構成に 1 つ以上の CIDR ブロックを指定でき、 Trident では、その範囲に含まれるノード IP を作成したエクスポートポリシーに追加できます。このようにして、Tridentは、所定のCIDR内にIPを持つノードのルールの追加と削除を自動的に管理します。

管理**LIF**とデータ**LIF**に**IPv6**アドレスを使用できますか。

Tridentは次のIPv6アドレスの定義をサポートします

- `managementLIF` および `dataLIF` ONTAP NASバックエンドの場合：
- `managementLIF` ONTAP SANバックエンドの場合：を指定することはできません `dataLIF` ONTAP SANバックエンドの場合：

TridentをIPv6で機能させるには、フラグ（インストール用 `tridentctl`）、（Tridentオペレータ用）、`IPv6`または（`Helm`インストール用） `tridentTPv6`を使用してインストールする必要があります `--use-ipv6`。

バックエンドの管理 **LIF** を更新できますか。

はい。 tridentctl update backend コマンドを使用してバックエンド管理 LIF を更新できます。

バックエンドの**DataLIF**を更新できますか。

DataLIFの更新は、および `ontap-nas-economy`でのみ実行できます `ontap-nas`。

Trident for Kubernetesで複数のバックエンドを作成できますか。

Tridentは、同じドライバでも異なるドライバでも、多数のバックエンドを同時にサポートできます。

Tridentはバックエンドクレデンシャルをどのように保存しますか。

Tridentは、バックエンドクレデンシャルをKubernetesシークレットとして保存します。

Tridentはどのようにして特定のバックエンドを選択しますか。

バックエンド属性を使用してクラスに適切なプールを自動的に選択できない場合は 'toragePools' パラメータと additionalStoragePools' パラメータを使用して ' 特定のプールセットを選択します

Tridentが特定のバックエンドからプロビジョニングされないようにするにはどうすればよいですか。

パラメータを `excludeStoragePools`使用して、Tridentがプロビジョニングに使用する一連のプールをフィルタリングし、に一致するプールをすべて削除します。

同じ種類のバックエンドが複数ある場合、**Trident**はどのようにして使用するバックエンドを選択しますか。

同じタイプの設定済みバックエンドが複数ある場合、Tridentはおよび `PersistentVolumeClaim`のパラメータに基づいて適切なバックエンドを選択します `StorageClass`。たとえば、ONTAP - NASドライバのバックエンドが複数ある場合、Tridentは、およびの `PersistentVolumeClaim`パラメータを照合し、および `PersistentVolumeClaim`に記載されている要件を提供できるバックエンドを `StorageClass`照合し `StorageClass`ます。要求に一致するバックエンドが複数ある場合、Tridentはそのうちの1つをランダムに選択します。

Tridentは**Element / SolidFire**で双方向**CHAP**をサポートしていますか。

はい。

Tridentでは、どのようにして**ONTAP**ボリュームに**qtree**を導入しますか。**1** つのボリュームに配置できる **qtree** の数はいくつですか。

`ontap-nas-economy`ドライバは、同じ`FlexVol volume`に最大`200`個の`qtree`（`50~300`の間で設定可能）、クラスタノードあたり`100,000`個、クラスタあたり`2.4M`個の`qtree`を作成します。エコノミードライバによって処理される新しいを入力すると、`PersistentVolumeClaim`新しい`qtree`に対応できる`FlexVol volume`がすでに存在するかどうかが確認されます。`qtree`に対応する`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 でプロビジョニングされたボリュームに **UNIX** アクセス権を設定するにはどうすればよいですか。

Tridentによってプロビジョニングされるボリュームに対してUNIX権限を設定するには、バックエンド定義ファイルにパラメータを設定します。

ボリュームをプロビジョニングする際に、明示的な **ONTAP NFS** マウントオプションを設定するにはどうすればよいですか。

Tridentでは、Kubernetesではデフォルトでマウントオプションがどの値にも設定されません。Kubernetesストレージクラスでマウントオプションを指定するには、次の例を参照して"こちらをご覧ください"ください。

プロビジョニングしたボリュームを特定のエクスポートポリシーに設定するにはどうすればよいですか？

適切なホストにボリュームへのアクセスを許可するには、バックエンド定義ファイルに設定されている「 exportPolicy 」パラメータを使用します。

ONTAPを使用した**Trident**によるボリューム暗号化の設定方法を教えてください。

Trident によってプロビジョニングされたボリュームで暗号化を設定するには、バックエンド定義ファイルの暗号化パラメータを使用します。詳細については、以下を参照してください。"TridentとNVEおよびNAEとの連携"

Tridentを使用して**ONTAP**の**QoS**を実装する最良の方法はどれですか。

ONTAP の QoS を実装するには、「 torageClasses 」を使用します。

Tridentでシンプロビジョニングまたはシックプロビジョニングを指定するにはどうすればよいですか。

ONTAP ドライバは、シンプロビジョニングまたはシックプロビジョニングをサポートします。ONTAP ドライバはデフォルトでシンプロビジョニングに設定されています。シックプロビジョニングが必要な場合は、バックエンド定義ファイルまたは「 torageClass 」を設定する必要があります。両方が設定されている場合は、「 torageClass 」が優先されます。ONTAP で次の項目を設定します。

1. 'S torageClass' で 'provisioningType' 属性を thick に設定します
2. バックエンド定義ファイルで 'backend spaceReserve パラメータを volume に設定して ' シックボリュームを有効にします

誤って **PVC** を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes では、バージョン 1.10 以降、 PVC 保護が自動的に有効になります。

Tridentで作成された**NFS PVC**を拡張できますか。

はい。Tridentによって作成されたNFS PVCを拡張できます。Trident自動拡張ポリシーは、容量しきい値に基づいてPVCを自動的に拡張できます。ONTAPボリュームの自動サイズ調整は別個の機能であり、サポートされているONTAP SANドライバに適用されます：PVCのサイズを変更せずに、バックエンドのFlexVolを拡張します。詳細については、"ONTAPボリュームの自動サイズ調整"を参照してください。

ボリュームが **SnapMirror** データ保護（ **DP** ）モードまたはオフラインモードの間にインポートできますか。

外部ボリュームが DP モードになっているかオフラインになっている場合、ボリュームのインポートは失敗します。次のエラーメッセージが表示されます。

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

リソースクォータをネットアップクラスタに変換する方法

Kubernetes ストレージリソースクォータは、ネットアップストレージの容量があるかぎり機能します。容量不足が原因でNetAppストレージがKubernetesクォータ設定に対応できない場合、Tridentはプロビジョニングを試行しますがエラーが発生します。

Tridentを使用してボリューム**Snapshot**を作成できますか。

はい。Tridentでは、オンデマンドのボリュームSnapshotとSnapshotからの永続的ボリュームの作成がサポートされています。スナップショットからPVSを作成するには、フィーチャーゲートが有効になっていることを確認し `VolumeSnapshotDataSource`ます。

Tridentボリュームスナップショットをサポートするドライバを教えてください。

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` 、 そして `azure-netapp-files`バックエンドドライバー。

Tridentで**ONTAP**を使用してプロビジョニングされたボリュームの**Snapshot**バックアップを作成する方法を教えてください。

これは 'ONTAP-NAS' 'ONTAP-SAN' および 'ONTAP-NAS-flexgroup ドライバで利用できますFlexVol レベルでは「 ONTAP-SAN-エコノミー 」ドライバに「スナップショットポリシー」を指定することもできます。

これはドライバでも使用できますが、qtreeレベルではなく、FlexVol volumeレベルで使用でき `ontap-nas-economy`ます。Tridentでプロビジョニングされたボリュームの Snapshotを作成できるようにするには、backendパラメータオプションを、ONTAPバックエンドで定義されている目的のSnapshotポリシーに設定し `snapshotPolicy`ます。ストレージコントローラで作成されたSnapshotは、Tridentでは認識されません。

Tridentでプロビジョニングされたボリュームに**Snapshot**リザーブの割合を設定できますか。

はい。バックエンド定義ファイルで属性を設定することで、Tridentを使用してSnapshotコピーを格納するために特定の割合のディスクスペースをリザーブできます `snapshotReserve`。を設定し、 `snapshotReserve`バックエンド定義ファイルでスナップショット予約の割合が設定されている場合は `snapshotPolicy`、バックエンドファイルで指定されている割合に従って設定され `snapshotReserve`ます。パーセンテージ番号が指定されていない場合 `snapshotReserve`、ONTAPはデフォルトでスナップショット予約のパーセンテージを5とします。この `snapshotPolicy`オプションをnoneに設定すると、Snapshotリザーブの割合は0に設定されます。

ボリュームの **Snapshot** ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。バックエンド定義ファイルで「 snapmirror directionDir 」パラメータを設定することで、 Trident によってプロビジョニングされたボリューム上のスナップショットディレクトリにアクセスできます。

Tridentを使用してボリューム用に**SnapMirror**を設定できますか。

現時点では、 SnapMirror は ONTAP CLI または OnCommand System Manager を使用して外部に設定する必要があります。

永続ボリュームを特定の **ONTAP Snapshot** にリストアするにはどうすればよいですか？

ボリュームを ONTAP Snapshot にリストアするには、次の手順を実行します。

1. 永続ボリュームを使用しているアプリケーションポッドを休止します。
2. ONTAP CLI または OnCommand システムマネージャを使用して、必要な Snapshot にリバートします。
3. アプリケーションポッドを再起動します。

Tridentは、負荷共有ミラーが設定されている**SVM**でボリュームをプロビジョニングできますか。

負荷共有ミラーは、NFS経由でデータを提供するSVMのルートボリューム用に作成できます。ONTAP は、Tridentによって作成されたボリュームの負荷共有ミラーを自動的に更新します。ボリュームのマウントが遅延する可能性があります。Tridentを使用して複数のボリュームを作成する場合、ボリュームをプロビジョニングする方法は、負荷共有ミラーを更新するONTAP によって異なります。

お客様 / テナントごとにストレージクラスの使用状況を分離するにはどうすればよいですか。

Kubernetes では、ネームスペース内のストレージクラスは使用できません。ただし、 Kubernetes を使用すると、ネームスペースごとにストレージリソースクォータを使用することで、ネームスペースごとに特定のストレージクラスの使用量を制限できます。特定のストレージへのネームスペースアクセスを拒否するには、そのストレージクラスのリソースクォータを 0 に設定します。

トラブルシューティング

Tridentのインストールおよび使用中に発生する可能性のある問題のトラブルシューティングには、ここに示すポインタを使用してください。

メモ	Tridentに関するサポートが必要な場合は、を使用してサポートバンドルを作成し `tridentctl logs -a -n trident`、NetAppサポートに送信してください。

全般的なトラブルシューティング

- Trident ポッドが正常に起動しない場合（たとえば、 Trident ポッドが 2 つ未満の「 ContainerCreating 」フェーズで停止した場合）、「 kubectl-n trident 」を実行して、展開が trident 、「 kubectl-n trident 」が pod trident- を記述します -** 追加のインサイトを提供できます。kubelet ログの取得 (journalctl -xeu kubelet など) も役立ちます。
- Trident ログに十分な情報がない場合は、インストールオプションに基づいてインストールパラメータに「 -d 」フラグを渡して、 Trident のデバッグモードを有効にしてみてください。

次に './tridentctl logs -n trident` を使用して debug が設定され ' ログ内で 'level=debug msg' を検索していることを確認します

オペレータとともにインストールされます

```
kubectl patch torc trident -n <namespace> --type=merge -p
'{"spec":{"debug":true}}'
```

すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには 'kubectl get pod -n trident' の出力の 'age' 列を確認します

Trident 20.07および20.10では、の代わりに `torc`使用して `tprov`ください。

Helm とともにインストールされます

```
helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
tridentDebug=true`
```

tridentctl を使用してインストールされます

```
./tridentctl uninstall -n trident
./tridentctl install -d -n trident
```

- バックエンド定義にを含めることで、各バックエンドのデバッグログを取得することもできます `debugTraceFlags`。たとえば、TridentログでAPI呼び出しとメソッドトラバーサルを取得するためにを指定します `debugTraceFlags: {"api":true, "method":true,}`。既存のバックエンドはで構成 `tridentctl backend update`できます `debugTraceFlags`。
- Red Hat Enterprise Linux CoreOS（RHCOS）を使用している場合は、がワーカーノードで有効になっていて、デフォルトで起動されていることを確認してください `iscsid`。この設定には、 OpenShift MachineConfig を使用するか、イグニッションテンプレートを変更します。
- Tridentを "Azure NetApp Files の特長"で使用する際に発生する可能性のある一般的な問題は、テナントシークレットとクライアントシークレットが、権限が不十分なアプリ登録から取得された場合です。Trident要件の完全なリストについては、"Azure NetApp Files の特長"構成を参照してください。
- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl status rpcbind' またはそれに相当する処理を実行することで確認できます
- Trident バックエンドが、以前に作業したことがあるにもかかわらず「 failed 」状態であると報告した場合は、バックエンドに関連付けられている SVM/admin クレデンシャルの変更が原因である可能性があります。「 tridentctl update backend 」または Trident ポッドのバウンスを使用してバックエンド情報を更新すると、この問題は修正されます。
- Docker をコンテナランタイムとして Trident をインストールするときに権限の問題が発生した場合は、「 --in cluster=false」 フラグを付けて Trident のインストールを試みてください。これはインストーラポッドを使用せず、「 trident-installer 」ユーザのために発生する許可の問題を回避します。
- 実行に失敗した後のクリーンアップには 'uninstall パラメータ <Uninstalling Trident > を使用しますデフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全

にアンインストールしてインストールできます。

- 以前のバージョンのTridentにダウングレードする場合は、 `tridentctl uninstall` Tridentを削除するコマンド。必要なをダウンロードします "Trident のバージョン" を使用してをインストールします `tridentctl install` コマンドを実行します
- インストールが成功した後、 PVC が「保留中」段階で停止した場合、「 kubectl 」を実行して PVC を記述すると、 Trident がこの PVC の PV のプロビジョニングに失敗した理由を追加情報に提供できます。

オペレータを使用したTridentの導入に失敗

オペレータを使用して Trident を導入する場合 'TridentOrchestrator のステータスは 'Installing から Installed に変わります'Failed' ステータスが表示され ' オペレータがそれ自体で回復できない場合は ' 次のコマンドを実行してオペレータのログを確認する必要があります

```
tridentctl logs -l trident-operator
```

trident-operator コンテナのログの末尾には、問題のある場所を示すことができます。たとえば、このような問題の 1 つは、エアーギャップ環境のアップストリームレジストリから必要なコンテナイメージをプルできないことです。

Trident のインストールが失敗した理由を理解するには、「 TridentOrchestrator 」のステータスを確認する必要があります。

```
kubectl describe torc trident-2
Name:          trident-2
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   trident.netapp.io/v1
Kind:          TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:          <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                       Trident is bound to another CR 'trident'
```

```
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                          Message
  ----     ------  ----               ----                          -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、 Trident のインストールに使用された「 TridentOrchestrator 」がすでに存在することを示します。各 Kubernetes クラスタは Trident のインスタンスを 1 つしか保持できないため、オペレータは任意の時点で作成可能なアクティブな TridentOrchestrator が 1 つだけ存在することを確認します。

また、 Trident ポッドのステータスを確認することで、適切でないものがあるかどうかを確認できます。

```
kubectl get pods -n trident

NAME                                READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                   1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw        4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                   1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                   1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv   1/1     Running            0
8m17s
```

1 つ以上のコンテナイメージがフェッチされなかったため、ポッドが完全に初期化できないことがわかります。

この問題に対処するには、「 TridentOrchestrator 」 CR を編集する必要があります。また、「 TridentOrchestrator 」を削除して、変更された正確な定義を持つ新しいものを作成することもできます。

Tridentの導入に失敗しました `tridentctl`

何が問題になったかを特定するために、インストーラをもう一度「 -d`` 」引数を使用して実行すると、デバッグモードが有効になり、問題の内容を理解するのに役立ちます。

```
./tridentctl install -n trident -d
```

問題を解決した後 ' 次のようにインストールをクリーンアップし 'tridentctl install コマンドを再度実行できます

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```

TridentとCRDを完全に取り外します。

Trident、作成されたCRD、および関連するカスタムリソースをすべて完全に削除できます。

警告	この操作は元に戻すことはできません。Tridentを完全に新規にインストールする場合を除き、この操作は行わないでください。CRDを削除せずにTridentをアンインストールするには、を参照してください"Trident をアンインストールします"。

Trident オペレータ

Tridentオペレータを使用してTridentをアンインストールし、CRDを完全に削除するには、次の手順に従います。

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

Helm

Helmを使用してTridentをアンインストールし、CRDを完全に削除するには：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>tridentctl</code>

を使用してTridentをアンインストールした後にCRDを完全に削除するには `tridentctl`

```
tridentctl obliviate crd
```

RWX rawブロックネームスペースo Kubernetes 1.26でNVMeノードのステージング解除が失敗する

Kubernetes 1.26を実行している場合、RWX rawブロックネームスペースでNVMe/TCPを使用すると、ノードのステージング解除が失敗することがあります。次のシナリオは、障害に対する回避策を提供します。または、Kubernetesを1.27にアップグレードすることもできます。

ネームスペースとポッドが削除されました

Tridentで管理されるネームスペース（NVMeの永続的ボリューム）がポッドに接続されているシナリオを考えてみましょう。ネームスペースをONTAPバックエンドから直接削除すると、ポッドを削除しようとすると、ステージング解除プロセスが停止します。このシナリオは、Kubernetesクラスタやその他の機能には影響しません。

回避策

該当するノードから永続的ボリューム（そのネームスペースに対応するボリューム）をアンマウントして削除します。

ブロックされたデータ**LIF**

```
 If you block (or bring down) all the dataLIFs of the NVMe Trident
backend, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```

.回避策

すべての機能を復元するには、`dataLIFS`を起動します。

ネームスペースマッピングが削除され

```
 If you remove the `hostNQN` of the worker node from the corresponding
subsystem, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```

.回避策

を追加します `hostNQN` サブシステムに戻ります。

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

ネットアップでは、さまざまな方法で Trident をサポートしています。ナレッジベース（KB）記事やDiscordチャネルなど、24時間365日利用可能な無料のセルフサポートオプションをご用意しています。

Tridentのサポートライフサイクル

Tridentでは、バージョンに応じて3つのレベルのサポートを提供しています。を参照してください "定義に対するNetAppソフトウェアバージョンのサポート"。

フルサポート
: Tridentは、リリース日から12か月間フルサポートを提供します。

限定サポート
: Tridentでは、リリース日から13~24カ月目に限定的なサポートを提供しています。

セルフサポート
: Tridentのマニュアルは、リリース日から25～36カ月目に入手できます。

バージョン	全面的なサポート終了	限定的なサポートの終了	セルフサポート終了
"26.10"	2027年10月	2028年10月	2029年10月
"26.06"	2027年7月	2028年7月	2029年7月
"26.02"	2027年2月	2028年2月	2029年2月
"25.10"	2026年10月	2027年10月	2028年10月
"25.06"	2026年6月	2027年6月	2028年6月
"25.02"	2026年2月	2027年2月	2028年2月
"24.10"	—	2026年10月	2027年10月
"24.06"	—	2026年6月	2027年6月
"24.02"	—	2026年2月	2027年2月
"23.10"	—	—	2026年10月
"23.07"	—	—	2026年7月
"23.04"	—	—	2026年4月

セルフサポート

トラブルシューティング記事の包括的なリストについては、 "ネットアップナレッジベース（ログインが必要）"を参照してください。

コミュニティサポート

コンテナユーザ（Trident開発者を含む）の活発なパブリックコミュニティがネットアップにあります"チャネルを外します"。プロジェクトに関する一般的な質問をしたり、同じような気のある同僚と関連するトピックについて話し合うのには、この場所が最適です。

NetAppテクニカルサポート

Tridentのサポートが必要な場合は、を使用してサポートバンドルを作成し `tridentctl logs -a -n trident`、に送信してください `NetApp Support <Getting Help>`。

を参照してください。

- "Tridentのリソース"
- "Kubernetes ハブ"

参照

Tridentポート

Tridentが通信に使用するポートの詳細については、こちらを参照してください。

概要

Tridentは、Kubernetesクラスタ内およびストレージバックエンドとの通信にさまざまなポートを使用します。以下は、主要なポート、その目的、およびセキュリティに関する考慮事項の概要です。

- アウトバウンドフォーカス：Kubernetesノード（コントローラとワーカー）は主にストレージLIF/IPへのトラフィックを開始するため、iptablesルールではこれらのポート上のノードIPから特定のストレージIPへのアウトバウンドを許可する必要があります。広範な「any-to-any」ルールは避けてください。
- 受信制限：内部Tridentポートをクラスタ内部トラフィックに制限します（たとえば、CalicoなどのCNIを使用）。ホストファイアウォール上で不要な受信露出はありません。
- プロトコルセキュリティ：
  - 可能な場合は TCP を使用します（信頼性が高くなります）。
  - 機密の場合はiSCSIにCHAP/IPsecを有効にし、管理にはTLS/HTTPSを有効にします（ポート443/8443）。
  - NFSv4（Tridentのデフォルト）の場合、必要ない場合はUDP/古いNFSv3ポート（例：4045～4049）を削除します。
  - 信頼できるサブネットに制限し、Prometheus（オプションのポート8001）などのツールを使用して監視します。

コントローラノードのポート

これらのポートは主にTridentオペレータ（バックエンド管理）用です。すべての内部ポートはポッドレベルです。ホストファイアウォールがCNIに干渉する場合にのみノードで許可します。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 8000	受信/送信（クラスタ内部）	Trident RESTサーバー（オペレーターとコントローラー間の通信）	すべて	ポッドCIDRに制限、外部への公開はありません。
TCP 8443	受信/送信（クラスタ内部）	バックチャネル HTTPS（安全な内部 API）	すべて	TLS暗号化。使用する場合はKubernetesサービスメッシュに制限されます。
TCP 8001	受信（クラスタ内部、オプション）	Prometheusメトリクス	すべて	監視ツール（RBACの使用など）にのみ公開し、使用しない場合は無効にします。
TCP 443	アウトバウンド	HTTPSからONTAP SVM/クラスタ管理LIF	ONTAP（すべて）、ANF	TLS証明書の検証が必要です。管理LIF IPのみに制限します。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 8443	アウトバウンド	HTTPSからEシリーズWeb Services Proxy	Eシリーズ（iSCSI）	デフォルトREST API；証明書を使用します。バックエンドYAMLで構成可能です。

ワーカーノードのポート

これらのポートは、CSI ノードデーモンセットとポッドマウント用です。データポートはストレージデータ LIF へのアウトバウンドです。NFSv3 を使用する場合は NFSv3 エクストラを含めます（NFSv4 の場合はオプション）。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
TCP 17546	受信（ポッドへのローカル）	CSI ノードの liveness / readiness プローブ	すべて	設定可能（--probe-port）；ホストの競合がないことを確認する；ローカルのみ。
TCP 8000	受信/送信（クラスタ内部）	Trident REST サーバ	すべて	上記と同様、pod-internal。
TCP 8443	受信/送信（クラスタ内部）	バックチャネル HTTPS	すべて	上記の通りです。
TCP 8001	受信（クラスタ内部、オプション）	Prometheusメトリクス	すべて	上記の通りです。
TCP 443	アウトバウンド	HTTPSからONTAP SVM/クラスタ管理LIF	ONTAP（すべて）、ANF	上記と同様、検出に使用されます。
TCP 8443	アウトバウンド	HTTPSからEシリーズWeb Services Proxy	Eシリーズ（iSCSI）	上記の通りです。
TCP/UDP 111	アウトバウンド	RPCBIND／portmapper	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	v3では必須、v4（ファイアウォールオフロード）ではオプション、NFSv4のみを使用する場合は制限されます。
TCP/UDP 2049	アウトバウンド	NFSデーモン	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	コアデータ、よく知られている、信頼性のためにTCPを使用。
TCP/UDP 635	アウトバウンド	マウントデーモン	ONTAP-NAS（NFSv3/v4）、ANF（NFS）	マウント。双方向コールバックが可能です（必要に応じて着信の一時コールを許可します）。
UDP 4045	アウトバウンド	NFSロックマネージャー（nlockmgr）	ONTAP-NAS（NFSv3）	ファイルロック。v4（pNFSハンドル）の場合はスキップ。UDP のみ。

ポート / プロトコル	送受信方向	目的	ドライバ/プロトコル	セキュリティに関する注意事項
UDP 4046	アウトバウンド	NFSステータスモニター（statd）	ONTAP-NAS（NFSv3）	通知。コールバックには受信一時ポート（1024～65535）が必要になる場合があります。
UDP 4049	アウトバウンド	NFSクォータデーモン（rquotad）	ONTAP-NAS（NFSv3）	クォータ。v4の場合はスキップします。
TCP 3260	アウトバウンド	iSCSIターゲット（検出/データ/CHAP）	ONTAP-SAN（iSCSI）、Eシリーズ（iSCSI）	既知のポート。このポートでCHAP 認証を実行。セキュリティのために相互 CHAP を有効にします。
TCP 445	アウトバウンド	SMB / CIFS	ONTAP-NAS（SMB）、ANF（SMB）	よく知られている、暗号化されたSMB3を使用する（Tridentアノテーション netapp.io/smb-encryption=true）。
TCP/UDP 88（オプション）	アウトバウンド	Kerberos認証	ONTAP（NFS/SMB/iSCSI と Kerb）	Kerberos を使用する場合（デフォルトではない）、ストレージではなく AD サーバーに接続します。
TCP/UDP 389（オプション）	アウトバウンド	LDAP	ONTAP（LDAP を使用した NFS/SMB）	同様に、名前解決/認証の場合、ADに制限します。

メモ	活性/レディネスプローブポートは、を使用して設置するときに変更できます `--probe-port` フラグ。このポートがワーカーノード上の別のプロセスで使用されていないことを確認することが重要です。

Trident REST API

"tridentctl コマンドとオプション"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を使用する状況

REST APIは、Kubernetes以外の環境でTridentをスタンドアロンバイナリとして使用する高度なインストールに役立ちます。

セキュリティを強化するため、ポッド内で実行する場合、Tridentは `REST API`デフォルトでlocalhostに制限されています。この動作を変更するには、ポッド構成でTridentの引数を設定する必要があり `-address`ます。

REST APIを使用する

これらのAPIの呼び出し方法の例については、debug)(`-d`フラグを渡します。詳細については、を参照してください "tridentctlを使用したTridentの管理"。

API は次のように機能します。

取得

GET <trident-address>/trident/v1/<object-type>

そのタイプのすべてのオブジェクトを一覧表示します。

GET <trident-address>/trident/v1/<object-type>/<object-name>

指定したオブジェクトの詳細を取得します。

投稿（ **Post** ）

POST <trident-address>/trident/v1/<object-type>

指定したタイプのオブジェクトを作成します。

- オブジェクトを作成するには JSON 構成が必要です。各オブジェクトタイプの仕様については、を参照してください"tridentctlを使用したTridentの管理"。
- オブジェクトがすでに存在する場合、動作は一定ではありません。バックエンドが既存のオブジェクトを更新しますが、それ以外のすべてのオブジェクトタイプで処理が失敗します。

削除

DELETE <trident-address>/trident/v1/<object-type>/<object-name>

指定したリソースを削除します。

メモ	バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、を参照してください "tridentctlを使用したTridentの管理"。

コマンドラインオプション

Tridentでは、Tridentオーケストレーションツールのコマンドラインオプションがいくつか公開されています。これらのオプションを使用して、導入環境を変更できます。

ロギング

-debug

デバッグ出力を有効にします。

-loglevel <level>

ロギングレベル（debug、info、warn、error、fatal）を設定します。デフォルトは info です。

Kubernetes

-k8s_pod

このオプションまたは `-k8s_api_server` をクリックしてKubernetesのサポートを有効にしこれを設定すると、 Trident はポッドの Kubernetes サービスアカウントのクレデンシャルを使用して API サーバに接続します。これは、サービスアカウントが有効になっている Kubernetes クラスタで Trident がポッドとして

実行されている場合にのみ機能します。

-k8s_api_server <insecure-address:insecure-port>

このオプションまたはを使用し `-k8s_pod`て、Kubernetesのサポートを有効にします。Trident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、Tridentをポッドの外部に導入できますが、サポートされるのはAPIサーバへの安全でない接続のみです。安全に接続するには、オプションを使用してポッドにTridentを導入し `-k8s_pod`ます。

Docker です

-volume_driver <name>

Dockerプラグインの登録時に使用するドライバ名。デフォルトはです `netapp`。

-driver_port <port-number>

UNIXドメインソケットではなく、このポートでリッスンします。

-config <file>

必須。バックエンド構成ファイルへのパスを指定する必要があります。

REST

-address <ip-or-host>

TridentのRESTサーバがリスンするアドレスを指定します。デフォルトは localhost です。localhost で聞いて Kubernetes ポッド内で実行しているときに、 REST インターフェイスにポッド外から直接アクセスすることはできません。使用 `-address ""` RESTインターフェイスにポッドのIPアドレスからアクセスできるようにするため。

注意	Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

-port <port-number>

TridentのRESTサーバがリスンするポートを指定します。デフォルトは 8000 です。

-rest

RESTインターフェイスを有効にします。デフォルトは true です。

Kubernetes オブジェクトと Trident オブジェクト

リソースオブジェクトの読み取りと書き込みを行うことで、 REST API を使用して Kubernetes や Trident を操作できます。Kubernetes と Trident 、 Trident とストレージ、 Kubernetes とストレージの関係を決定するリソースオブジェクトがいくつかあります。これらのオブジェクトの中には Kubernetes で管理されるものと Trident で管理されるものがあります。

オブジェクトは相互にどのように相互作用しますか。

おそらく、オブジェクト、その目的、操作方法を理解する最も簡単な方法は、 Kubernetes ユーザからのスト

レージ要求を 1 回だけ処理することです。

1. ユーザーは、「 PersistentVolumeClaim 」を作成して、特定のサイズの新しい「 PersistentVolume 」を、管理者が以前に設定した Kubernetes の「 torageClass 」から要求します。
2. Kubernetes の「 torageClass 」は、 Trident をプロビジョニングツールとして識別し、要求されたクラスのボリュームのプロビジョニング方法を Trident に指示するパラメータを含んでいます。
3. Trident は、対応する「 Backends 」と「 toragePools 」を識別する同じ名前の「 S torageClass 」を参照します。この名前は、このクラスのボリュームのプロビジョニングに使用できます。
4. Trident は、対応するバックエンドにストレージをプロビジョニングし、 2 つのオブジェクトを作成します。 Kubernetes では、「 PersistentVolume 」とは、ボリュームを検索、マウント、処理する方法を Kubernetes に伝える「 PersistentVolume 」と、「 PersistentVolume 」と実際のストレージの関係を保持する Trident 内のボリュームです。
5. Kubernetes は 'PersistentVolumeClaim を新しい 'PersistentVolume' にバインドしますPersistentVolume が実行される任意のホストに PersistentVolume をマウントする 'PersistentVolumeClaim を含むポッド。
6. ユーザーは、 Trident を指す「 VolumeSnapshotClass 」を使用して、既存の PVC の「 VolumeSnapshot 」を作成します。
7. Trident が PVC に関連付けられているボリュームを特定し、バックエンドにボリュームの Snapshot を作成します。また ' スナップショットの識別方法を Kubernetes に指示する 'VolumeSnapshotContent' も作成します
8. ユーザーは 'VolumeSnapshot' をソースとして使用して 'PersistentVolumeClaim を作成できます
9. Trident は必要なスナップショットを識別し、「 PersistentVolume 」と「 Volume 」の作成に関連する一連のステップを実行します。

ヒント	Kubernetes オブジェクトの詳細については、を参照することを強く推奨します "永続ボリューム" Kubernetes のドキュメントのセクション。

Kubernetes `PersistentVolumeClaim` オブジェクト

Kubernetes の「 PersistentVolumeClaim 」オブジェクトは、 Kubernetes クラスタユーザが作成したストレージの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN、solidfire-san-エコノミー 構成、solidfire-san-SAN間にあるSolidFireを実現します
trident.netapp.io/cloneFromPVC	cloneSourceVolume の実行中です	ontap-nas、ontap-san、solidfire-san、azure-netapp-files、ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ONTAP - NAS 、 ONTAP - SAN
trident.netapp.io/protocol	プロトコル	任意
trident.netapp.io/exportPolicy	エクスポートポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAP-SAN
trident.netapp.io/snapshotReserve	Snapshot リザーブ	ontap-nas、ontap-nas-flexgroup 、ontap-san
trident.netapp.io/snapshotDirectory	snapshotDirectory の略	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/blockSize	ブロックサイズ	solidfire - SAN
trident.netapp.io/skipリカバリキュー	リカバリキューをスキップ	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup、ontap-san、ontap-san-economy

作成された PV に「削除」再利用ポリシーがある場合、 Trident は PV が解放されると（つまり、ユーザーが PVC を削除したときに）、 PV と元のボリュームの両方を削除します。削除操作が失敗した場合、 Trident は PV をマークします。そのような状態で操作が成功するか、 PV が手動で削除されるまで、定期的に再試行します。PV が「 + Retain + 」ポリシーを使用している場合、 Trident はそのポリシーを無視し、管理者が Kubernetes とバックエンドからクリーンアップすると想定します。これにより、ボリュームを削除する前にバックアップまたは検査を行うことができます。PV を削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST API (tridentctl') を使用して削除してください。

Trident では CSI 仕様を使用したボリュームスナップショットの作成がサポートされています。ボリュームスナップショットを作成し、それをデータソースとして使用して既存の PVC のクローンを作成できます。これにより、 PVS のポイントインタイムコピーを Kubernetes にスナップショットの形で公開できます。作成した Snapshot を使用して新しい PVS を作成できます。「 +On-Demand Volume Snapshots + 」を見て、これがどのように機能するかを確認してください。

Tridentが提供するのも `cloneFromPVC` および `splitOnClone` クローンを作成するためのアノテーションこれらの注釈を使用して、CSI実装を使用せずにPVCのクローンを作成できます。

例 : ユーザがすでに「 m ysql 」という PVC を持っている場合、ユーザは「 trident.netapp.io/cloneFromPVC: mysql 」などの注釈を使用して「 m ysqlclone 」という新しい PVC を作成できます。このアノテーションセットを使用すると、 Trident はボリュームをゼロからプロビジョニングするのではなく、 MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- NetAppでは、アイドル状態のボリュームをクローニングすることを推奨
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- また 'ONTAP-NAS' および 'ONTAP-SAN' ドライバを使用すると 'pvc 注釈 trident.netapp.io/splitOnClone` を trident.netapp.io/cloneFromPVC` と組み合わせて設定することが望ましい場合がありますTrident は 'trident.netapp.io/splitOnClone` を true に設定した場合 ' クローン・ボリュームを親ボリュームからスプリットするため ' 一部のストレージ効率を失うことなく ' クローン・ボリュームのライフサイクルを親ボリュームから完全に分離しますtrident.netapp.io/splitOnClone` を設定したり 'false に設定したりしないと ' 親ボリュームとクローンボリューム間の依存関係を作成する代わりに ' バックエンドでのスペース消費が削減されますこれにより ' クローンを最初に削除しない限り ' 親ボリュームを削除できなくなりますクロー

ンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリュームのサイズが大きく異なっており、 ONTAP ではストレージ効率化のメリットはありません。

。 `sample-input` Directoryには、Tridentで使用するPVC定義の例が含まれています。を参照してください をクリックして、Tridentボリュームに関連付けられているパラメータと設定の完全な概要を確認します。

Kubernetes `PersistentVolume` オブジェクト

Kubernetes の 'PersistentVolume' オブジェクトは 'Kubernetes クラスタで利用できるようになったストレージの一部ですポッドに依存しないライフサイクルがあります。

メモ	Trident は 'PersistentVolume' オブジェクトを作成し ' プロビジョニングするボリュームに基づいて自動的に Kubernetes クラスタに登録します自分で管理することは想定されていません。

Trident をベースとする「 torageClass 」を参照する PVC を作成すると、 Trident は対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しい PV を登録します。プロビジョニングされたボリュームと対応する PV の構成では、 Trident は次のルールに従います。

- Trident は、 Kubernetes に PV 名を生成し、ストレージのプロビジョニングに使用する内部名を生成します。どちらの場合も、名前がスコープ内で一意であることが保証されます。
- ボリュームのサイズは、 PVC で要求されたサイズにできるだけ近いサイズに一致しますが、プラットフォームによっては、最も近い割り当て可能な数量に切り上げられる場合があります。

Kubernetes `StorageClass` オブジェクト

Kubernetes の「 torageClass 」オブジェクトは、「 PersistentVolumeClaims 」内の名前によって指定され、一連のプロパティを持つストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Trident を使用する Kubernetes の「 torageClass 」オブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters: <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、クラスのボリュームのプロビジョニング方法を Trident に指示します。

ストレージクラスのパラメータは次のとおりです。

属性	を入力します	必須	説明
属性（ Attributes ）	[string] 文字列をマップします	いいえ	後述の「属性」セクションを参照してください
ストレージプール	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
AdditionalStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
excludeStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、azure-netapp-files、ontap-san-economy	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ

属性	を入力します	値	提供	リクエスト	でサポートされます
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ontap-nas 、ontapさん、solidfireさん
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ontap-nas 、ontapさん、solidfireさん
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、ONTAP-NAS-エコノミー 、ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

^1 ^ ： ONTAP Select システムではサポートされていません

ほとんどの場合、要求された値はプロビジョニングに直接影響します。たとえば、シックプロビジョニングを要求した場合、シックプロビジョニングボリュームが使用されます。ただし、 Element ストレージプールでは、提供されている IOPS の最小値と最大値を使用して、要求された値ではなく QoS 値を設定します。この場合、要求された値はストレージプールの選択のみに使用されます。

理想的には ' 属性だけを使用して ' 特定のクラスのニーズを満たすために必要なストレージの特性をモデル化できますTrident は ' 指定した属性の *ALL* に一致するストレージ・プールを自動的に検出して選択します

「 attributes 」を使用してクラスに適切なプールを自動的に選択できない場合は、「 toragePools 」および「 additionalStoragePools 」パラメータを使用してプールをさらに改良したり、特定のプールセットを選択したりできます。

'toragePools' パラメータを使用すると ' 指定した属性に一致するプールのセットをさらに制限できますつまり 'attributes' パラメータと 'toragePools' パラメータで指定されたプールの交点をプロビジョニングに使用しますどちらか一方のパラメータを単独で使用することも、両方を同時に使用することも

「 additionalStoragePools 」パラメータを使用すると、「 attributes 」パラメータと「 toragePools 」パラメータで選択されたプールに関係なく、 Trident がプロビジョニングに使用するプールのセットを拡張できます。

excludeStoragePools' パラメータを使用して、 Trident がプロビジョニングに使用するプールのセットをフィルタリングできます。このパラメータを使用すると、一致するプールがすべて削除されます。

'toragePools' パラメータと 'additionalStoragePools' パラメータでは ' 各エントリは '<backend>:<storagePoolList>' の形式で指定したバックエンドのストレージプールのカンマ区切りリストですたとえば、「 additionalStoragePools 」の値は「 ontapnas_192.168.1.100 ： aggr1 、 aggr2 ； solidfire_192.168.1.101 ： bronze 」のようになります。これらのリストでは、バックエンド値とリスト値の

両方に正規表現値を使用できます。tridentctl get backend を使用してバックエンドとそのプールのリストを取得できます

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に Trident が選択するストレージプール / バックエンドには影響しません。代わりに、 Kubernetes Persistent Volume でサポートされるパラメータを提供するだけです。ワーカーノードはファイルシステムの作成操作を担当し、 xfsprogs などのファイルシステムユーティリティを必要とする場合があります。

属性	を入力します	値	説明	関連するドライバ	Kubernetes のバージョン
FSstype （英語）	文字列	ext4、ext3、xfs	ブロックボリュームのファイルシステムのタイプ	solidfire-san-group、ontap/nas、ontap -nas-エコノミー、ontap -nas-flexgroup 、ontap -san 、ONTAP - SAN -経済性	すべて
allowVolumeExpansion の略	ブール値	true false	PVC サイズの拡張のサポートをイネーブルまたはディセーブルにします	ontap-nas 、ontap-nas-economy、ontap -nas-flexgroup 、ontap-san 、ontap-san-economy、solidfire-san、azure-netapp-files	1.11 以上
volumeBindingMode のようになりました	文字列	即時、WaitForFirstConsumer	ボリュームバインドと動的プロビジョニングを実行するタイミングを選択します	すべて	1.19～1.26

ヒント

- 。 `fsType` パラメータは、SAN LUNに必要なファイルシステムタイプを制御する場合に使用します。また、Kubernetesでは、の機能も使用されます `fsType` ファイルシステムが存在することを示すために、ストレージクラスに格納します。ボリューム所有権は、を使用して制御できます `fsGroup` ポッドのセキュリティコンテキスト（使用する場合のみ） `fsType` が設定されます。を参照してください "Kubernetes ：ポッドまたはコンテナのセキュリティコンテキストを設定します" を使用したボリューム所有権の設定の概要については、を参照してください `fsGroup` コンテキスト（Context）。Kubernetesでが適用されます `fsGroup` 次の場合のみ値を指定します
  - 「 fsType 」はストレージクラスで設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 NFS エクスポートにはファイルシステムがすでに存在します。fsGroup を使用するには ' ストレージ・クラスで fsType を指定する必要がありますこの値は 'NFS' に設定することも ' ヌル以外の任意の値に設定することもできます

- を参照してください "ボリュームを展開します" ボリューム拡張の詳細については、を参照してください。
- Trident インストーラバンドルには、「 `sample -input/storageclass-*.yaml 」で Trident で使用するストレージクラス定義の例がいくつか用意されています。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

Kubernetes `VolumeSnapshotClass` オブジェクト

Kubernetes 'VolumeSnapshotClass' オブジェクトは 'S torageClasses' に似ていますこの Snapshot コピーは、複数のストレージクラスの定義に役立ちます。また、ボリューム Snapshot によって参照され、Snapshot を必要な Snapshot クラスに関連付けます。各ボリューム Snapshot は、単一のボリューム Snapshot クラスに関連付けられます。

スナップショットを作成するには 'VolumeSnapshotClass' を管理者が定義する必要がありますボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

「 driver 」は、「 csi-snapclass 」クラスのボリュームスナップショットの要求が Trident によって処理される Kubernetes を指定します。「要素ポリシー」は、スナップショットを削除する必要がある場合に実行されるアクションを指定します。「削除ポリシー」が「削除」に設定されている場合、 Snapshot を削除すると、ボリューム Snapshot オブジェクトおよびストレージクラスタ上の基盤となる Snapshot は削除されます。または、「 Retain 」に設定すると、「 VolumeSnapshotContent 」と物理スナップショットが保持されます。

Kubernetes `VolumeSnapshot` オブジェクト

Kubernetes の VolumeSnapshot オブジェクトは ' ボリュームのスナップショットを作成する要求ですPVC がボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームスナップショット要求が受信されると、 Trident はバックエンドでのボリュームのスナップショット作成を自動的に管理し、ユニークな「 VolumeSnapshotContent 」オブジェクトを作成することによってスナップショットを公開します。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。

メモ

VolumeSnapshot のライフサイクルはソース PVC から独立しています。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 はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes `VolumeSnapshotContent` オブジェクト

Kubernetes の「 VolumeSnapshotContent 」オブジェクトは、すでにプロビジョニングされているボリュームから取得されたスナップショットを表します。これは「 PersistentVolume 」と似ており、ストレージ・クラスタ上でプロビジョニングされた Snapshot を表します。「 PersistentVolumeClaim 」および「 PersistentVolume 」オブジェクトと同様に、スナップショットが作成されると、「 VolumeContent Snapshot」 オブジェクトは「 VolumeSnapshot 」オブジェクトへの 1 対 1 のマッピングを保持します。これは、スナップショットの作成を要求しました。

「 VolumeSnapshotContent 」オブジェクトには、スナップショットを一意に識別する詳細（「 napshotHandle 」など）が含まれています。この「 napshotHandle 」は、 PV の名前と「 VolumeSnapshotContent 」オブジェクトの名前を組み合わせた一意のものです。

Trident では、スナップショット要求を受信すると、バックエンドにスナップショットが作成されます。スナップショットが作成されると、 Trident は「 VolumeSnapshotContent 」オブジェクトを構成し、そのスナップショットを Kubernetes API に公開します。

メモ	通常、オブジェクトを管理する必要はありませ `VolumeSnapshotContent`ん。ただし、Trident の外部でを作成する場合は例外です"ボリュームSnapshotのインポート"。

`VolumeGroupSnapshotClass`Kubernetesオブジェクト

Kubernetes `VolumeGroupSnapshotClass`オブジェクトはに似てい `VolumeSnapshotClass`ます。これらは複数のストレージクラスを定義するのに役立ち、ボリュームグループスナップショットによって参照され、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、以下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
```

はTridentによって処理されます。に実行するアクションを指定します。 `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。または、に設定する `Retain`と、 `VolumeGroupSnapshotContent`物理Snapshotが保持されます。

`VolumeGroupSnapshot`Kubernetes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ための要求です。PVCがユーザーによるボリュームへ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、ユーザーによる既存のPVCのスナップショットを作成するための要求です。

ボリュームグループ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 を作成するときにスナップショットを DataSource として使用できます。

メモ	VolumeGroupSnapshot のライフサイクルはソース PVC とは独立しています。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 はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。ボリュームグループスナップショットは、関連するすべてのスナップショットが削除されると削除されます。

`VolumeGroupSnapshotContent`Kubernetes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、に似て `PersistentVolume`おり、ストレージクラスタにプロビジョニングされたSnapshotを示します。オブジェクトと `PersistentVolume`オブジェクトと同様に、 `PersistentVolumeClaim`Snapshotが作成されると、 `VolumeSnapshotContent`オブジェクトはSnapshotの作成を要求したオブジェクトへの1対1のマッピングを保持し `VolumeSnapshot`ます。

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のスナップショットを作成します。ボリュームグループのスナップショットが作成されると、Tridentは `VolumeGroupSnapshotContent`オブジェクトを作成し、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、管理者が定義した Kubernetes API 内のエンドポイントであり、類似するオブジェクトのグループ化に使用されます。Kubernetes では、オブジェクトのコレクションを格納するためのカスタムリソースの作成をサポートしています。これらのリソース定義を取得するには 'kubectl get CRDs' を実行します

カスタムリソース定義（ CRD ）と関連するオブジェクトメタデータは、 Kubernetes によってメタデータストアに格納されます。これにより、 Trident の独立したストアが不要になります。

Tridentは、オブジェクトを使用し `CustomResourceDefinition`て、Tridentバックエンド、Tridentストレージクラス、TridentボリュームなどのTridentオブジェクトのIDを保持します。これらのオブジェクトは Trident によって管理されます。また、 CSI のボリュームスナップショットフレームワークには、ボリュームスナップショットの定義に必要ないくつかの SSD が導入されています。

CRD は Kubernetes の構成要素です。上記で定義したリソースのオブジェクトは Trident によって作成されます。簡単な例として 'tridentctl' を使用してバックエンドを作成すると ' 対応する tridentBackendsCRD オブジェクトが Kubernetes によって消費されるように作成されます

Trident の CRD については、次の点に注意してください。

- Trident をインストールすると、一連の CRD が作成され、他のリソースタイプと同様に使用できるようになります。
- Tridentをアンインストールするには、を使用します `tridentctl uninstall` コマンドであるTridentポッドが削除されましたが、作成されたSSDはクリーンアップされません。を参照してください "Trident を

アンインストールします" Trident を完全に削除して再構成する方法を理解する。

Trident `StorageClass`オブジェクト

TridentではKubernetesに対応するストレージクラスが作成されます `StorageClass` を指定するオブジェクト `csi.trident.netapp.io` プロビジョニング担当者のフィールドに入力します。ストレージクラス名がKubernetesの名前と一致していること `StorageClass` 表すオブジェクト。

メモ	Kubernetes では、 Trident をプロビジョニングツールとして使用する Kubernetes 「 torageClass 」が登録されると、これらのオブジェクトが自動的に作成されます。

ストレージクラスは、ボリュームの一連の要件で構成されます。Trident は、これらの要件と各ストレージプール内の属性を照合し、一致する場合は、そのストレージプールが、そのストレージクラスを使用するボリュームのプロビジョニングの有効なターゲットになります。

REST API を使用して、ストレージクラスを直接定義するストレージクラス設定を作成できます。ただし、Kubernetes の導入では、新しい Kubernetes の「 torageClass 」オブジェクトを登録するときに、これらのオブジェクトが作成されることを期待しています。

Trident バックエンドオブジェクト

バックエンドとは、 Trident がボリュームをプロビジョニングする際にストレージプロバイダを表します。 1 つの Trident インスタンスであらゆる数のバックエンドを管理できます。

メモ	これは、自分で作成および管理する 2 つのオブジェクトタイプのうちの 1 つです。もう 1 つは、 Kubernetes の「 torageClass 」オブジェクトです。

これらのオブジェクトの作成方法の詳細については、を参照してください。 "バックエンドの設定"。

Trident `StoragePool`オブジェクト

ストレージ プールは、各バックエンドで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。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。

他のオブジェクトとは異なり、ストレージプールの候補は常に自動的に検出されて管理されます。

Trident `Volume`オブジェクト

ボリュームはプロビジョニングの基本単位であり、NFS共有、iSCSI LUN、FC LUNなどのバックエンドエンドエンドポイントで構成されます。Kubernetesでは、これらはに直接対応し `PersistentVolumes`ます。ボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

メモ	• Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。 • 関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP （ NAS 、 SAN ）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - * ：使用する Snapshot ポリシー
Snapshot リザーブ	文字列	いいえ	ONTAP - * ： Snapshot 用にリザーブされているボリュームの割合
エクスポートポリシー	文字列	いいえ	ONTAP-NAS* ：使用するエクスポートポリシー
snapshotDirectory の略	ブール値	いいえ	ONTAP-NAS* ： Snapshot ディレクトリが表示されているかどうか
unixPermissions	文字列	いいえ	ONTAP-NAS* ：最初の UNIX 権限
ブロックサイズ	文字列	いいえ	SolidFire - * ：ブロック / セクターサイズ
ファイルシステム	文字列	いいえ	ファイルシステムのタイプ

属性	を入力します	必須	説明
リカバリキューをスキップ	文字列	いいえ	ボリュームの削除中は、ストレージ内のリカバリキューをバイパスし、ボリュームを直ちに削除します。

Trident は ' ボリュームの作成時に internalName を生成しますこの構成は 2 つのステップで構成されます。最初に、ストレージプレフィックス（デフォルトの「 trident 」またはバックエンド構成のプレフィックス）をボリューム名の前に付加し、「 <prefix> - <volume-name> 」という形式の名前を付けます。その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAP バックエンドでは、ハイフンをアンダースコアで置き換えます（つまり、内部名は「 <prefix>_<volume-name>` 」になります）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム設定を使用して、 REST API を使用してボリュームを直接プロビジョニングできますが、 Kubernetes 環境では、ほとんどのユーザが標準の Kubernetes の「 PersistentVolumeClaim 」メソッドを使用することを想定しています。Trident は、プロビジョニングプロセスの一環として、このボリュームオブジェクトを自動的に作成します。

Trident `Snapshot`オブジェクト

Snapshot はボリュームのポイントインタイムコピーで、新しいボリュームのプロビジョニングやリストア状態に使用できます。Kubernetes では ' これらは 'VolumeSnapshotContent' オブジェクトに直接対応します各 Snapshot には、 Snapshot のデータのソースであるボリュームが関連付けられます。

個々の「スナップショット」オブジェクトには、以下のプロパティが含まれています。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン（「 1 」）
名前	文字列	はい。	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前

メモ　Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetes の「 VolumeSnapshot 」オブジェクト要求が作成されると、 Trident は元のストレージシステム上にスナップショットオブジェクトを作成することによって動作します。このスナップショットオブジェクトの「 internalName 」は、プレフィックス「 snapshot-` と「 VolumeSnapshot 」オブジェクトの「 UID 」を

組み合わせることによって生成されます（例：「 snapshot-e8d8d8a0ca-9826-11e9-9807-525400f3f660 」）。「 volumeName 」と「 volumeInternalName 」には、バッキングボリュームの詳細を取得して値を設定します。

Trident `ResourceQuota`オブジェクト

Tridentデーモンセットは、Kubernetesで利用可能な最高の優先度クラスであるプライオリティクラスを使用して `system-node-critical`、ノードの正常なシャットダウン時にTridentがボリュームを識別してクリーンアップできるようにし、リソースへの負荷が高いクラスタでは、Tridentデーモンセットポッドが優先度の低いワークロードをプリエンプトできるようにします。

これを実現するために、Tridentはオブジェクトを使用し `ResourceQuota`て、`Trident`デーモンセットの「`system-node-critical`」優先クラスを確実に満たします。デプロイメントおよびデーモンセットの作成の前に、`Trident`はオブジェクトを検索し `ResourceQuota`、検出されていない場合は適用します。

デフォルトのリソース割り当ておよび優先クラスをより詳細に制御する必要がある場合は'custom.yamlを生成するか'Helmチャートを使用してResourceQuotaオブジェクトを構成します

次に示すのは'ResourceQuota'オブジェクトがTridentのデマ作用を優先する例です

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values:
          - system-node-critical
```

リソースクォータの詳細については、を参照してください。 "Kubernetes：リソースクォータ"。

クリーンアップ `ResourceQuota` インストールが失敗した場合

まれに'ResourceQuotaオブジェクトが作成された後にインストールが失敗する場合は'最初に実行します "アンインストール中です" を再インストールします。

それでも解決しない場合は'ResourceQuotaオブジェクトを手動で削除します

取り外します `ResourceQuota`

独自のリソース割り当てを制御する場合は、次のコマンドを使用してTridentオブジェクトを削除できます `ResourceQuota`。

```
kubectl delete quota trident-csi -n trident
```

PODセキュリティ標準（PSS）およびセキュリティコンテキストの制約（SCC）

Kubernetesポッドのセキュリティ標準（PSS）とポッドのセキュリティポリシー（PSP）によって、権限レベルが定義され、ポッドの動作が制限されます。また、OpenShift Security Context Constraints（SCC）でも、OpenShift Kubernetes Engine固有のポッド制限を定義します。このカスタマイズを行うために、Tridentはインストール時に特定の権限を有効にします。次のセクションでは、Tridentによって設定される権限について詳しく説明します。

メモ	PSSは、Podセキュリティポリシー（PSP）に代わるものです。PSPはKubernetes v1.21で廃止され、v1.25で削除されます。詳細については、を参照してください。 "Kubernetes：セキュリティ"。

必須の**Kubernetes Security Context**と関連フィールド

アクセス権	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、を参照してください "Kubernetes：マウントの伝播"。
ホストネットワーク	iSCSIデーモンに必要です。「iscsiadm」はiSCSIマウントを管理し、ホストネットワークを使用してiSCSIデーモンと通信します。
ホストIPC	NFSは'IPC（プロセス間通信）を使用して'nfsdと通信します
ホストPID	NFSで開始する必要があり `rpc-statd`ます。Tridentは、NFSボリュームをマウントする前にが実行されているかどうかをホストプロセスに照会して判断し`rpc-statd`ます。
機能	SYS_Admin'機能は'特権コンテナのデフォルト機能の一部として提供されますたとえば、Dockerは特権コンテナにこれらの機能を設定します。「CapPrm：0000003fffffffffffff`Capff：0000003fffffffff」
Seccom	Seccompプロファイルは特権コンテナでは常に「制限されていない」ため、Tridentでは有効にできません。

アクセス権	説明
SELinux	OpenShiftでは、特権コンテナは（「Super Privileged Container」）ドメインで実行され、非特権コンテナはドメインで実行さ `spc_t`れ `container_t`ます。では `containerd`、がインストールされていると `container-selinux`、すべてのコンテナがドメイン内で実行され `spc_t`、SELinuxが実質的に無効になります。そのため、Tridentはコンテナに追加しません `seLinuxOptions`。
DAC	特権コンテナは、ルートとして実行する必要があります。CSIに必要なUNIXソケットにアクセスするために、非特権コンテナはrootとして実行されます。

PODセキュリティ標準（PSS）

ラベル	説明	デフォルト
'pod -security.esvus.io/enforce `po-security.Kubernetes io/enforce-version	Tridentコントローラとノードをインストールネームスペースに登録できるようにします。ネームスペースラベルは変更しないでください。	「enforce：特権」「enforce-version：」は、現在のクラスタのバージョンまたはテストされたPSSの最新バージョンです

警告

名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating：…」または「Warning：trident-csi-…」が表示される場合があります。この場合は'Privilegedの名前空間ラベルが変更されていないかどうかを確認してくださいその場合は、Tridentを再インストールします。

PoDセキュリティポリシー（PSP）

フィールド	説明	デフォルト
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowedCSIDrivers	TridentはインラインCSIエフェメラルボリュームを使用しません。	空です
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
「allowedFlexVolumes」を参照してください	Tridentはを利用しません "FlexVolドライバ"そのため、これらのボリュームは許可されるボリュームのリストに含まれていません。	空です
「allowedHostPaths」を参照してください	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空です

フィールド	説明	デフォルト
allowedProcMountTypes	Tridentでは'ProcMountTypes'は使用しません	空です
"allowedUnsafeSysctls"	Tridentには安全でないsysctls'は必要ありません。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
defaultAllowPrivilegeEscalation`	権限の昇格は、各Tridentポッドで処理されます。	「偽」
「forbiddenSysctls」と入力します	sysctlsは許可されません	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「hostIPC」	NFSボリュームをマウントするには'nfsdと通信するためにホストIPCが必要です	「真」
「ホストネットワーク」	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「hostPID」	ノードでRPC-statdが実行されているかどうかを確認するには'ホストPIDが必要です	「真」
「hostPorts」	Tridentはホストポートを使用しません。	空です
「特権」	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsGroup`	Tridentコンテナはrootとして実行されます。	「RunAsAny」
runAsUser	Tridentコンテナはrootとして実行されます。	runAsany`
runtimeClass'	Tridentは'RuntimeClasses'を使用しません	空です

フィールド	説明	デフォルト
SELinux	Tridentは'seLinuxOptions'を設定していませんこれは'コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」、「projected」、「emptyDir」

セキュリティコンテキストの制約（**SCC**）

ラベル	説明	デフォルト
allowHostDirVolumePlugin	Tridentノードのポッドは、ノードのルートファイルシステムをマウントします。	「真」
"allowHostIPC"	NFSボリュームをマウントするには'nfsdと通信するためにホストIPCが必要です	「真」
「allowHostNetwork」を参照してください	iscsiadmには、iSCSIデーモンと通信するためのホストネットワークが必要です。	「真」
「allowHostPID」を参照してください	ノードでRPC-statdが実行されているかどうかを確認するには'ホストPIDが必要です	「真」
"allowHostPorts`	Tridentはホストポートを使用しません。	「偽」
'allowPrivilegeEscalation'	特権コンテナは、特権昇格を許可する必要があります。	「真」
allowPrivilegeContainer]を参照してください	Tridentノードのポッドでは、ボリュームをマウントするために特権コンテナを実行する必要があります。	「真」
"allowedUnsafeSysctls"	Tridentには安全でないsysctls'は必要ありません。	「 NONE 」
「allowedCapabilities」を参照してください	権限のないTridentコンテナにはデフォルトよりも多くの機能が必要ないため、特権コンテナには可能なすべての機能が付与されます。	空です
defaultAddCapabilities	特権コンテナに追加する機能は必要ありません。	空です
「fsGroup」と入力します	Tridentコンテナはrootとして実行されます。	「RunAsAny」

ラベル	説明	デフォルト
「グループ」	このSCCはTridentに固有で、ユーザにバインドされています。	空です
「readOnlyRootFilesystem」	Tridentノードのポッドは、ノードのファイルシステムに書き込む必要があります。	「偽」
DDropCapabilitiesが必要です	Tridentノードのポッドは特権コンテナを実行するため、機能をドロップすることはできません。	「 NONE 」
runAsUser	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「seLinuxContext」	Tridentは'seLinuxOptions'を設定していませんこれは'コンテナランタイムとKubernetesディストリビューションがSELinuxを処理する方法には現在のところ違いがあるためです	空です
「seccompProfiles」	特権のあるコンテナは常に「閉鎖的」な状態で実行されます。	空です
「supplementalGroups」を参照してください	Tridentコンテナはrootとして実行されます。	「RunAsAny」
「ユーザー」	このSCCをTridentネームスペースのTridentユーザにバインドするエントリが1つあります。	該当なし
「ボリューム」	Tridentポッドには、このボリュームプラグインが必要です。	「hostPath」, downwardAPI, projected , emptyDir

法的通知

著作権に関する声明、商標、特許などにアクセスできます。

著作権

"https://www.netapp.com/company/legal/copyright/"

商標

NetApp 、 NetApp のロゴ、および NetApp の商標ページに記載されているマークは、 NetApp, Inc. の商標です。その他の会社名および製品名は、それぞれの所有者の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

ネットアップが所有する特許の最新リストは、次のサイトで入手できます。

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

プライバシーポリシー

"https://www.netapp.com/company/legal/privacy-policy/"

オープンソース

NetApp for Tridentで使用されているサードパーティの著作権およびライセンスは、の各リリースのNOTICESファイルで確認できます https://github.com/NetApp/trident/。

著作権に関する情報

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用権を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用権については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、http://www.netapp.com/TMに記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。