Astra Trident 23.04ドキュメント

Astra Trident

NetApp
March 05, 2026

This PDF was generated from https://docs.netapp.com/ja-jp/trident-2304/index.html on March 05, 2026.
Always check docs.netapp.com for the latest.

目次

Astra Trident 23.04ドキュメント 1
リリースノート 2
新機能 2
23.04の新機能 2
23.01.1の変更点 3
23.01の変更点 3
22.10の変更 4
2007年22月の変更 5
2004年10月22日の変更 6
22.01.1 の変更 7
22.01.0での変更点 7
21.10.1 の変更点 8
21.10.0ノヘンコウテン 8
既知の問題 9
詳細については、こちらをご覧ください 10
以前のバージョンのドキュメント 10
概念 11
Astra Trident の詳細をご確認ください 11
概要 11
サポートされる Kubernetes クラスタアーキテクチャ 11
アストラとは 11
を参照してください。 12
ONTAP ドライバ 12
Astra Controlでサポートされるドライバ 12
ONTAP向けAstra Tridentストレージドライバ 12
プロビジョニング 13
ストレージクラスの関連付け 13
ボリュームの作成 14
ボリューム Snapshot 14
ボリュームSnapshotの作成方法について説明します 14
仮想プール 15
仮想プールについて説明します 15
ボリュームアクセスグループ 16
ボリュームアクセスグループについて学習する 16
はじめに 18
ぜひお試しください 18
試乗について学びます 18
要件 18
Astra Trident 23.01に関する重要な情報 18

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） 18
サポートされるバックエンド（ストレージ） 19
機能の要件 19
テスト済みのホストオペレーティングシステム 20
ホストの設定 20
ストレージシステムの構成： 20
Astra Trident ポート 21
コンテナイメージと対応する Kubernetes バージョン 21
Astra Trident をインストール 23
Astra Tridentのインストール方法をご確認ください 23
Tridentオペレータを使用してインストール 28
tridentctlを使用してインストールします 54
次の手順 58
手順 1 ：バックエンドを作成する 59
手順 2 ：ストレージクラスを作成する 59
手順 3 ：最初のボリュームをプロビジョニングします 61
手順 4 ：ボリュームをポッドにマウントする 62
Trident で Astra を管理 64
Astra Trident をアップグレード 64
Astra Trident をアップグレード 64
オペレータにアップグレードしてください 66
tridentctl を使用してアップグレードします 74
Astra Trident をアンインストール 78
Helm を使用してアンインストールします 78
Trident オペレータを使用してをアンインストールします 79
を使用してをアンインストールします `tridentctl` 80
Trident をダウングレード 80
ダウングレードするタイミング 80
ダウングレードしない場合 80
Operator を使用して Astra Trident をインストールする場合のダウングレードプロセス 80
を使用してAstra Tridentをインストールした場合のダウングレードプロセス `tridentctl` 82
Astra Trident を使用 84
ワーカーノードを準備します 84
適切なツールを選択する 84
ノードサービスの検出 84
NFS ボリューム 85
iSCSI ボリューム 85
バックエンドを設定 88
バックエンドを設定 88
Azure NetApp Files の特長 89
Google Cloudバックエンド用にCloud Volumes Service を設定します 101

NetApp HCI または SolidFire バックエンドを設定します 118
ONTAP SANドライバ 125
ONTAP NAS ドライバ 147
NetApp ONTAP 対応の Amazon FSX 177
kubectl を使用してバックエンドを作成します 189
TridentBackendConfig 190
手順の概要 191
手順 1 ： Kubernetes Secret を作成します 191
手順2：を作成します TridentBackendConfig CR 193
手順3：のステータスを確認します TridentBackendConfig CR 194
（オプション）手順 4 ：詳細を確認します 194
kubectl を使用してバックエンド管理を実行します 196
バックエンドを削除します 196
既存のバックエンドを表示します 196
バックエンドを更新します 196
tridentctl を使用してバックエンド管理を実行します 197
バックエンドを作成します 197
バックエンドを削除します 197
既存のバックエンドを表示します 198
バックエンドを更新します 198
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 198
バックエンド管理オプション間を移動します 199
バックエンドを管理するためのオプション 199
管理 tridentctl を使用してバックエンドを TridentBackendConfig 199
管理 TridentBackendConfig を使用してバックエンドを tridentctl 203
ストレージクラスを管理する 205
ストレージクラスを設計する 205
ストレージクラスを作成する。 205
ストレージクラスを削除する 206
既存のストレージクラスを表示します 206
デフォルトのストレージクラスを設定する 206
ストレージクラスのバックエンドを特定します 207
ボリューム操作を実行する 207
CSI トポロジを使用します 207
スナップショットを操作します 215
ボリュームを展開します 220
ボリュームをインポート 227
ネームスペース間でNFSボリュームを共有します 237
機能 237
クイックスタート 237
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 238

共有ボリュームを削除 240
使用 tridentctl get 下位のボリュームを照会する 240
制限 240
を参照してください。 240
Astra Trident を監視 240
概要 241
手順 1 ： Prometheus ターゲットを定義する 241
手順 2 ： Prometheus ServiceMonitor を作成します 241
ステップ 3 ： PrompQL を使用して Trident 指標を照会する 242
Astra Trident AutoSupport の計測データ 243
Astra Trident の指標を無効化 244
Trident for Docker が必要です 245
導入の前提条件 245
要件を確認します 245
Astra Trident を導入 248
Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降） 248
従来の方法（バージョン 1.12 以前） 250
システム起動時に Astra Trident を起動 251
Astra Trident をアップグレードまたはアンインストールする 252
アップグレード 253
をアンインストールします 254
ボリュームを操作します 254
ボリュームを作成します 254
ボリュームを削除します 255
ボリュームのクローンを作成します 255
外部で作成されたボリュームにアクセス 257
ドライバ固有のボリュームオプション 257
ログを収集します 262
トラブルシューティング用にログを収集する 262
一般的なトラブルシューティングのヒント 263
複数の Astra Trident インスタンスを管理 263
Docker Managed Plugin （バージョン 1.13 / 17.03 以降）の手順 263
従来の（バージョン 1.12 以前）の場合の手順 264
ストレージ構成オプション 264
グローバル構成オプション 264
ONTAP構成 265
Element ソフトウェアの設定 271
既知の問題および制限事項 273
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。 273

ボリューム名は 2 文字以上にする必要があります。 274
Docker Swarm には、 Astra Trident がストレージやドライバのあらゆる組み合わせでサポートしないようにする一定の動作があります。 274
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。 274
よくある質問 275
一般的な質問 275
Trident がリリースされる頻度を教えてください。 275
Astra 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。 275
Astra Trident には、他のネットアップ製品との依存関係はありますか。 275
Astra Trident の設定の詳細をすべて取得するにはどうすればよいですか。 275
Astra Trident を使用してストレージをプロビジョニングする方法に関するメトリクスを取得できますか。 275
Astra Trident を CSI プロビジョニング担当者として使用すると、ユーザエクスペリエンスは変化しますか。 275
Kubernetes クラスタに Astra Trident をインストールして使用 275
のサポート対象のバージョンを指定します `etcd` ？ 275
Astra 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。 276
Astra Trident はリモートからインストールできますか。 276
Astra Trident でハイアベイラビリティを構成できますか。 276
Astra 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。 276
Astra Trident で使用されるロールと権限を教えてください。 276
Astra 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。 276
2 つの別々の Kubernetes クラスタに対して、同じ ONTAP バックエンド SVM を 2 つの別々の Astra Trident インスタンスに対して共有できますか。 276
ContainerLinux （旧 CoreOS ）に Astra Trident をインストールすることはできますか。 276
ネットアップの Cloud Volumes ONTAP で Astra Trident を使用できますか。 277
Astra Trident は Cloud Volume サービスと連携していますか。 277
トラブルシューティングとサポート 277
ネットアップは Astra Trident をサポートしていますか。 277
サポートケースを作成するにはどうすればよいですか？ 277
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？ 277
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。 277
不具合を発生させる場所 277
ネットアップが Trident の Astra について簡単に質問できたらどうなりますか。コミュニティやフォーラムはありますか？ 277
ストレージシステムのパスワードが変更され、 Astra Trident が機能しなくなった場合、どのように回復すればよいですか。 277
Astra Trident が Kubernetes ノードを検出できない。この問題を解決するにはどうすればよいですか 278
Trident ポッドが破損すると、データは失われますか？ 278
Astra Trident をアップグレード 278

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）？ 278
Trident を以前のリリースにダウングレードできますか。 278
バックエンドとボリュームを管理 278
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。 278
Astra Trident が ONTAP バックエンドに CHAP を設定できるか。 279
Astra 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。 279
データ LIF にポートを指定できるか。 279
管理 LIF とデータ LIF に IPv6 アドレスを使用できますか。 279
バックエンドの管理 LIF を更新できますか。 279
バックエンドのデータ LIF を更新できるか。 279
Kubernetes 向け Astra Trident で複数のバックエンドを作成できますか。 279
Astra 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。 279
Astra Trident ではどのようにして特定のバックエンドを選択しますか。 280
Astra 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。 280
同じ種類のバックエンドが複数ある場合、 Astra Trident はどのバックエンドを使用するかをどのように選択しますか。 280
Astra Trident は、 Element / SolidFire で双方向 CHAP をサポートしていますか。 280
Trident が ONTAP ボリュームに qtree を導入する方法を教えてください。1 つのボリュームに配置できる qtree の数はいくつですか。 280
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。 280
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。 280
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？ 281
ONTAP を使用して Astra Trident 経由でボリューム暗号化を設定する方法を教えてください。 281
Trident 経由で ONTAP に QoS を実装するには、どのような方法が最適ですか。 281
Trident 経由でシンプロビジョニングや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。 281
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。 281
Astra Trident によって作成された NFS PVC を拡張できますか。 281
Astra Trident の外部で作成したボリュームを Astra Trident にインポートできますか。 281
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。 281
Astra Trident によって作成された iSCSI PVC を拡張できますか。 282
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 282
Trident を使用してボリューム Snapshot を作成できますか。 282
Astra Trident の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。 282
ONTAP を使用して Astra Trident でプロビジョニングしたボリュームの Snapshot バックアップを作成する方法を教えてください。 282

Trident 経由でプロビジョニングしたボリュームの Snapshot リザーブの割合を設定できますか。 282
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。 283
Astra Trident を使用して、ボリューム用の SnapMirror をセットアップできますか。 283
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？ 283
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。 283
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。 283
サポート 284
トラブルシューティング 285
全般的なトラブルシューティング 285
オペレータを使用して失敗した Trident の導入をトラブルシューティングします 287
を使用したTridentの導入に失敗した場合のトラブルシューティング `tridentctl` 289
ベストプラクティスと推奨事項 290
導入 290
専用のネームスペースに導入します 290
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 290
ストレージ構成 290
プラットフォームの概要 290
ONTAP と Cloud Volumes ONTAP のベストプラクティス 290
SolidFire のベストプラクティス 295
詳細情報の入手方法 297
Astra Trident を統合 297
ドライバの選択と展開 297
ストレージクラスの設計 301
仮想プールの設計 302
ボリューム操作 303
OpenShift サービスを導入します 305
指標サービス 307
データ保護とディザスタリカバリ 308
Astra Tridentのレプリケーションとリカバリ 308
SVMレプリケーションとリカバリ 308
ボリュームのレプリケーションとリカバリ 310
Snapshotによるデータ保護 310
Astra Control Centerアプリケーションのレプリケーション 310
セキュリティ 310
セキュリティ 310
Linux Unified Key Setup（LUKS；統合キーセットアップ） 312
参照 318
Astra Trident ポート 318
Astra Trident ポート 318
Astra Trident REST API 318
REST APIを使用する状況 318

REST APIを使用する 318
コマンドラインオプション 319
ロギング 319
Kubernetes 319
Docker です 320
REST 320
ネットアップの製品が Kubernetes と統合されます 320
アストラ 320
ONTAP 320
Cloud Volumes ONTAP 321
NetApp ONTAP 対応の Amazon FSX 321
Element ソフトウェア 321
NetApp HCI 321
Azure NetApp Files の特長 321
Cloud Volumes Service for Google Cloud 321
Kubernetes オブジェクトと Trident オブジェクト 321
オブジェクトは相互にどのように相互作用しますか。 322
Kubernetes `PersistentVolumeClaim` オブジェクト 322
Kubernetes `PersistentVolume` オブジェクト 324
Kubernetes `StorageClass` オブジェクト 324
Kubernetes `VolumeSnapshotClass` オブジェクト 329
Kubernetes `VolumeSnapshot` オブジェクト 329
Kubernetes `VolumeSnapshotContent` オブジェクト 329
Kubernetes `CustomResourceDefinition` オブジェクト 330
Trident `StorageClass` オブジェクト 330
Trident バックエンドオブジェクト 331
Trident `StoragePool` オブジェクト 331
Trident `Volume` オブジェクト 331
Trident `Snapshot` オブジェクト 333
Astra Trident `ResourceQuota` オブジェクト 333
tridentctl コマンドとオプション 334
使用可能なコマンドとオプション 335
`create` 336
`delete` 336
`get` 336
`images` 337
`import volume` 337
`install` 337
`logs` 338
`send` 339
`uninstall` 339

update . . . 339
upgrade . . . 339
version . . . 340
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） . . . 340
必須のKubernetes Security Contextと関連フィールド . . . 340
PODセキュリティ標準（PSS） . . . 341
PoDセキュリティポリシー（PSP） . . . 341
セキュリティコンテキストの制約（SCC） . . . 343
法的通知 . . . 345
著作権 . . . 345
商標 . . . 345
特許 . . . 345
プライバシーポリシー . . . 345
オープンソース . . . 345

Astra Trident 23.04ドキュメント

リリースノート

新機能

リリースノートでは、最新バージョンの Astra Trident の新機能、拡張機能、およびバグ修正に関する情報を提供しています。

。 `tridentctl` インストーラzipファイルに含まれているLinux用のバイナリは、テスト済みでサポートされているバージョンです。に注意してください `macos` バイナリはで提供されます `/extras` zipファイルの一部はテストされていないか、サポートされていません。

23.04の新機能

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

の修正

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

拡張機能

- Kubernetes：
  - Kubernetes 1.27のサポートを追加。
  - LUKSボリュームのインポートのサポートが追加されました。
  - ReadWriteOncePod PVCアクセスモードのサポートが追加されました。
  - ノードの正常でないシャットダウン時にONTAP-SAN-*ボリュームで強制的に接続解除がサポートされるようになりました。

- すべてのontap-san-*ボリュームでノード単位のigroupを使用するようになりました。LUNはigroupにマッピングされるだけで、それらのノードにアクティブにパブリッシュされるため、セキュリティ体制が強化されます。アクティブなワークロードに影響を与えることなく既存のボリュームを安全であるとTridentが判断した場合、必要に応じて新しいigroupスキームに切り替えます（"問題#758"）。
- Tridentで管理されていないigroupをONTAP-SAN-*バックエンドからクリーンアップし、Tridentのセキュリティを強化

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

非推奨

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

23.01.1の変更点

の修正

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

23.01の変更点

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

の修正

- Kubernetes：Helm（"問題#783、#794"）。

拡張機能

Kubernetes

- Kubernetes 1.26のサポートを追加。

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

Astra Trident

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

非推奨

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

22.10の変更

- Astra 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

Astra Tridentに関する重要な情報22.10

- 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、Astra Tridentを22.10にアップグレードする必要があります。
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用するよう強制し、推奨値をに設定するようになりました `find_multipaths: no` multipath.confファイル内。

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

の修正

- を使用して作成されたONTAP バックエンドに固有の修正済み問題 `credentials` 22.07.0アップグレード時にフィールドがオンラインにならない（"問題 #759"）。
- **Docker**：一部の環境でDockerボリュームプラグインが起動しないという問題 が修正されました（"問題#548" および "問題#760"）。

- レポートノードに属するデータLIFのサブセットのみが公開されるように、ONTAP SANバックエンド固有の修正されたSLM問題 。
- ボリュームの接続時にiSCSI LUNの不要なスキャンが発生するというパフォーマンス問題 の問題が修正されました。
- Astra Trident iSCSIワークフロー内で詳細な再試行を削除し、失敗の時間を短縮。外部の再試行間隔も短縮
- 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のサポートが追加されました。Kubernetes 1.25にアップグレードする前に、Astra 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。既存のボリュームは、アクティブなワークロードに影響を与えずに安全であるとAstra Tridentが判断したときに、必要に応じて新しいigroupスキームに切り替えられます。
- 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。
- ANFドライバへの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をシンプルなホスト情報とアクティブなサービス検出に絞り込み、ベストエフォートを実現ワーカーノードでNFS / iSCSIサービスを使用できることを確認します。

ドキュメント

新しい "PODセキュリティ標準"（PSS）セクションに、インストール時にAstra Tridentによって有効化され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。Astra Trident の最新機能をいくつかご紹介します。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TridentリリースからアップグレードしてAzure NetApp Files を使用している場合は、を参照してください `location` configパラメータは'必須のシングルトンフィールドになりました

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"）
- を設定するときに、自動サポート画像をプルする固定問題 `imageRegistry` インストール中にフラグを付けます。（"GitHub問題#715"）
- ANF ドライバが複数のエクスポートルールでボリュームのクローニングに失敗した修正済み問題 。

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での変更点

の修正

- * Kubernetes ：大規模なクラスタのノード登録バックオフ再試行時間を延長します。
- azure-NetApp-files ドライバが、同じ名前の複数のリソースによって混乱することがあるという解決済みの問題 。
- ONTAP SAN IPv6 データ LIF が角かっこで指定した場合に機能するようになりました。
- すでにインポートされているボリュームをインポートしようとすると、 EOF 問題 が返され、 PVC は保留状態になります。（"GitHub問題#489"）
- Fixed 問題 ： Astra Trident では、 SolidFire ボリュームで作成される Snapshot が 32 個を超えるとパフォーマンスが低下します。
- SSL 証明書の作成時に SHA-1 を SHA-256 に置き換えました。
- リソース名が重複して 1 箇所に操作が制限されるように ANF ドライバを修正しました。
- リソース名が重複して 1 箇所に操作が制限されるように ANF ドライバを修正しました。

拡張機能

- Kubernetes の機能拡張：
  - Kubernetes 1.23のサポートを追加。
  - 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"）
- ANF ボリュームでの「 unixPermissions 」オプションのサポートが追加されました。（"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ノヘンコウテン

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 ）
- Astra Trident がシャットダウン時に致命的なエラーを記録した修正版問題（ GitHub 問題 597 ）。
- Kubernetes 関連の修正：
  - を使用してSnapshotを作成する場合、リストアサイズの最小値としてボリュームの使用済みスペースを返します `ontap-nas` および `ontap-nas-flexgroup` ドライバ（GitHub問題 645）。
  - 問題 を修正 `Failed to expand filesystem` ボリュームのサイズ変更後にエラーがログに記録されました（GitHub問題 560）。
  - ポッドが固定される問題 を修正 `Terminating` 状態（GitHub問題 572）。
  - のケースを修正しました `ontap-san-economy` FlexVol はスナップショットLUNの一部である場合があります(GitHub問題 533)。

- 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 ）。
- 問題は修正され、 Astra Trident のすべてのインストーラが OpenShift としてプレーン Kubernetes を識別できるようになりました（ GitHub 問題 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

- のサポートが追加されました `unixPermissions` GCP - CVSパフォーマンスボリュームのオプション。
- 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
  - に演算子の画像を追加しました `tridentctl` [画像]コマンド（GitHub問題 570）。

実験的な機能強化

- で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 `ontap-san` ドライバ。
- のテクニカルプレビュー* RESTサポートを追加 `ontap-nas-flexgroup`、 `ontap-san`および `ontap-nas-economy` ドライバ。

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。

- Astra 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、value.yamlを更新して設定する必要があります `excludePodSecurityPolicy` 終了： `true` または、を追加します `--set excludePodSecurityPolicy=true` に移動します `helm upgrade` コマンドを実行してからクラスタをアップグレードしてください。
- Astra Tridentでは空白が強制されるようになりました `fsType` (`fsType=""`）を含むボリューム `fsType` ストレージクラスで指定されています。Tridentでは、Kubernetes 1.17以降を使用している場合は空白の入力がサポートされます `fsType` NFSボリューム。iSCSIボリュームの場合、を設定する必要があります `fsType` ストレージクラスで、を適用する場合 `fsGroup` セキュリティコンテキストの使用。
- 複数のAstra 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に異なる値を設定する必要があります `storagePrefix` ONTAP バックエンドの値を指定するか、別のを使用します `TenantName` SolidFire バックエンドの場合：Astra Trident は、 Astra Trident の他のインスタンスが作成したボリュームを検出できません。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。 Astra Trident は、ボリューム作成をべき等の操作として扱います。状況 `storagePrefix` または `TenantName` 同じバックエンドに作成されたボリュームでは名前が競合する可能性があるため、同じ名前を変更しないでください。
- Astra Tridentのインストール時（を使用 `tridentctl` またはTrident Operator）を使用し、を使用します `tridentctl` Astra Tridentを管理するには、が次の条件を満たしている必要があります `KUBECONFIG` 環境変数が設定されています。これは、Kubernetesクラスタにそれを示すために必要です `tridentctl` 対

策を検討してください。複数のKubernetes環境を使用する場合は、を確認してください `KUBECONFIG` ファイルは正確に取得されます。

- 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、が必要なRHEL / RedHat CoreOSインスタンスに該当します `discard` "マウントオプション"; discard mountOption がに含まれていることを確認します [StorageClass^]をクリックして、オンラインブロックの破棄をサポートします。
- Kubernetes クラスタごとに複数の Astra Trident インスタンスがある場合、 Astra Trident は他のインスタンスと通信できず、作成した他のボリュームを検出できません。そのため、 1 つのクラスタ内で複数のインスタンスを実行している場合、予期しない動作が発生したり、誤ったりすることがあります。Kubernetes クラスタごとに Trident のインスタンスが 1 つだけ必要です。
- If Astra Tridentベース `StorageClass` Tridentがオフラインのときにkubernetesからオブジェクトが削除されると、対応するストレージクラスがオンラインに戻ってもTridentから削除されることはありません。これらのストレージクラスは、を使用して削除してください `tridentctl` またはREST API。
- 対応する PVC を削除する前に Astra Trident によってプロビジョニングされた PV を削除しても、 Astra Trident は自動的に元のボリュームを削除しません。ボリュームは、から削除する必要があります `tridentctl` またはREST API。
- 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経由でAstra Tridentを使用する場合は、と指定する必要があります `managementLIF` および `dataLIF` バックエンドの定義を角かっこで囲みます。例： `[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。

> 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：Astra 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

- "Astra Trident GitHub"
- "Astra Trident のブログ"

以前のバージョンのドキュメント

Astra Trident 23.04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をベースに提供しています "Astra Tridentのリリースとサポートのライフサイクル"。

- "Astra Trident 23.01"
- "Trident 10月22日"
- "トライデント22.07年アストラト"
- "トライデント22.04アストラ"
- "Trident 22.01"

概念

Astra Trident の詳細をご確認ください

Astra Trident は、ネットアップがの一部として管理している、完全にサポートされているオープンソースプロジェクトです "Astra 製品ファミリー"。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に対する要求を満たすことができるように設計されています。

概要

Kubernetes クラスタにポッドとして Trident を導入し、 Kubernetes ワークロードに動的なストレージオーケストレーションサービスを提供コンテナ化されたアプリケーションは、ONTAP （AFF / FAS / Select / Cloud / Amazon FSX for NetApp ONTAP ）、Elementソフトウェア（NetApp HCI / SolidFire）、Azure NetApp Files サービス、Google Cloud上のCloud Volumes Service など、ネットアップの幅広いポートフォリオの永続的ストレージをすばやく簡単に消費できます。

Astra Trident は、 NetApp の Astra の基盤テクノロジでもあり、 NetApp のスナップショット、バックアップ、レプリケーション、クローニングに業界をリードするデータ管理テクノロジを活用して、 Kubernetes ワークロードのデータ保護、ディザスタリカバリ、ポータビリティ、移行のユースケースに対応します。

サポートされる **Kubernetes** クラスタアーキテクチャ

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

Kubernetes クラスタアーキテクチャ	サポートされます	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。

アストラとは

Astra を使用すると、 Kubernetes で実行されている大量のデータコンテナ化ワークロードを、パブリッククラウドとオンプレミスの間で簡単に管理、保護、移動できます。Astra は、ネットアップの実績ある拡張可能なパブリッククラウドストレージポートフォリオとオンプレミスのストレージポートフォリオから、 Astra Trident を使用して永続的なコンテナストレージをプロビジョニングし、提供します。また、 Kubernetes ワークロード向けに、 Snapshot 、バックアップとリストア、アクティビティログ、アクティブクローニングによるデータ保護、ディザスタ / データリカバリ、データ監査、移行のユースケースなど、アプリケーションに対応した高度なデータ管理機能も豊富に用意されています。

Astra のページで無料トライアルに登録できます。

を参照してください。

- "ネットアップアストラ製品ファミリー"
- "Astra Control Service のマニュアル"
- "Astra Control Center のドキュメント"
- "Astra API ドキュメント"

ONTAP ドライバ

Astra Trident は、 ONTAP クラスタとの通信に使用する 5 つの ONTAP ストレージドライバを提供します。

Astra Controlでサポートされるドライバ

Astra Controlは、で作成したボリュームに対して、シームレスな保護、ディザスタリカバリ、および移動（Kubernetesクラスタ間でボリュームを移動）を提供します `ontap-nas`、 `ontap-nas-flexgroup`および `ontap-san` ドライバ。を参照してください "Astra Controlレプリケーションの前提条件" を参照してください。

- を使用する必要があります `ontap-nas` データ保護、ディザスタリカバリ、モビリティを必要とする本番環境のワークロード向けのサービスです。
- 使用 `ontap-san-economy` 想定されるボリューム使用量がONTAP でサポートされる量よりも大幅に多い場合
- 使用 `ontap-nas-economy` 想定されるボリューム使用量が、ONTAP でサポートされるおよびよりも大幅に多い場合にのみ該当します `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向けAstra Tridentストレージドライバ

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

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-economy`	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、NFS、SMB

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-flexgroup	NFS SMB	ファイルシステム	RWO、ROX、RWX、RWOP	""、NFS、SMB
ontap-san	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
ontap-san	iSCSI	ファイルシステム	RWO、ROX、RWOP rwxはファイルシステムボリュームモードでは使用できません。	xfs、 ext3、 ext4
ontap-san-economy	iSCSI	ブロック	RWO、ROX、RWX、RWOP	ファイルシステムなし。rawブロックデバイスです
ontap-san-economy	iSCSI	ファイルシステム	RWO、ROX、RWOP 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バックエンドの認証には、セキュリティロール（ユーザ名とパスワード）のログインクレデンシャル、またはONTAPクラスタにインストールされている秘密鍵と証明書を使用します。を使用して既存のバックエンドを更新し、認証モードを移行することができます `tridentctl update backend`。

プロビジョニング

Trident の Astra プロビジョニングの主なフェーズは 2 つあります。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ます。このフェーズでは、保留中のボリュームのストレージクラスに関連付けられたストレージプールからストレージプールを選択する必要があります。

ストレージクラスの関連付け

バックエンドストレージプールをストレージクラスに関連付けるには、ストレージクラスの要求された属性とその両方が必要です `storagePools`、 `additionalStoragePools`および `excludeStoragePools` リスト。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。要求された属性とプール名がストレージプールの属性と名前ですべて一致した場合、 Astra Trident がそのストレージプールを、そのストレージクラスに適した一連のストレージプールに追加します。さらに、TridentのAstraでは、にリストされているすべてのストレージプールが追加されます `additionalStoragePools` 属性がストレージクラスの要求した属性の一部または全部

を満たしていない場合も、そのセットにリストされます。を使用する必要があります `excludeStoragePools` ストレージクラスに対して使用するストレージプールを上書きおよび削除するリスト。Astra 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、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確認され、除外済みとマークされているストレージが削除されます。

ボリュームの作成

Trident がさらに、ストレージクラスとストレージプールの間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、最初にそのボリュームのストレージクラス用の一連のストレージプールが Trident から取得されます。 また、ボリュームにプロトコルを指定した場合、 Astra Trident は要求されたプロトコルを提供できないストレージプールを削除します（たとえば、 NetApp HCI / SolidFire バックエンドはファイルベースのボリュームを提供できませんが、 ONTAP NAS バックエンドはブロックベースのボリュームを提供できません）。Trident がこのセットの順序をランダム化し、ボリュームを均等に分散してから、各ストレージプールでボリュームを順番にプロビジョニングしようとします。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。Astra 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 * プロビジョニングに失敗した場合にのみ、障害 * を返します。

ボリューム Snapshot

Trident がドライバ用のボリュームスナップショットの作成をどのように処理するかについては、こちらをご覧ください。

ボリューム**Snapshot**の作成方法について説明します

- をクリックします `ontap-nas`、 `ontap-san`、 `gcp-cvs`および `azure-netapp-files` ドライバ、各永続ボリューム（PV）はFlexVol にマッピングされます。その結果、ボリューム Snapshot はネットアップ Snapshot として作成されます。NetApp のスナップショット・テクノロジーは ' 競合するスナップショット・テクノロジーよりも高い安定性 ' 拡張性 ' リカバリ性 ' パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- をクリックします `ontap-nas-flexgroup` ドライバ、各永続ボリューム（PV）はFlexGroup にマッピングされます。その結果、ボリューム Snapshot は NetApp FlexGroup Snapshot として作成されます。NetApp のスナップショット・テクノロジーは ' 競合するスナップショット・テクノロジーよりも高い安定性 ' 拡張性 ' リカバリ性 ' パフォーマンスを提供しますSnapshot コピーは、作成時とストレージスペースの両方で非常に効率的です。
- をクリックします `ontap-san-economy` ドライバとPVSは、共有FlexVol上に作成されたLUNにマッピングされます。PVS のボリューム Snapshot は、関連付けられた LUN の FlexClone を実行することで実現されます。ONTAP の FlexClone テクノロジにより、最大規模のデータセットでもほぼ瞬時にコピーを作成できます。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。
- をクリックします `solidfire-san` ドライバ。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。
- を使用して作業している場合 `ontap-nas` および `ontap-san` ドライバ、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、関連付けられている ONTAP Snapshot が Astra Trident から削除されます。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。

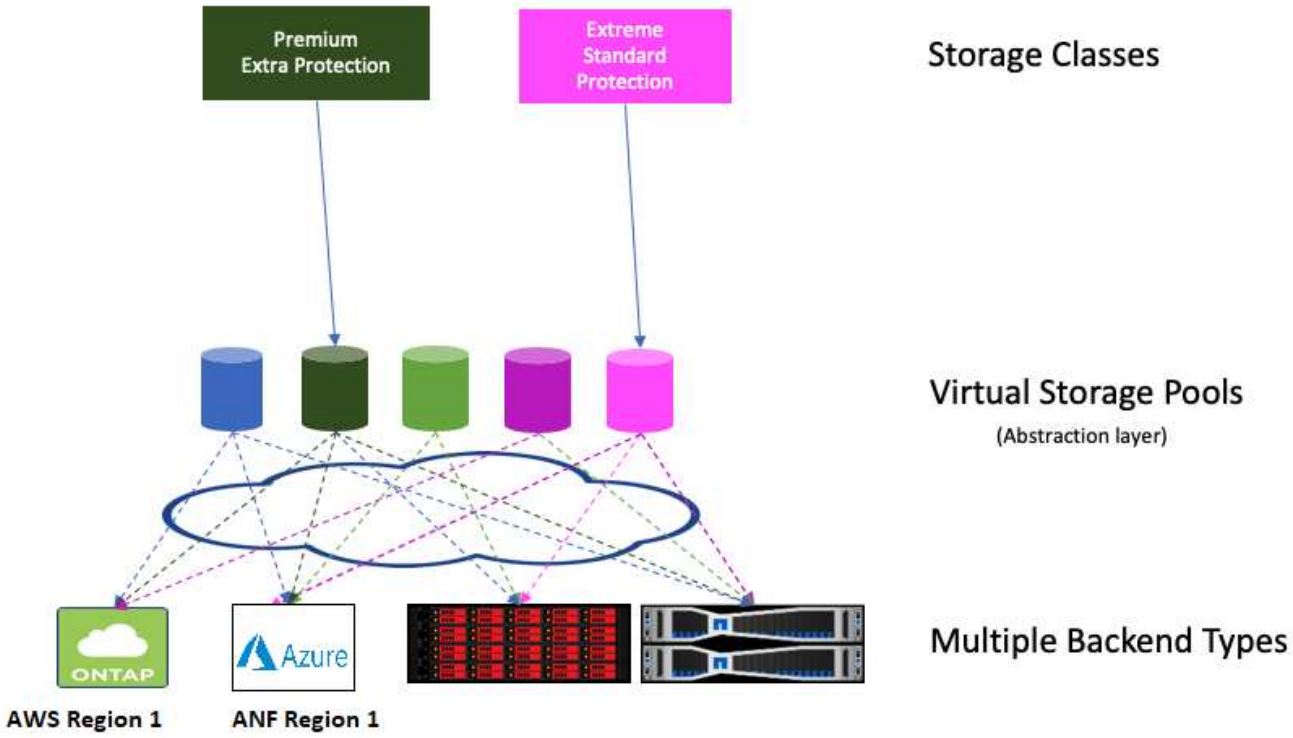
ネットアップの Trident では、ボリューム Snapshot を使用して PVS を新規作成できます。これらの Snapshot から PVS を作成するには、サポート対象の ONTAP および CVS バックエンドに対して FlexClone テクノロジを使用します。Snapshot から PV を作成する場合、バッキングボリュームは Snapshot の親ボリュームの FlexClone です。。 `solidfire-san` ドライバは、Elementソフトウェアのボリュームクローンを使用してSnapshotからPVSを作成します。ここで、 Element Snapshot から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

仮想プール

仮想プールは、Astra Tridentストレージバックエンドと Kubernetesの間に抽象化レイヤを提供します `StorageClasses`。管理者は、を作成することなく、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ます `StorageClass` 目的の条件を満たすために使用する物理バックエンド、バックエンドプール、またはバックエンドタイプを指定します。

仮想プールについて説明します

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



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

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

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

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

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

ボリュームアクセスグループ

Trident がどのように活用されているかをご確認ください "ボリュームアクセスグループ"。

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、 CSI モードで Astra Trident を使用している場合は、このセクションを無視できます。Astra Trident は、強化された CSI プロビジョニングツールとしてインストールされた場合、 CHAP を使用します。

ボリュームアクセスグループについて学習する

Astra Trident は、ボリュームアクセスグループを使用して、プロビジョニングするボリュームへのアクセスを制御できるCHAPが無効になっている場合は、というアクセスグループが検索されます `trident` 構成に1つ以上のアクセスグループIDを指定していない場合。

Trident が設定されたアクセスグループに新しいボリュームを関連付けても、アクセスグループ自体は作成も管理もされません。ストレージバックエンドが Astra Trident に追加される前に、アクセスグループが存在している必要があります。また、 Kubernetes クラスタ内の、バックエンドでプロビジョニングされたボリュームをマウントできるすべてのノードの 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"。

デフォルトを使用している構成から構成を変更する場合 `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。

はじめに

ぜひお試しください

ネットアップでは、リクエストに応じてすぐに使用できるラボイメージを提供しています "ネットアップのテスト用ドライブ"。

試乗について学びます

テストドライブは、 3 ノードの Kubernetes クラスタと Astra Trident がインストールおよび設定されたサンドボックス環境を提供します。Astra Trident をよく理解し、機能を調べるのに最適な方法です。

もう 1 つのオプションは、を参照することです "kubeadm インストールガイド" Kubernetes が提供します。

> 本番環境では、この手順で構築した Kubernetes クラスタを使用しないでください。本番環境向けのクラスタを作成するには、ディストリビューションに付属の本番環境導入ガイドを使用します。

Kubernetes を初めて使用する場合は、概念とツールについて理解しておいてください "こちらをご覧ください"。

要件

Astra Tridentをインストールする前に、次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

Astra Trident 23.01に関する重要な情報

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

  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 Astra 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

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.12
- Kubernetes 1.21~1.27
- Mirantis Kubernetes Engine 3.5
- OpenShift 4.9 ~ 4.12

Trident オペレータは、次のリリースでサポートされています。

- Anthosオンプレミス（VMware）とAnthos（ベアメタル1.12
- Kubernetes 1.21~1.27
- OpenShift 4.9 ~ 4.12

Astra Trident は、 Google Kubernetes Engine （ GKE ）、 Amazon Elastic Kubernetes Services （ EKS ）、 Azure Kubernetes Service （ AKS ）、 Rancher 、 VMware Tanzu Portfolio など、フルマネージドで自己管理型の Kubernetes サービスが数多く提供されています。

> Astra Tridentがインストールされ ているKubernetesクラスタを1.24から1.25以降にアップグレードする前に、を参照してください "Helm ベースのオペレータインストレーションをアップグレードします"。

サポートされるバックエンド（ストレージ）

Astra Trident を使用するには、次のバックエンドを 1 つ以上サポートする必要があります。

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

機能の要件

次の表は、このリリースの Astra Trident で利用できる機能と、サポートする 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？
CSI Trident	1.21 ~ 1.27	いいえ
ボリューム Snapshot	1.21 ~ 1.27	いいえ
ボリューム Snapshot からの PVC	1.21 ~ 1.27	いいえ

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？
iSCSI PV のサイズ変更	1.21 ~ 1.27	いいえ
ONTAP 双方向 CHAP	1.21 ~ 1.27	いいえ
動的エクスポートポリシー	1.21 ~ 1.27	いいえ
Trident のオペレータ	1.21 ~ 1.27	いいえ
CSI トポロジ	1.21 ~ 1.27	いいえ

テスト済みのホストオペレーティングシステム

Astra 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、動作確認済みのものは次のとおりです。

- OpenShift Container Platform（AMD64およびARM64）でサポートされているRed Hat CoreOS（RHCOS）のバージョン
- RHEL 8+（AMD64およびARM64）
- Ubuntu 22.04以降（AMD64およびARM64）
- Windows Server 2019（AMD64）

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

。 `tridentctl` ユーティリティは、これらのLinuxディストリビューションでも動作します。

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ツールまたはiSCSI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：

Astra Tridentでは、バックエンド構成でストレージシステムを使用する前に、変更が必要になる場合があります。

"バックエンドを設定"

Astra Trident ポート

Astra Tridentが通信するには、特定のポートへのアクセスが必要です。

"Astra Trident ポート"

コンテナイメージと対応する Kubernetes バージョン

エアギャップのある環境では、 Astra Trident のインストールに必要なコンテナイメージを次の表に示します。を使用します `tridentctl images` 必要なコンテナイメージのリストを確認するコマンド。

Kubernetes のバージョン	コンテナイメージ
v1.21.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）
v1.22.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）

Kubernetes のバージョン	コンテナイメージ
v1.3.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）
v1.24.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）
v1.25.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）

Kubernetes のバージョン	コンテナイメージ
v1.26.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）
v1.27.0	• Docker.io/NetApp/trident：23.04.0 • docker.io / netapp/trident-autosupport：23.04 • registry.k8s.io/sig-storage/csi-provisioner：v3.4.1 • registry.k8s.io/sig-storage/csi-attacher：v4.2.0 • registry.k8s.io/sig-storage/csi-resizer：v1.7.0 • registry.k8s.io/sig-storage/csi-snapshotter：v6.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar：v2.7.0 • docker.io/netapp/trident-operator：23.04.0（オプション）

Kubernetesバージョン1.21以降では、検証済みの `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x` イメージは、の場合にのみ作成します `v1` のバージョンがを処理しています `volumesnapshots.snapshot.storage.k8s.gcr.io` CRD。状況に応じて `v1beta1` バージョンは、の有無にかかわらず、CRDに対応しています `v1` バージョン：検証済みを使用します `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x` イメージ（Image）：

Astra Trident をインストール

Astra Tridentのインストール方法をご確認ください

ネットアップでは、Astra Tridentをさまざまな環境や組織に導入できるように、複数のインストールオプションを提供しています。Tridentは、Tridentオペレータ（手動またはHelmを使用）またはで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。

Astra Tridentに関する重要な情報23.04

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

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

作業を開始する前に

インストールパスに関係なく、次のものが必要です。

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

に慣れていない場合は "基本概念"今こそ、そのための絶好の機会です。

インストール方法を選択します

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

Trident演算子を使用する

Tridentのオペレータは、手動で導入する場合でも、Helmを使用する場合でも、Astra Tridentのリソースを動的に管理して簡単にインストールできます。それは可能である "Tridentのオペレータ環境をカスタマイズ" で属性を使用する `TridentOrchestrator` カスタムリソース（CR）。

Tridentオペレータには次のようなメリットがあります。

** Astra Tridentオブジェクト作成**

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

**** 自己回復機能****

OperatorはAstra Tridentのインストールを監視し、導入が削除されたときや誤って変更された場合などの問題に対処するための手段をアクティブに講じます。A `trident-operator-<generated-id>` ポッドが作成され、が関連付けられます `TridentOrchestrator` Astra Tridentをインストールした CR。これにより、クラスタ内にAstra 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することで、インストールがべき等の状態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、 Astra Trident を有効にしてデバッグログを生成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、にパッチを適用します `TridentOrchestrator` をクリックして設定します `spec.debug` 終了： `true`：

```
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、新しいポッドの作成がトリガーされ、それに応じてインストールが変更される場合があります。

** 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、オペレータが既存の Astra 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確実に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、オペレータによって Astra Trident はインストールされません。Astra Trident がすでにオペレータとともにインストールされている場合、サポート対象外の Kubernetes バージョンに Astra 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。

** NetApp Consoleを使用した Kubernetes クラスタ管理**

NetApp Consoleを使用するAstra Tridentでは、最新バージョンのAstra Tridentにアップグレードしたり、ストレージ クラスを追加および管理して作業環境に接続したり、 Cloud Backup Serviceを使用して永続ボリュームをバックアップしたりできます。コンソールは、手動または Helm を使用して、 Tridentオペレーターを使用したAstra Trident のデプロイメントをサポートします。

を使用します `tridentctl`

既存の環境をアップグレードする必要がある場合や、高度にカスタマイズすることを検討している場合は、アップグレードを検討する必要があります 。これは、従来の方法であった Astra Trident を導入する方法です。

可能です Tridentリソースのマニフェストを生成するには、次の手順を実行します導入、開始、サービスアカウント、 Astra Trident がインストールの一部として作成するクラスタロールが含まれます。

22.04 リリース以降、 Astra Trident がインストールされるたびに AES キーが再生成されなくなりました。今回のリリースでは、 Astra Trident がインストールする新しいシークレットオブジェクトが、インストール全体で維持されます。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、それより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、Astra Tridentをインストールする最も簡単な方法であり、ネットワークの制限を課すことの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、必要なTridentを格納するためにデフォルトのレジストリが使用されます (`docker.io`）とCSIを参照してください (`registry.k8s.io`）イメージ。

標準モードを使用すると、Astra Tridentインストーラは次のように動作します。

- インターネット経由でコンテナイメージを取得します
- 導入環境またはノードのデプロイを作成し、Kubernetesクラスタ内のすべての対象ノードでAstra Tridentポッドがスピンアップします

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。

- 適切なバージョンのを導入します `kubectl` Astra Tridentの導入元となるリモートマシン。
- Kubernetesクラスタから構成ファイルをコピーし、を設定します `KUBECONFIG` リモートマシンの環境変数。
- を開始します `kubectl get nodes` コマンドを使用して、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。

- Astra 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ます。を使用してを導入した場合 `tridentctl`を使用する場合は、適切なバージョンのを使用する必要があります `tridentctl` Astra Tridentをアンインストールするためのバイナリ。同様に、演算子を使用してを配置する場合は、を編集する必要があります `TridentOrchestrator` CRおよびSET `spec.uninstall=true` Astra Tridentをアンインストールする方法
- オペレータベースの導入環境で、削除して代わりにを使用する場合は `tridentctl` Astra Tridentを導入するには、まずを編集する必要があります `TridentOrchestrator` をクリックして設定します `spec.uninstall=true` Astra Tridentをアンインストールする方法次に、を削除します `TridentOrchestrator` オペレータによる導入も可能です。その後、を使用してをインストールできます `tridentctl`。
- オペレータベースの手動導入環境で、HelmベースのTridentオペレータ環境を使用する場合は、最初に手動でオペレータをアンインストールしてからHelmインストールを実行する必要があります。これにより、Helm は必要なラベルとアノテーションを使用して Trident オペレータを導入できます。これを行わないと、 Helm ベースの Trident オペレータの導入が失敗し、ラベル検証エラーとアノテーション検証エラーが表示されます。を使用する場合は `tridentctl`- Helmベースの展開を使用すると、問題を発生させずに導入できます。

その他の既知の設定オプション

VMware Tanzu Portfolio 製品に Astra Trident をインストールする場合：

- クラスタが特権ワークロードをサポートしている必要があります。
- 。 `--kubelet-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。デフォルトはです `/var/vcap/data/kubelet`。

  を使用してkubeletの場所を指定します `--kubelet-dir` は、Trident Operator、Helm、およびで動作することがわかっています `tridentctl` 導入：

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オペレータが手動で導入してAstra Tridentをインストールできます。このプロセスでは、環境 をインストールする際に、Astra Tridentで必要なコンテナイメージがプライベートレジストリに格納されません。プライベートイメージレジストリがある場合は、を使用します "オフライン導入のプロセス"。

Astra Tridentに関する重要な情報23.04

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

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

Tridentオペレータを手動で導入し、**Trident**をインストール

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

作業を開始する前に

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

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

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

Astra Tridentインストーラパッケージには、Tridentオペレータの導入とAstra Tridentのインストールに必要なものがすべて含まれています。から最新バージョンのTridentインストーラをダウンロードして展開します

"GitHubの_Assets_sectionを参照してください"。

```
wget https://github.com/NetApp/trident/releases/download/v23.04.0/trident-
installer-23.04.0.tar.gz
tar -xf trident-installer-23.04.0.tar.gz
cd trident-installer
```

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

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

Astra Tridentインストーラには、オペレータのインストールや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。このバンドルファイルを使用すると、オペレータを簡単に導入し、デフォルトの設定でAstra Tridentをインストールできます。

- クラスタでKubernetes 1.24以前を実行している場合は、を使用します `bundle_pre_1_25.yaml`。
- 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>* is `bundle_pre_1_25` または `bundle_post_1_25` 使用しているKubernetesのバージョンに基づきます。

  ```
  cp kustomization_<bundle>.yaml 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> `bundle_pre_1_25` または `bundle_post_1_25` 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**： `TridentOrchestrator` **Trident**をインストール

これで、を作成できます `TridentOrchestrator` Astra Tridentを導入必要に応じて、を実行できます "Tridentのインストールをカスタマイズ" で属性を使用する `TridentOrchestrator` 仕様

```
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
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:23.04
    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:23.04.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v23.04.0
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	このツールを使用してAstra Tridentをインストールしている `TridentOrchestrator` CR。
インストール済み	Astra Trident のインストールが完了しました。
アンインストール中です	OperatorはAstra Tridentをアンインストールしています。理由はです `spec.uninstall=true`。
アンインストール済み	Astra Trident がアンインストールされました。
失敗しました	オペレータがインストール、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 Astra Trident。オペレータはこの状態からのリカバリを自動的に試行します。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。もう一つ 存在します。

ポッドの作成ステータスを使用する

作成したポッドのステータスを確認することで、Astra 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`

を使用できます `tridentctl` インストールされているAstra Tridentのバージョンを確認します。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.04.0        | 23.04.0        |
+----------------+----------------+
```

次のステップ

できるようになりました。 "バックエンドとストレージクラスを作成し、ボリュームをプロビジョニングして、ポッドにボリュームをマウントします"。

Tridentオペレータを手動で導入（オフラインモード）

Tridentオペレータが手動で導入してAstra Tridentをインストールできます。このプロセスでは、環境 をインストールする際に、Astra Tridentで必要なコンテナイメージがプライベートレジストリに格納されます。プライベートイメージレジストリがない場合は、を使用します "標準的な導入のプロセス"。

Astra Tridentに関する重要な情報**23.04**

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

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

Tridentオペレータを手動で導入し、**Trident**をインストール

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

作業を開始する前に

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

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

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

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

```
wget https://github.com/NetApp/trident/releases/download/v23.04.0/trident-
installer-23.04.0.tar.gz
tar -xf trident-installer-23.04.0.tar.gz
cd trident-installer
```

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

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

インチ `/deploy/operator.yaml`、を更新します `image: docker.io/netapp/trident-`

operator:23.04.0 イメージレジストリの場所を反映します。。 "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例：

- image: <your-registry>/trident-operator:23.04.0 すべての画像が1つのレジストリにある場合。
- image: <your-registry>/netapp/trident-operator:23.04.0 TridentイメージがCSIイメージとは別のレジストリにある場合。

ステップ**4**：**Trident**オペレータを導入

Astra Tridentインストーラには、オペレータのインストールや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。このバンドルファイルを使用すると、オペレータを簡単に導入し、デフォルトの設定でAstra Tridentをインストールできます。

- クラスタでKubernetes 1.24以前を実行している場合は、を使用します bundle_pre_1_25.yaml。
- 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>* is bundle_pre_1_25 または bundle_post_1_25 使用しているKubernetesのバージョンに基づきます。

  ```
  cp kustomization_<bundle>.yaml 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> bundle_pre_1_25 または bundle_post_1_25 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 kustomize deploy/ > 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:23.04"
tridentImage: "<your-registry>/trident:23.04.0"
```

異なるレジストリ内の画像

を追加する必要があります `sig-storage` に移動します `imageRegistry` 別のレジストリの場所を使用します。

```
imageRegistry: "<your-registry>/sig-storage"
autosupportImage: "<your-registry>/netapp/trident-autosupport:23.04"
tridentImage: "<your-registry>/netapp/trident:23.04.0"
```

手順**6**： `TridentOrchestrator` **Trident**をインストール

これで、を作成できます `TridentOrchestrator` Astra Tridentを導入必要に応じて、さらに行うことができます "Tridentのインストールをカスタマイズ" で属性を使用する `TridentOrchestrator` 仕様次の例は、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>/netapp/trident-autosupport:23.04
  Debug:              true
  Image Registry:     <your-registry>/sig-storage
  Namespace:          trident
  Trident Image:      <your-registry>/netapp/trident:23.04.0
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/netapp/trident-
autosupport:23.04
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Http Request Timeout:       90s
    Image Pull Secrets:
    Image Registry:       <your-registry>/sig-storage
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
    Trident Image:        <your-registry>/netapp/trident:23.04.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v23.04.0
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	このツールを使用してAstra Tridentをインストールしている `TridentOrchestrator` CR。
インストール済み	Astra Trident のインストールが完了しました。
アンインストール中です	OperatorはAstra Tridentをアンインストールしています。理由はです `spec.uninstall=true`。
アンインストール済み	Astra Trident がアンインストールされました。
失敗しました	オペレータがインストール、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 Astra Trident。オペレータはこの状態からのリカバリを自動的に試行します。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。もう一つ 存在します。

ポッドの作成ステータスを使用する

作成したポッドのステータスを確認することで、Astra 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`

を使用できます `tridentctl` インストールされているAstra Tridentのバージョンを確認します。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.04.0        | 23.04.0        |
+----------------+----------------+
```

次のステップ

できるようになりました。 "バックエンドとストレージクラスを作成し、ボリュームをプロビジョニングして、ポッドにボリュームをマウントします"。

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Astra Tridentをインストールできます。このプロセスでは、環境 をインストールする際に、Astra Tridentで必要なコンテナイメージがプライベートレジストリに格納されません。プライベートイメージレジストリがある場合は、を使用します "オフライン導入のプロセス"。

Astra Tridentに関する重要な情報23.04

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

  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**Astra Trident**をインストール

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

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

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

手順

1. Astra 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. 使用 `helm install` をクリックし、次の例に示すように、導入環境の名前を指定します `23.04.0` は、インストールするAstra Tridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 23.04.0
--create-namespace --namespace <trident-namespace>
```

> Tridentのネームスペースを作成済みの場合は、を参照してください `--create-namespace` パラメータでネームスペースが追加で作成されることはありません。

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには、のように指定します `debug`をクリックし、次のコマンドを実行します `--set` コマンドを入力します `23.04.0` は、インストールするAstra Tridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 23.04.0
--create-namespace --namespace --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	
`affinity`	ポッド割り当てのアフィニティ	
`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	のレジストリを指定します trident-operator、trident、およびその他の画像。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。	""
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	kubeletの内部状態のホスト位置を上書きできます。	"/var/lib/kubeelet"
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。	true
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Astra Tridentを動作させることができます。	false
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	Astra Tridentの定期的なAutoSupport レポートを無効にできます。	false
tridentAutosupportImageTag	Astra Trident AutoSupport コンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Astra TridentのAutoSupport コンテナがHTTPプロキシ経由で自宅に通信できるようになります。	""
tridentLogFormat	Astra Tridentのログ形式を設定します (text または json）。	"text"

オプション	説明	デフォルト
`tridentDisableAuditLog`	Astra Trident監査ロガーを無効にします。	`true`
`tridentLogLevel`	Astra Trident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"`
`tridentDebug`	Astra Tridentのログレベルをに設定できます `debug`。	`false`
`tridentLogWorkflows`	特定のAstra Tridentワークフローを有効にして、トレースロギングやログ抑制を実行できます。	""
`tridentLogLayers`	特定のAstra Tridentレイヤでトレースロギングやログ抑制を有効にできます。	""
`tridentImage`	Astra Tridentのイメージを完全に上書きできます。	""
`tridentImageTag`	Astra 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`	WindowsワーカーノードにAstra Tridentをインストールできます。	`false`
`enableForceDetach`	強制切り離し機能を有効にできます。	`false`
`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。	`false`

コントローラポッドとノードポッドについて

Astra 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実行されます。Astra Tridentボリュームをマウントするすべての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。

次のステップ

できるようになりました。 "バックエンドとストレージクラスを作成し、ボリュームをプロビジョニングし

て、ポッドにボリュームをマウントします"。

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Astra Tridentをインストールできます。このプロセスでは、環境 をインストールする際に、Astra Tridentで必要なコンテナイメージがプライベートレジストリに格納されます。プライベートイメージレジストリがない場合は、を使用します "標準的な導入のプロセス"。

Astra Tridentに関する重要な情報23.04

- Astra Tridentに関する次の重要な情報をお読みください。*

** ：Trident ** に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

  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Astra Tridentをインストール

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

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

作業を開始する前に

に加えて "導入の前提条件" 必要です "Helm バージョン 3"。

手順

1. Astra 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. 使用 `helm install` 展開およびイメージレジストリの場所の名前を指定します。。 "TridentとCSIの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。例では、 `23.04.0` は、インストールするAstra 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
23.04.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace>
```

異なるレジストリ内の画像

を追加する必要があります `sig-storage` に移動します `imageRegistry` 別のレジストリの場所を使用します。

```
helm install <name> netapp-trident/trident-operator --version
23.04.0 --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=<your-registry>/netapp/trident-operator:23.04.0 --set
tridentAutosupportImage=<your-registry>/netapp/trident-
autosupport:23.04 --set tridentImage=<your-
registry>/netapp/trident:23.04.0 --create-namespace --namespace
<trident-namespace>
```

Tridentのネームスペースを作成済みの場合は、を参照してください `--create -namespace` パラメータでネームスペースが追加で作成されることはありません。

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

インストール中に設定データを渡す

インストール中に設定データを渡すには、次の 2 つの方法があります。

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

たとえば、のデフォルト値を変更するには、のように指定します `debug`をクリックし、次のコマンドを実行します `--set` コマンドを入力します `23.04.0` は、インストールするAstra Tridentのバージョンです。

```
helm install <name> netapp-trident/trident-operator --version 23.04.0
--create-namespace --namespace --set tridentDebug=true
```

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	
`affinity`	ポッド割り当てのアフィニティ	
`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。	
`imageRegistry`	のレジストリを指定します `trident-operator`、`trident`、およびその他の画像。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。	""
`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`	kubeletの内部状態のホスト位置を上書きできます。	"/var/lib/kubeelet"
`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"`

オプション	説明	デフォルト
operatorDebug	Tridentオペレータのログレベルをdebugに設定できます。	true
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Astra Tridentを動作させることができます。	false
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	Astra Tridentの定期的なAutoSupport レポートを無効にできます。	false
tridentAutosupportImageTag	Astra Trident AutoSupport コンテナのイメージのタグを上書きできます。	<version>
tridentAutosupportProxy	Astra TridentのAutoSupport コンテナがHTTPプロキシ経由で自宅に通信できるようになります。	""
tridentLogFormat	Astra Tridentのログ形式を設定します (text または json）。	"text"
tridentDisableAuditLog	Astra Trident監査ロガーを無効にします。	true
tridentLogLevel	Astra Trident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"
tridentDebug	Astra Tridentのログレベルをに設定できます debug。	false
tridentLogWorkflows	特定のAstra Tridentワークフローを有効にして、トレースロギングやログ抑制を実行できます。	""
tridentLogLayers	特定のAstra Tridentレイヤでトレースロギングやログ抑制を有効にできます。	""

オプション	説明	デフォルト
`tridentImage`	Astra Tridentのイメージを完全に上書きできます。	""
`tridentImageTag`	Astra 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`	WindowsワーカーノードにAstra Tridentをインストールできます。	`false`
`enableForceDetach`	強制切り離し機能を有効にできます。	`false`
`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。	`false`

コントローラポッドとノードポッドについて

Astra 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実行されます。Astra Tridentボリュームをマウントするすべての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。

次のステップ

できるようになりました。 "バックエンドとストレージクラスを作成し、ボリュームをプロビジョニングして、ポッドにボリュームをマウントします"。

Tridentオペレータのインストールをカスタマイズ

Tridentオペレータは、の属性を使用してAstra Tridentのインストールをカスタマイズできます `TridentOrchestrator` 仕様インストールをカスタマイズする場合は、それ以上のカスタマイズが必要です `TridentOrchestrator` 引数allow、使用を検討してください `tridentctl` 必要に応じて変更するカスタムYAMLマニフェストを生成します。

コントローラポッドとノードポッドについて

Astra 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実行されます。Astra Tridentボリュームをマウントするすべてのホストでノードポッドが実行されている必要があります。

Kubernetes "ノードセレクタ" および "寛容さと汚れ" は、特定のノードまたは優先ノードで実行されるように

ポッドを制限するために使用されます。「ControllerPlugin」およびを使用します `NodePlugin`を使用すると、拘束とオーバーライドを指定できます。

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

設定オプション

`spec.namespace` は、で指定します `TridentOrchestrator` Astra Tridentがインストールされているネームスペースを指定します。このパラメータ * は、 Astra Trident のインストール後に更新できません * 。これを実行すると、が実行されます `TridentOrchestrator` ステータスをに変更します `Failed`。Astra Tridentは、ネームスペース間での移行を意図していません。

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`	Astra Trident をインストールするネームスペース	デフォルト
`debug`	Astra Trident のデバッグを有効にします	いいえ
`enableForceDetach`	`ontap-san` および `ontap-san-economy` のみ。 KubernetesのNon-Graceful Node Shutdown（NGN）と連携して、ノードに障害が発生した場合に、マウントされたボリュームを含むワークロードを新しいノードに安全に移行する機能をクラスタ管理者に提供します。 *これは23.04の実験的な機能です。*を参照してください [フォースデタッチの詳細] を参照してください。	`false`
`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。	いいえ
`useIPv6`	IPv6 経由の Astra Trident をインストール	いいえ
`k8sTimeout`	Kubernetes 処理のタイムアウト	30 秒
`silenceAutosupport`	AutoSupportバンドルをNetAppに送信しない 自動	いいえ
`autosupportImage`	AutoSupport テレメトリのコンテナイメージ	"netapp/trident-autosupport：23.07"

パラメータ	説明	デフォルト
`autosupportProxy`	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`	Astra Trident のアンインストールに使用するフラグ	いいえ
`logFormat`	Astra Trident のログ形式が使用 [text 、 JSON]	テキスト（ Text ）
`tridentImage`	インストールする Astra Trident イメージ	"NetApp/Trident：23.07"
`imageRegistry`	形式の内部レジストリへのパス `<registry FQDN>[:port][/subpath]`	「k8s.gcr.io/sig-storage」（k8s 1.19以降） または"quay.io/k8scsi"
`kubeletDir`	ホスト上の kubelet ディレクトリへのパス	"/var/lib/kubeelet"
`wipeout`	完全な削除を実行するために削除するリソースのリスト Astra Trident	
`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`
`controllerPluginNodeSelector`	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

パラメータ	説明	デフォルト
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

ポッドパラメータの書式設定の詳細については、を参照してください "ポッドをノードに割り当てます"。

フォースデタッチの詳細

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

Astra TridentはKubernetes NGNに依存しているため、削除しないでください out-of-service 許容できないすべてのワークロードが再スケジュールされるまで、正常でないノードから影響を受けます。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クラスタ管理者がを適用したとき node.kubernetes.io/out-of-service=nodeshutdown:NoExecute ノードおよびに影響を与えます enableForceDetach がに設定されます `true`Astra Tridentがノードのステータスを判断し、次の処理を実行します。

1. そのノードにマウントされたボリュームのバックエンドI/Oアクセスを停止します。
2. Astra Tridentノードオブジェクトをにマークします dirty (新しい出版物には安全ではありません)。

Tridentコントローラは、（とマークされたあとに）ノードが再認定されるまで、新しいパブリッシュボリューム要求を拒否します dirty`をクリックします。マウントされたPVCでスケジュールされているワークロード（クラスタノードが正常で準備が完了したあとも含む）は、Astra Tridentがノードを検証できるまで受け入れられません `clean (新しい出版物のための安全)。

ノードの健常性が回復してtaintが削除されると、Astra 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. ノードがに含まれている場合 cleanable 状態（out-of-service taintが削除され、ノードがinになっています Ready 状態）。古い公開済みパスはすべてクリーンで、Astra Tridentはノードをとして再登録します clean 新しいボリュームのノードへの公開を許可します。

構成例

上記の属性は、を定義するときに使用できます TridentOrchestrator をクリックして、インストールをカスタマイズします。

例**1**：基本的なカスタム構成

次に、基本的なカスタム構成の例を示します。

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

例**2**：ノードセレクタを使用して導入します

次の例では、ノードセレクタを使用してTridentを導入する方法を示します。

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

例3：Windowsワーカーノードに導入する

この例は、Windowsワーカーノードへの導入を示しています。

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

tridentctlを使用してインストールします

tridentctlを使用してインストールします

を使用して、Astra Tridentをインストールできます `tridentctl`。このプロセスでは、Astra Tridentで必要なコンテナイメージがプライベートレジストリに格納されているかどうかに関係なく、環境 のインストールを実行します。をカスタマイズします `tridentctl` 配置については、を参照してください "tridentctl 展開をカスタマイズします"。

Astra Tridentに関する重要な情報23.04

- Astra Tridentに関する次の重要な情報をお読みください。*

 ：Trident に関する重要な情報

- TridentでKubernetes 1.27がサポートされるようになりました。Kubernetesをアップグレードする前にTridentをアップグレード
- Astra Tridentは、SAN環境でマルチパス構成を厳密に使用し、推奨される値をに設定します `find_multipaths: no` multipath.confファイル内。

  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Astra Tridentをインストールします `tridentctl`

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

作業を開始する前に

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

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

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

Astra Tridentインストーラパッケージは、Tridentポッドを作成し、そのステート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、プロビジョニングやクラスタホストへのボリュームの接続などのアクションを実行します。から最新バージョンのTridentインストーラをダウンロードして展開します "GitHubの_Assets_sectionを参照してください"。例では、選択した<trident-installer-XX.XX.X.tar.gz> Tridentバージョンを使用してupdate_Tridentを更新します。

```
wget https://github.com/NetApp/trident/releases/download/v23.04.0/trident-
installer-23.04.0.tar.gz
tar -xf trident-installer-23.04.0.tar.gz
cd trident-installer
```

手順**2**：**Astra Trident**をインストールする

を実行して、必要なネームスペースにAstra Tridentをインストールします `tridentctl install` コマンドを実行し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:23.04 --trident
-image <your-registry>/trident:23.04.0
```

異なるレジストリ内の画像

を追加する必要があります `sig-storage` に移動します `imageRegistry` 別のレジストリの場所を使用します。

```
./tridentctl install -n trident --image-registry <your-registry>/sig-
storage --autosupport-image <your-registry>/netapp/trident-
autosupport:23.04 --trident-image <your-
registry>/netapp/trident:23.04.0
```

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=23.04.0
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

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

```
kubectl get pods -n trident

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

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

を使用します `tridentctl`

を使用できます `tridentctl` インストールされているAstra Tridentのバージョンを確認します。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.04.0        | 23.04.0        |
+----------------+----------------+
```

構成例

例**1**：**Windows**ノードで**Astra Trident**の実行を有効にします

WindowsノードでAstra Tridentを実行できるようにするには、次の手順を実行します。

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

例**2**：強制切り離しを有効にします

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

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

次のステップ

できるようになりました。 "バックエンドとストレージクラスを作成し、ボリュームをプロビジョニングして、ポッドにボリュームをマウントします"。

tridentctlのインストールをカスタマイズします

Astra 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

Astra Tridentインストーラを使用して、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、を使用してリポジトリの場所を指定することを推奨します `--image-registry` スイッチ。の形式を指定します `<registry FQDN>[:port]`。

Kubernetesのディストリビューションを使用している場合 `kubelet` データを通常以外のパスに保持します `/var/lib/kubelet`を使用して、代替パスを指定できます `--kubelet-dir`。

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。を使用する `--generate-custom-yaml` パラメータは、インストーラのに次のYAMLファイルを作成します `setup` ディレクトリ：

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`
- `trident-service.yaml`
- `trident-namespace.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、を使用できます `--use-custom-yaml` をクリックして、カスタム導入環境をインストールします。

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

次の手順

Astra Tridentのインストールが完了したら、バックエンドの作成、ストレージクラスの作成、ボリュームのプロビジョニング、ポッドへのボリュームのマウントを実行できます。

手順 1 ：バックエンドを作成する

これで、 Astra Trident がボリュームのプロビジョニングに使用するバックエンドを作成できるようになります。これを行うには、を作成します `backend.json` 必要なパラメータを含むファイル。さまざまなバックエンドタイプの設定ファイルの例については、を参照してください `sample-input` ディレクトリ。

を参照してください "こちらをご覧ください" バックエンドタイプのファイルを設定する方法の詳細については、を参照してください。

```
cp sample-input/<backend template>.json backend.json
vi backend.json
```

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

作成に失敗した場合は、バックエンド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 -n trident logs
```

問題に対処したら、この手順の最初に戻ってやり直してください。トラブルシューティングのヒントについては、を参照してください "トラブルシューティング" セクション。

手順 2 ：ストレージクラスを作成する

Kubernetes ユーザは、を指定する Persistent Volume クレーム（ PVC ）を使用してボリュームをプロビジョニングします "ストレージクラス" 名前で検索できます。詳細情報はユーザには表示されませんが、ストレージクラスは、そのクラスに使用されるプロビジョニングツール（この場合は Trident ）と、そのクラスがプロビジョニングツールにもたらす意味を特定します。

ストレージクラスの Kubernetes ユーザがボリュームを必要なときに指定するストレージクラスを作成します。このクラスの構成では、前の手順で作成したバックエンドをモデリングし、 Astra Trident が新しいボリュームのプロビジョニングにこのバックエンドを使用するようにする必要があります。

をベースにしたストレージクラスが最もシンプルになりました `sample-input/storage-class-csi.yaml.templ` インストーラに付属のファイル *`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
```

これはKubernetesオブジェクトなので、を使用します `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
```

Kubernetes と Astra Trident の両方で、 * basic-csi * ストレージクラスが表示され、 Astra Trident がバックエンドのプールを検出しました。

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

手順 3 ：最初のボリュームをプロビジョニングします

これで、最初のボリュームを動的にプロビジョニングできます。これは Kubernetes を作成することで実現されます "永続的ボリュームの要求" （ PVC ）オブジェクト。

作成したストレージクラスを使用するボリュームの PVC を作成します。

を参照してください `sample-input/pvc-basic-csi.yaml` たとえば、のように指定します。ストレージクラス名が、作成した名前と一致していることを確認します。

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

kubectl get pvc --watch
NAME      STATUS    VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS   AGE
basic     Pending
basic          1s
basic     Pending   pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   0
basic          5s
basic     Bound     pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   1Gi
RWO           basic          7s
```

手順 4 ：ボリュームをポッドにマウントする

次に、ボリュームをマウントします。nginxポッドを起動し、の下にPVをマウントします `/usr/share/nginx/html`。

```
cat << EOF > task-pv-pod.yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
EOF
kubectl create -f task-pv-pod.yaml
```

```
# Wait for the pod to start
kubectl get pod --watch

# Verify that the volume is mounted on /usr/share/nginx/html
kubectl exec -it task-pv-pod -- df -h /usr/share/nginx/html

# Delete the pod
kubectl delete pod task-pv-pod
```

この時点でポッド（アプリケーション）は存在しなくなりますが、ボリュームはまだ存在しています。必要に応じて、別のポッドから使用できます。

ボリュームを削除するには、要求を削除します。

```
kubectl delete pvc basic
```

これで、次のような追加タスクを実行できます。

- "追加のバックエンドを設定"
- "追加のストレージクラスを作成する。"

Trident で Astra を管理

Astra Trident をアップグレード

Astra Trident をアップグレード

Astra Trident は四半期ごとにリリースサイクルを実施し、毎年 4 つのメジャーリリースをリリースしています。各新しいリリースは、以前のリリースに基づいてビルドされ、新機能とパフォーマンスの強化に加え、バグの修正や改善点が追加されています。ネットアップでは、Astra Tridentの新機能を活用するために、1年に1回以上アップグレードすることを推奨しています。

アップグレード前の考慮事項

最新リリースの Astra Trident にアップグレードする際は、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、Astra 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 20.01 以降では、のベータ版のみが提供されます "ボリューム Snapshot" がサポートされます。Kubernetes 管理者は、従来のアルファスナップショットを保持するために、アルファスナップショットオブジェクトを安全にバックアップするか、ベータ版に変換するように注意する必要があります。
  - CSI のボリュームスナップショットは、 Kubernetes 1.20 以降の GA 機能になりました。アップグレードする前に、を使用してアルファスナップショットCRDを削除する必要があります `tridentctl obliviate alpha-snapshot-crd` アルファスナップショット仕様のCRDを削除します。
  - ボリュームSnapshotのベータリリースでは、一連のCRDとSnapshotコントローラが変更されています。どちらもAstra Tridentをアップグレードする前にセットアップ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クラスタをアップグレードする前に知っておくべきこと"。
- バージョン19.04以前からのアップグレードでは、いずれもAstra Tridentメタデータを自社から移行する必要があります `etcd` をCRDオブジェクトに追加します。を確認します "Astra Tridentリリースに固有のドキュメント" を参照して、アップグレードの仕組みを確認してください。
- アップグレードするときは、この作業を行うことが重要です `parameter.fsType` インチ `StorageClasses` Astra Tridentが使用。削除して再作成することができます `StorageClasses` 実行前のボリュームの中断はなし。
  - 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

Astra Tridentバージョンは日付ベースです `YY.MM` 命名規則。「YY」は年の最後の2桁、「MM」は月です。ドットリリースは、の後に続きます `YY.MM.X` 条約。ここで、「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、22.04（22.04.1などのDOTリリースを含む）から23.04に直接アップグレードできます。
- 以前のリリースを使用している場合は、具体的な手順について、該当するリリースのドキュメントを参照してアップグレードを実行してください。そのためには、最初に 4 つのリリースウィンドウに対応する最新リリースにアップグレードする必要があります。たとえば'18.07を実行していて'20.07リリースにアップグレードする場合は'次のように複数ステップのアップグレードプロセスを実行します
  a. 最初のアップグレードは 18.07 から 19.07 へ。
  b. その後 '19.07 から 20.07 にアップグレードします

> OpenShift Container PlatformでTridentオペレータを使用してアップグレードする場合は、Trident 21.01.1以降にアップグレードする必要があります。21.01.0 でリリースされた Trident オペレータには、 21.01.1 で修正された既知の問題が含まれています。詳細については、を参照してください "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

通常は、最初のインストールと同じ方法でアップグレードする必要がありますが、可能です "インストール方法を切り替えます"。

Astra Tridentの最初のインストールに使用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Astra 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、Astra Tridentは手動またはHelm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、を使用します `kubectl describe tproc trident` をクリックし、Helmを使用してAstra Tridentがインストールされたかどうかを確認します。
   - Helmラベルがある場合は、Helmを使用してAstra Tridentがインストールされています。
   - Helmラベルがない場合は、Astra TridentをTridentオペレータを使用して手動でインストールしています。

ステップ**3**：アップグレード方法を選択します

Astra Tridentは2つの方法でアップグレードできます。

オペレータを使用してアップグレードするタイミング

可能です "Tridentオペレータを使用してアップグレードします" 次の場合：

- オペレータまたはを使用してAstra Tridentを最初にインストールした `tridentctl`。
- CSI Tridentをアンインストールしても、インストールのメタデータは保持されます。
- CSIベースのAstra Tridentがインストールされている。の19.07以降のすべてのリリースはCSIベースです。Tridentネームスペース内のポッドを調べてバージョンを確認できます。

- 23.01より前のバージョンでは、次の名前が使用されていました。 `trident-csi-*`
- 23.01以降でポッドの命名には次のものが使用されます。
  - `trident-controller-<generated id>` コントローラポッドの場合
  - `trident-node-<operating system>-<generated id>` ノードポッドの場合
  - `trident-operator-<generated id>` オペレータポッド用

を使用している場合は、Tridentのアップグレードにオペレータを使用しないでください `etcd` Tridentリリース（19.04以前）。

を使用してアップグレードするタイミング `tridentctl`

可能です 「tridenctl」を使用してAstra Tridentを最初にインストールした場合。

`tridentctl` は従来のAstra Tridentのインストール方法であり、複雑なカスタマイズを必要とするお客様に最適なオプションを提供します。詳細については、を参照してください "インストール方法を選択します"。

演算子に変更があります

Astra Tridentの21.01リリースでは、運用者のアーキテクチャが次のように変更されました。

- 演算子は * cluster を対象とした * になりました。Trident 演算子の以前のインスタンス（バージョン 20.04 ～ 20.10 ）は、 * 名前空間スコープ * でした。クラスタを対象としたオペレータが有利な理由は次のとおりです。
  - リソースのアカウンタビリティ：オペレータは、 Astra 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するようになりました。Astra Tridentのインストールの一環として、オペレータはを使用して複数のリソースを作成し、管理します `ownerReferences`。メンテナンス `ownerReferences` クラスタを対象としたリソースでは、OpenShiftなどの特定のKubernetesディストリビュータでエラーが発生する可能性があります。これは、クラスタを対象としたオペレータによって緩和されます。Trident リソースの自動修復とパッチ適用には、この要件が不可欠です。
  - アンインストール中のクリーンアップ： Astra Trident を完全に削除するには、関連するリソースをすべて削除する必要があります。ネームスペースを対象としたオペレータが、クラスタを対象としたリソース（ clusterRole 、 ClusterRoleBinding 、 PodSecurityPolicy など）の削除で問題が発生し、クリーンアップが完了しない場合があります。クラスタを対象としたオペレータがこの問題を排除し、必要に応じて、 Astra Trident を完全にアンインストールし、 Aresh をインストールできます。
- `TridentProvisioner` がに置き換えられました `TridentOrchestrator` Astra Tridentのインストールと管理に使用したカスタムリソース。また、に新しいフィールドが導入されます `TridentOrchestrator` 仕様Tridentのネームスペースは、を使用してからインストールまたはアップグレードするように指定できます `spec.namespace` フィールド。例を見てみましょう "こちらをご覧ください"。

オペレータにアップグレードしてください

オペレータを使用して手動またはHelmを使用して、既存のAstra Tridentインストールを簡単にアップグレードできます。

Tridentオペレータを使用してアップグレード

通常は、最初にインストールしたときと同じ方法でAstra Tridentをアップグレードする必要があります。レビュー "アップグレード方法を選択します" をクリックしてください。

> ネームスペースを対象としたオペレータ（バージョン20.07~20.10）を使用してインストールされたAstra Tridentのインスタンスからアップグレードする場合、Tridentオペレータは自動的に次の処理を行います。
>
> - 移行 `tridentProvisioner` をに追加します `tridentOrchestrator` 同じ名前のオブジェクト
> - を削除します `TridentProvisioner` オブジェクトと `tridentprovisioner` CRD
> - Astra Tridentを、使用しているクラスタを対象としたオペレータのバージョンにアップグレードします
> - 最初にインストールされていたネームスペースと同じ場所にAstra Tridentをインストール

クラスタを対象としたTridentオペレータ環境をアップグレード

クラスタを対象としたTridentオペレータインストールをアップグレードできます。すべてのAstra Tridentバージョン21.01以降では、クラスタを対象とした演算子を使用します。

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. Astra Tridentのバージョンを確認します。

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

2. 現在の Astra Trident インスタンスのインストールに使用した Trident オペレータを削除たとえば、22.01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 22.01/trident-installer/deploy/bundle.yaml -n trident
```

3. を使用して初期インストールをカスタマイズした場合 `TridentOrchestrator` 属性を編集できます `TridentOrchestrator` インストールパラメータを変更するオブジェクト。これには、ミラーリングされたTridentおよびCSIイメージレジストリをオフラインモードに指定したり、デバッグログを有効にしたり、イメージプルシークレットを指定したりするための変更が含まれます。
4. 環境に適したバンドルYAMLファイルとAstra Tridentバージョンを使用してAstra Tridentをインストールします。たとえば、Kubernetes 1.27用Astra Trident 23.04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 23.04.0/trident-installer/deploy/bundle_post_1_25.yaml
-n trident
```

Tridentでは、オペレータのインストールやKubernetesバージョンに関連するオブジェクトの作成に使用できるバンドルファイルが提供されています。

- クラスタでKubernetes 1.24以前を実行している場合は、を使用します "Bundle_pre_1_25.yaml"。
- 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"。

結果

Tridentのオペレータが、既存のAstra Tridentインストールを特定し、オペレータと同じバージョンにアップグレードします。

名前空間を対象としたオペレータインストールをアップグレードします

ネームスペースを対象としたオペレータ（バージョン20.07~20.10）を使用してインストールされたAstra Tridentのインスタンスから、クラスタを対象としたオペレータを対象としたインストールにアップグレードできます。

作業を開始する前に

からnamespace-scoped演算子をデプロイするために使用するバンドルYAMLファイルが必要です `https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/BUNDLE.YAML` ここで、 *vXX.XX* は、バージョン番号およびです *BUNDLE.YAML* はバンドルYAMLファイル名です。

手順

1. を確認します `TridentProvisioner` 既存のTridentインストールのステータスはです `Installed`。

```
kubectl describe tprov trident -n trident | grep Message: -A 3

Message:  Trident installed
Status:   Installed
Version:  v20.10.1
```

ステータスがになっている場合 `Updating`をクリックし、続行する前に解決してください。可能なステータス値のリストについては、を参照してください "こちらをご覧ください"。

2. を作成します `TridentOrchestrator` Tridentインストーラに付属のマニフェストを使用したCRD。

```
# Download the release required [23.04.0]
mkdir 23.04.0
cd 23.04.0
wget
https://github.com/NetApp/trident/releases/download/v23.04.0/trident-
installer-23.04.0.tar.gz
tar -xf trident-installer-23.04.0.tar.gz
cd trident-installer
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. マニフェストを使用して、名前空間を対象とした演算子を削除します。
   a. 正しいディレクトリにいることを確認します。

```
pwd
/root/20.10.1/trident-installer
```

   b. namespace-scoped演算子を削除します。

```
kubectl delete -f deploy/<BUNDLE.YAML> -n trident

serviceaccount "trident-operator" deleted
clusterrole.rbac.authorization.k8s.io "trident-operator" deleted
clusterrolebinding.rbac.authorization.k8s.io "trident-operator"
deleted
deployment.apps "trident-operator" deleted
podsecuritypolicy.policy "tridentoperatorpods" deleted
```

   c. Tridentオペレータが削除されたことを確認します。

```
kubectl get all -n trident

NAME                                  READY   STATUS    RESTARTS   AGE
pod/trident-csi-68d979fb85-dsrmn   6/6     Running   12         99d
pod/trident-csi-8jfhf              2/2     Running   6          105d
pod/trident-csi-jtnjz              2/2     Running   6          105d
pod/trident-csi-lcxvh              2/2     Running   8          105d

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)              AGE
service/trident-csi    ClusterIP   10.108.174.125   <none>
34571/TCP,9220/TCP   105d

NAME                           DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR                                        AGE
daemonset.apps/trident-csi   3         3         3       3
3           kubernetes.io/arch=amd64,kubernetes.io/os=linux   105d

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/trident-csi    1/1     1            1           105d

NAME                                      DESIRED   CURRENT   READY
AGE
replicaset.apps/trident-csi-68d979fb85    1         1         1
105d
```

4. （オプション）インストールパラメータを変更する必要がある場合は、を更新します `TridentProvisioner` 仕様これには、の値を変更するなどの変更が含まれます `tridentImage`、`autosupportImage`、プライベートイメージリポジトリ、および提供 `imagePullSecrets`)名前空間を対象とした演算子を削除した後、クラスタを対象とした演算子をインストールする前。更新可能なパラメータの一覧については、を参照してください "設定オプション"。

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

5. クラスタを対象としたTridentオペレータをインストールします。
   a. 正しいディレクトリにいることを確認します。

```
pwd
/root/23.04.0/trident-installer
```

   b. 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インストールします。

Tridentでは、オペレータのインストールやKubernetesバージョンに関連するオブジェクトの作成に使用できるバンドルファイルが提供されています。

- クラスタでKubernetes 1.24以前を実行している場合は、を使用します "Bundle_pre_1_25.yaml"。
- 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"。

```
kubectl create -f deploy/<BUNDLE.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

#All tridentProvisioners will be removed, including the CRD itself
kubectl get tprov -n trident
Error from server (NotFound): Unable to list "trident.netapp.io/v1,
Resource=tridentprovisioners": the server could not find the
requested resource (get tridentprovisioners.trident.netapp.io)

#tridentProvisioners are replaced by tridentOrchestrator
kubectl get torc
NAME      AGE
trident   13s
```

c. ネームスペースのTridentポッドを確認します。。 `trident-controller` ポッド名は、23.01で導入された命名規則を反映しています。

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS
AGE
trident-controller-79df798bdc-m79dc      6/6     Running   0
1m41s
trident-node-linux-xrst8                 2/2     Running   0
1m41s
trident-operator-5574dbbc68-nthjv        1/1     Running   0
1m52s
```

d. Tridentが目的のバージョンに更新されていることを確認

```
kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v23.04.0
```

Helm ベースのオペレータインストレーションをアップグレードします

Helm ベースのオペレータインストレーションをアップグレードするには、次の手順を実行します。

> Astra 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、value.yamlを更新して設定する必要があります `excludePodSecurityPolicy` 終了： `true` または、を追加します `--set excludePodSecurityPolicy=true` に移動します `helm upgrade` コマンドを実行してからクラスタをアップグレードしてください。

手順

1. 最新の Astra Trident リリースをダウンロード
2. を使用します `helm upgrade` コマンドを入力します `trident-operator-23.04.0.tgz` アップグレード後のバージョンが反映されます。

```
helm upgrade <name> trident-operator-23.04.0.tgz
```

> 初期インストール時にデフォルト以外のオプションを設定した場合（TridentイメージおよびCSIイメージのプライベートなミラーレジストリを指定するなど）は、を使用します `--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。
>
> たとえば、のデフォルト値を変更するには、のように指定します `tridentDebug`を使用して、次のコマンドを実行します。
>
> ```
> helm upgrade <name> trident-operator-23.04.0-custom.tgz --set
> tridentDebug=true
> ```

3. 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。

結果

Tridentのオペレータが、既存のAstra Tridentインストールを特定し、オペレータと同じバージョンにアップグレードします。

オペレータ以外のインストールからアップグレードします

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：

手順

1. 最新の Astra Trident リリースをダウンロード

```
# Download the release required [23.04.0]
mkdir 23.04.0
cd 23.04.0
wget
https://github.com/NetApp/trident/releases/download/v22.01.1/trident-
installer-23.04.0.tar.gz
tar -xf trident-installer-23.04.0.tar.gz
cd trident-installer
```

2. を作成します `tridentorchestrator` マニフェストからのCRD。

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

3. クラスタ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.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. を作成します `TridentOrchestrator` Astra Tridentのインストール用にCR。

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

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m
trident-csi-xrst8                   2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0          5m41s
```

5. Tridentが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:                v23.04.0
```

結果

既存のバックエンドと PVC は自動的に使用可能

tridentctl を使用してアップグレードします

を使用すると、既存のAstra Tridentインストールを簡単にアップグレードできます `tridentctl`。

を使用して**Astra Trident**をアップグレードします `tridentctl`

Astra 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、 Astra Trident 環境で使用されている Persistent Volume Claim （ PVC ；永続的ボリューム要求）と Persistent Volume （ PV ；永続的ボリューム）は削除されません。Astra Trident がオフラインの間は、すでにプロビジョニング済みの PVS を引き続き使用でき、 Astra Trident は、オンラインに戻った時点で作成された PVC に対してボリュームをプロビジョニングします。

作業を開始する前に

レビュー "アップグレード方法を選択します" を使用してアップグレードする前に `tridentctl`。

手順

1. のアンインストールコマンドを実行します `tridentctl` CRDと関連オブジェクトを除くAstra Tridentに関連付けられているすべてのリソースを削除する。

```
./tridentctl uninstall -n <namespace>
```

2. Astra Tridentを再インストールします。を参照してください "tridentctl を使用して Astra Trident をインストールします"。

> 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。

を使用してボリュームをアップグレードする `tridentctl`

アップグレード後は、新しいTridentリリースで提供される豊富な機能（オンデマンドのボリュームSnapshotなど）を利用できます。を使用してボリュームをアップグレードできます `tridentctl upgrade` コマンドを実行します

レガシーボリュームがある場合は、NFSまたはiSCSIタイプからCSIタイプにアップグレードして、Astra Tridentのすべての新機能を使用する必要があります。Trident によってプロビジョニングされたレガシー PV は、従来の機能セットをサポートします。

作業を開始する前に

ボリュームをCSIタイプにアップグレードすることを決定する前に、次の点を考慮してください。

- 場合によっては、すべてのボリュームをアップグレードする必要はありません。以前に作成したボリュームには引き続きアクセスでき、正常に機能します。
- PV は、アップグレード時に展開 / 起動可能セットの一部としてマウントできます。展開 / 起動セットを停止する必要はありません。
- アップグレード時に、スタンドアロンの POD に PV を接続することはできません。ボリュームをアップグレードする前に、ポッドをシャットダウンする必要があります。
- アップグレードできるのは、 PVC にバインドされているボリュームだけです。PVC にバインドされていないボリュームは、アップグレード前に削除およびインポートする必要があります。

手順

1. を実行します `kubectl get pv` をクリックしてPVSをリスト表示します。

```
kubectl get pv
NAME                        CAPACITY     ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                   STORAGECLASS    REASON   AGE
default-pvc-1-a8475         1073741824   RWO            Delete
Bound    default/pvc-1          standard                19h
default-pvc-2-a8486         1073741824   RWO            Delete
Bound    default/pvc-2          standard                19h
default-pvc-3-a849e         1073741824   RWO            Delete
Bound    default/pvc-3          standard                19h
default-pvc-4-a84de         1073741824   RWO            Delete
Bound    default/pvc-4          standard                19h
trident                     2Gi          RWO            Retain
Bound    trident/trident                                19h
```

現在、Trident 20.07によって作成されたPVSのうちの4つが、を使用しています `netapp.io/trident` プロビジョニング担当者：

2. を実行します `kubectl describe pv` PVの詳細を確認します。

```
kubectl describe pv default-pvc-2-a8486

Name:            default-pvc-2-a8486
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: netapp.io/trident
                 volume.beta.kubernetes.io/storage-class: standard
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Bound
Claim:           default/pvc-2
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        1073741824
Node Affinity:   <none>
Message:
Source:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    10.xx.xx.xx
    Path:      /trid_1907_alpha_default_pvc_2_a8486
    ReadOnly:  false
```

PVはを使用して作成されました `netapp.io/trident` プロビジョニング担当者とプロビジョニングタイプはNFSです。Astra Trident が提供する新機能をすべてサポートするには、この PV を CSI タイプにアップグレードする必要があります。

3. を実行します `tridentctl upgrade volume <name-of-trident-volume>` 従来のAstra Tridentボリュームを**CSI**仕様にアップグレードするコマンド。

```
./tridentctl get volumes -n trident
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
|            NAME      |  SIZE   | STORAGE CLASS | PROTOCOL |
BACKEND UUID             | STATE  | MANAGED |
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-3-a849e | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-1-a8475 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
| default-pvc-4-a84de | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+

./tridentctl upgrade volume default-pvc-2-a8486 -n trident
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
|            NAME      |  SIZE   | STORAGE CLASS | PROTOCOL |
BACKEND UUID             | STATE  | MANAGED |
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
| default-pvc-2-a8486 | 1.0 GiB | standard      | file     | c5a6f6a4-
b052-423b-80d4-8fb491a14a22 | online | true    |
+----------------------+---------+---------------+----------
+--------------------------------------+--------+---------+
```

4. を実行します `kubectl describe pv` ボリュームが**CSI**ボリュームであることを確認します。

```
kubectl describe pv default-pvc-2-a8486
Name:            default-pvc-2-a8486
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: csi.trident.netapp.io
                 volume.beta.kubernetes.io/storage-class: standard
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Bound
Claim:           default/pvc-2
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        1073741824
Node Affinity:   <none>
Message:
Source:
    Type:              CSI (a Container Storage Interface (CSI) volume
source)
    Driver:            csi.trident.netapp.io
    VolumeHandle:      default-pvc-2-a8486
    ReadOnly:          false
    VolumeAttributes:      backendUUID=c5a6f6a4-b052-423b-80d4-
8fb491a14a22

internalName=trid_1907_alpha_default_pvc_2_a8486
                           name=default-pvc-2-a8486
                           protocol=file
Events:                <none>
```

Astra Trident をアンインストール

Astra Trident のインストール方法に応じて、複数の方法でアンインストールできます。

Helm を使用してアンインストールします

Helmを使用してAstra 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 Astra 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
```

Trident オペレータを使用してをアンインストールします

Operator を使用して Astra Trident をインストールした場合、次のいずれかの方法で Trident をアンインストールできます。

- 編集 **`TridentOrchestrator`** アンインストールフラグを設定するには：を編集できます `TridentOrchestrator` をクリックして設定します `spec.uninstall=true`。を編集します `TridentOrchestrator` CRおよびを設定します `uninstall` 次のような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。必要に応じて、TridentOrchestratorをクリーンアップし、新しいTridentOrchestratorを作成する必要があります。
  Tridentをもう一度インストールします。

- 削除 **`TridentOrchestrator`**:を削除する `TridentOrchestrator` Astra Tridentの導入に使用したCRでは、Tridentをアンインストールするようオペレータに指示します。オペレータがの削除を処理します `TridentOrchestrator` さらに、Astra Tridentの導入とデプロイを削除し、インストールの一部として作成したTridentポッドを削除します。
  Astra Tridentを完全に削除し（作成したCRDを含む）、スレートを効果的に消去するには、編集します `TridentOrchestrator` を渡します `wipeout` オプション次の例を参照してください。

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

Astra Trident が完全にアンインストールされ、管理対象のバックエンドとボリュームに関連するすべてのメタデータがクリアされます。以降のインストールは新規インストールとして扱われます。

> 完全なアンインストールを実行する場合にのみ、 CRD の消去を検討してください。この操作は元に戻せません。 最初からやり直す必要がある場合や、 **Astra Trident** の新規インストールを作成する場合を除き、 **CRD** を消去しないでください

を使用してをアンインストールします tridentctl

を実行します uninstall のコマンド tridentctl 次のように、Astra 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。ただし、CRDと関連オブジェクトは削除されます。そのため、インストーラを再実行して、より新しいバージョンに簡単に更新できます。

```
./tridentctl uninstall -n <namespace>
```

Astra Trident の完全な削除を実行するには、 Astra Trident によって作成された CRD のフィナライザを削除し、 CRD を削除する必要があります。

Trident をダウングレード

旧バージョンの Astra Trident にダウングレードする手順をご確認ください。

ダウングレードするタイミング

次のような理由でダウングレードを検討してください。

- 危機管理計画
- アップグレードの結果として見つかったバグの即時修正
- 依存関係の問題、失敗したアップグレード、および不完全なアップグレード

CRD を使用する Astra Trident リリースに移行する場合は、ダウングレードを検討する必要があります。Astra Tridentは、ステートの維持にCRDを使用するため、作成されたすべてのストレージエンティティ（バックエンド、ストレージクラス、PV、ボリュームスナップショット）には、に書き込まれるデータではなく、関連するCRDオブジェクトが含まれています trident PV（以前にインストールしたAstra Tridentのバージョンで使用）新しく作成された PVS 、バックエンド、およびストレージクラスはすべて CRD オブジェクトとして管理されます。

CRD（19.07以降）を使用して実行されているAstra Tridentのバージョンのダウングレードのみを試みます。これにより、ダウングレードの実行後に、現在のAstra Tridentリリースで実行された処理を確認できます。

ダウングレードしない場合

を使用するTridentのリリースにダウングレードしないでください etcd 状態を維持するため（19.04以前）。現在の Astra Trident リリースで実行したすべての処理は、ダウングレード後に反映されません。以前のバージョンに戻す場合、新しく作成した PVS は使用できません。バックエンド、 PVS 、ストレージクラス、ボリューム Snapshot （作成 / 更新 / 削除）などのオブジェクトに加えられた変更は、以前のバージョンに戻すと Astra Trident には表示されません。以前のバージョンに戻しても、アップグレードされていないかぎり、以前のリリースを使用してすでに作成された PVS へのアクセスは中断されません。

Operator を使用して Astra Trident をインストールする場合のダウングレードプロセス

Trident Operatorを使用したインストールの場合、ダウングレードプロセスは異なり、を使用する必要はありません tridentctl。

Trident オペレータを使用してインストールを完了した場合は、 Astra Trident を次のいずれかにダウングレー

ドできます。

- 名前空間を対象とした演算子（ 20.07-2010 ）を使用してインストールされるバージョン。
- クラスタを対象とした演算子（ 21.01 以降）を使用してインストールされるバージョン。

クラスタを対象とした演算子にダウングレードします

Astra Trident を、クラスタを対象としたオペレータを使用するリリースにダウングレードするには、次の手順に従います。

手順

1. "Astra Trident をアンインストール"。既存のインストールを完全に削除する場合を除き、**CRD**は削除しないでください。
2. Tridentのオペレータは、ご使用のバージョンに関連付けられているオペレータマニフェストを使用することで削除できます。例： `https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/bundle.yaml` ここで、 `vXX.XX` は、バージョン番号です（例 `v22.10`）および `bundle.yaml` はバンドルYAMLファイル名です。
3. 必要なバージョンの Astra Trident をインストールして、ダウングレードを続行します。目的のリリースのマニュアルに従ってください。

名前空間を対象とした演算子にダウングレードします

このセクションでは、名前空間を対象とした演算子を使用してインストールされる、 20.07 ～ 20.10 の範囲の Astra Trident リリースへのダウングレード手順を要約します。

手順

1. "Astra Trident をアンインストール"。 既存のインストールを完全に削除する場合を除き、 **CRD** を削除しないでください。
   次を確認します。 `tridentorchestrator` が削除されました。

```
#Check to see if there are any tridentorchestrators present
kubectl get torc
NAME        AGE
trident     20h

#Looks like there is a tridentorchestrator that needs deleting
kubectl delete torc trident
tridentorchestrator.trident.netapp.io "trident" deleted
```

2. Tridentのオペレータは、ご使用のバージョンに関連付けられているオペレータマニフェストを使用することで削除できます。例： `https://github.com/NetApp/trident/tree/stable/vXX.XX/deploy/bundle.yaml` ここで、 `vXX.XX` は、バージョン番号です（例 `v22.10`）および `bundle.yaml` はバンドルYAMLファイル名です。
3. を削除します `tridentorchestrator` CRD。

```
#Check to see if ``tridentorchestrators.trident.netapp.io`` CRD is
present and delete it.

kubectl get crd tridentorchestrators.trident.netapp.io

NAME                                          CREATED AT
tridentorchestrators.trident.netapp.io   2021-01-21T21:11:37Z

kubectl delete crd tridentorchestrators.trident.netapp.io

customresourcedefinition.apiextensions.k8s.io
"tridentorchestrators.trident.netapp.io" deleted
```

Astra Trident がアンインストールされました。

4. 目的のバージョンをインストールしてダウングレードを続行します。目的のリリースのマニュアルに従ってください。

Helm を使用してダウングレードしてください

ダウングレードするには、を使用します `helm rollback` コマンドを実行します次の例を参照してください。

```
helm rollback trident [revision #]
```

を使用して**Astra Trident**をインストールした場合のダウングレードプロセス `tridentctl`

を使用してAstra Tridentをインストールした場合 `tridentctl`をダウングレードするには、次の手順を実行します。このシーケンスに従って、 Astra Trident 21.07 から 20.07 に移行するためのダウングレードプロセスを順を追って説明します。

ダウングレードを開始する前に、Kubernetesクラスタのスナップショットを作成する必要があります `etcd`。これにより、 Astra Trident の CRD の現在の状態をバックアップできます。

手順

1. を使用してTridentがインストールされていることを確認します `tridentctl`。Astra Trident のインストール方法がわからない場合は、次の簡単なテストを実行してください。
   a. Trident ネームスペースにあるポッドを表示します。
   b. クラスタで実行されている Astra Trident のバージョンを特定します。を使用できます `tridentctl` または、Tridentポッドで使用されるイメージを見てみましょう。
   c. 「* A」が表示されない場合 `tridentOrchestrator`、（または）A `tridentprovisioner`、（または）という名前のポッド `trident-operator-xxxxxxxxxx-xxxxx`を使用して、`Astra Trident *`をインストールします `tridentctl`。

2. 既存のを使用してAstra Tridentをアンインストール `tridentctl` バイナリ。 この場合は、 21.07 バイナリを使用してアンインストールします。

```
tridentctl version -n trident
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.07.0        | 21.07.0        |
+----------------+----------------+

tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted Trident daemonset.
INFO Deleted Trident service.
INFO Deleted Trident secret.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Deleted pod security policy.
podSecurityPolicy=tridentpods
INFO The uninstaller did not delete Trident's namespace in case it is
going to be reused.
INFO Trident uninstallation succeeded.
```

3. これが完了したら、希望するバージョンの Trident バイナリ（この例では 20.07 ）を取得し、 Astra Trident のインストールに使用します。のカスタム YAML を生成できます "カスタマイズされたインストール" 必要に応じて、

```
cd 20.07/trident-installer/
./tridentctl install -n trident-ns
INFO Created installer service account.
serviceaccount=trident-installer
INFO Created installer cluster role.               clusterrole=trident-
installer
INFO Created installer cluster role binding.
clusterrolebinding=trident-installer
INFO Created installer configmap.                  configmap=trident-
installer
...
...
INFO Deleted installer cluster role binding.
INFO Deleted installer cluster role.
INFO Deleted installer service account.
```

ダウングレードプロセスが完了します。

Astra Trident を使用

ワーカーノードを準備します

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

NFSツール

NFSツールを使用している場合は、次の手順でインストールします。 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `gcp-cvs`

iSCSIツール

使用する場合はiSCSIツールをインストールします。 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`

> 最新バージョンのRed Hat CoreOSには、デフォルトでNFSとiSCSIがインストールされています。

ノードサービスの検出

Astra Tridentは、ノードでiSCSIサービスや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

Astra Tridentが、検出されたサービスを特定するためのイベントをノードに対して作成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

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

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

iSCSIの自己回復機能

ONTAP システムでは、Astra TridentがiSCSIの自己修復機能を5分ごとに実行し、以下を実現します。

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。Astra Tridentが、修理の優先順位と、修理に先手を打つタイミングを判断
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。

自己回復アクティビティのログはにあります `trident-main` 各Demonsetポッドにコンテナを配置します。ログを表示するには、を設定しておく必要があります `debug` Astra Tridentのインストール中に「true」に設定。

Astra Tridentの自動修復機能は、次のような問題を防止します。

- 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。古いセッションの場合、Astra Tridentは7分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

iSCSIツールをインストール

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

作業を開始する前に

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

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

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

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
```

> 確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 `defaults`。

5. を確認します `iscsid` および `multipathd` 実行中：

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

6. を有効にして開始します `iscsi`：

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

Ubuntu

1. 次のシステムパッケージをインストールします。

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

2. open-iscsi 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
```

> 確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 `defaults`。

5. を確認します `open-iscsi` および `multipath-tools` 有効になっていて実行中：

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

> Ubuntu 18.04の場合は、ターゲットポートをで検出する必要があります `iscsiadm` 開始する前に `open-iscsi` iSCSIデーモンを開始します。または、を変更することもできます `iscsi` サービスを開始します `iscsid` 自動的に。

> iSCSI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。

バックエンドを設定

バックエンドを設定

バックエンドは、 Astra Trident とストレージシステムの関係を定義します。Trident がス

トレージシステムとの通信方法を Trident から指示し、 Astra Trident がボリュームをプロビジョニングする方法も解説します。

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

- "Azure NetApp Files バックエンドを設定します"
- "Cloud Volumes Service for Google Cloud Platform バックエンドを設定します"
- "NetApp HCI または SolidFire バックエンドを設定します"
- "ONTAPまたはCloud Volumes ONTAP NASドライバを使用したバックエンドの設定"
- "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 Astra 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 （ANF）をAstra Tridentのバックエンドとして設定できます。ANFバックエンドを使用してNFSボリュームとSMBボリュームを接続できます。

考慮事項

- Azure NetApp Files サービスでは、100GB未満のボリュームはサポートされません。100 GB のボリュームが小さい場合は、 Trident が自動的に作成します。
- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート

Azure NetApp Files バックエンドを設定する準備をします

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

NFSボリュームと**SMB**ボリュームの前提条件

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

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

- 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"。
- `subscriptionID` Azure NetApp Files を有効にしたAzureサブスクリプションから選択します。
- `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"。
- A "カスタム投稿者ロール" をサブスクリプションレベルで選択します (`assignableScopes`)以下のアクセス許可は、Astra Tridentが必要とするものに限定されます。カスタムロールを作成したあと、"Azureポータルを使用してロールを割り当てます"。

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

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

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

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

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

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

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

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/writ
e",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/dele
te",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/rea
d",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/wri
te",
```

```
"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/del
ete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/subvolumes/Get
Metadata/action",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTargets/r
ead",
                    "Microsoft.Network/virtualNetworks/read",
                    "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrations
/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。

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 2019を実行しているKubernetesクラスタ。Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- 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Astra Tridentのシークレットが少なくとも1つ含まれています。シークレットを生成します `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

ANF用のNFSとSMBのバックエンド構成オプションについて説明し、設定例を確認してください。

バックエンド構成オプション

Astra 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、要求された場所で利用可能で、要求されたサービスレベルとサブネットに一致する容量プールにANFボリュームを作成します。

Astra Trident は、手動 QoS 容量プールをサポートしていません。

ANF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
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	「 azure-NetApp-files 」
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + ランダムな文字
`subscriptionID`	Azure サブスクリプションのサブスクリプション ID	
`tenantID`	アプリケーション登録からのテナント ID	
`clientID`	アプリケーション登録からのクライアント ID	
`clientSecret`	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	
`serviceLevel`	の1つ `Standard`、 `Premium`または `Ultra`	"" （ランダム）

パラメータ	説明	デフォルト
location	新しいボリュームを作成する Azure の場所の名前	
resourceGroups	検出されたリソースをフィルタリングするためのリソースグループのリスト	"[]" （フィルタなし）
netappAccounts	検出されたリソースをフィルタリングするための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	""
subnet	に委任されたサブネットの名前 Microsoft.Netapp/volumes	""
networkFeatures	ボリューム用のVNet機能のセットです。の場合もあります Basic または Standard。 ネットワーク機能は一部の地域では使用できず、サブスクリプションで有効にする必要がある場合があります。を指定します networkFeatures この機能を有効にしないと、ボリュームのプロビジョニングが失敗します。	""
nfsMountOptions	NFS マウントオプションのきめ細かな制御。 SMBボリュームでは無視されます。 NFSバージョン4.1を使用してボリュームをマウントするには、を参照してください nfsvers=4 カンマで区切って複数のマウントオプションリストを指定し、NFS v4.1 を選択します。 ストレージクラス定義で設定されたマウントオプションは、バックエンド構成で設定されたマウントオプションよりも優先されます。	"nfsvers=3 "
limitVolumeSize	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）

パラメータ	説明	デフォルト
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例：`\{"api": false, "method": true, "discovery": true}`。トラブルシューティングを行って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`

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

必要な権限とリソース

PVCの作成時に「No capacity pools found」エラーが発生した場合、アプリケーション登録に必要な権限とリソース（サブネット、仮想ネットワーク、容量プール）が関連付けられていない可能性があります。デバッグが有効になっている場合、Astra Tridentはバックエンドの作成時に検出されたAzureリソースをログに記録します。適切なロールが使用されていることを確認します。

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

。 `resourceGroups`、 `netappAccounts`および `capacityPools` 値は、検出されたリソースのセットをこのストレージバックエンドで使用可能なリソースに制限するフィルタであり、任意の組み合わせで指定できます。完全修飾名の形式は次のとおりです。

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

ボリュームのプロビジョニング

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

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームに対するエクスポートルール `exportRule` CIDR表記のIPv4アドレスまたはIPv4サブネットの任意の組み合わせをカンマで区切って指定する必要があります。 SMBボリュームでは無視されます。	"0.0.0.0/0 "
`snapshotDir`	.snapshot ディレクトリの表示を制御します	いいえ
`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

例 **1** ：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、 ANF に委譲されたネットアップアカウント、容量プール、サブネットがすべて検出され、それらのプールまたはサブネットの 1 つに新しいボリュームがランダムに配置されます。理由 `nasType` は省略されています `nfs` デフォルトが適用され、バックエンドがNFSボリュームにプロビジョニングされます。

この構成は、 ANF の利用を開始して何を試してみるときに理想的ですが、実際には、プロビジョニングするボリュームの範囲をさらに設定することを検討しています。

```
---
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
```

例 **2** ：容量プールフィルタを使用した特定のサービスレベル設定

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

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

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

例**4**：仮想プールの構成

このバックエンド構成では、 1 つのファイルに複数のストレージプールを定義します。これは、異なるサービスレベルをサポートする複数の容量プールがあり、それらを表すストレージクラスを Kubernetes で作成する場合に便利です。プールを区別するために、仮想プールのラベルを使用しました `performance`。

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

ストレージクラスの定義

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

を使用した定義の例 `parameter.selector` フィールド

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

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

SMBボリュームの定義例

を使用します `nasType`、 `node-stage-secret-name`および `node-stage-secret-namespace`を使用して、SMBボリュームを指定し、必要なActive Directoryクレデンシャルを指定できます。

例**1**：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"
```

例**2**：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}
```

例**3**：ボリュームごとに異なるシークレットを使用する

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

`nasType: `smb` SMBボリュームをサポートするプールでフィルタリングします。 `nasType: `nfs` または `nasType: `null` NFSプールに対してフィルタを適用します。

バックエンドを作成します

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

Google CloudバックエンドにCloud Volumes Service を設定します

ネットアップCloud Volumes Service for Google CloudをAstra Tridentのバックエンドとして構成する方法を、提供されている構成例を使用して説明します。

Cloud Volumes Service for Google Cloudに対するAstra Tridentサポートの詳細をご確認ください

TridentがCloud Volumes Service ボリュームを作成できるのは、2つのうちの1つです "サービスタイプ"：

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

必要なもの

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

- NetApp Cloud Volumes Service で設定されたGoogle Cloudアカウント

- Google Cloud アカウントのプロジェクト番号
- を使用するGoogle Cloudサービスアカウント `netappcloudvolumes.admin` ロール
- Cloud Volumes Service アカウントのAPIキーファイル

バックエンド構成オプション

各バックエンドは、 1 つの Google Cloud リージョンにボリュームをプロビジョニングします。他のリージョンにボリュームを作成する場合は、バックエンドを追加で定義します。

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	"GCP-cvs"
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + API キーの一部
`storageClass`	CVSサービスタイプを指定するためのオプションのパラメータ。 使用 `software` をクリックしてCVSサービスタイプを選択します。それ以外の場合は、Astra TridentがCVSパフォーマンスサービスのタイプを引き継ぎます（`hardware`）。	
`storagePools`	CVSサービスタイプのみ。ボリューム作成用のストレージプールを指定するオプションのパラメータ。	
`projectNumber`	Google Cloud アカウントのプロジェクト番号。この値は、Google Cloudポータルのホームページにあります。	
`hostProjectNumber`	共有VPCネットワークを使用する場合は必須です。このシナリオでは、 `projectNumber` は、サービスプロジェクトです `hostProjectNumber` は、ホストプロジェクトです。	

パラメータ	説明	デフォルト
`apiRegion`	Astra TridentがCloud Volumes Service ボリュームを作成するGoogle Cloudリージョン。複数リージョンのKubernetesクラスタを作成する場合は、に作成されたボリューム `apiRegion` 複数のGoogle Cloudリージョンのノードでスケジュールされたワークロードで使用できます。 リージョン間トラフィックは追加コストを発生させます。	
`apiKey`	を使用したGoogle Cloudサービスアカウントの APIキー `netappcloudvolumes.admin` ロール。 このレポートには、 Google Cloud サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツが含まれています（バックエンド構成ファイルにそのままコピーされます）。	
`proxyURL`	CVSアカウントへの接続にプロキシサーバが必要な場合は、プロキシURLを指定します。プロキシサーバには、 HTTP プロキシまたは HTTPS プロキシを使用できます。 HTTPS プロキシの場合、プロキシサーバで自己署名証明書を使用するために証明書の検証はスキップされます。 認証が有効になっているプロキシサーバはサポートされていません。	
`nfsMountOptions`	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します。	"" （デフォルトでは適用されません）

パラメータ	説明	デフォルト
`serviceLevel`	新しいボリュームのCVS -パフォーマンスレベルまたはCVSサービスレベル。 CVS -パフォーマンスの値はです `standard`、 `premium`または `extreme`。 CVSの値はです `standardsw` または `zoneredundantstandardsw`。	CVS -パフォーマンスのデフォルトは「Standard」です。 CVSのデフォルトは"standardsw"です。
`network`	Cloud Volumes Service ボリュームに使用するGoogle Cloudネットワーク。	デフォルト
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： `\{"api":false, "method":true}`。 トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

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

ボリュームのプロビジョニングオプション

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

パラメータ	説明	デフォルト
`exportRule`	新しいボリュームのエクスポートルール。CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切って指定する必要があります。	"0.0.0.0/0 "

パラメータ	説明	デフォルト
`snapshotDir`	にアクセスします `.snapshot` ディレクトリ	いいえ
`snapshotReserve`	Snapshot 用にリザーブされているボリュームの割合	"" （ CVS のデフォルト値をそのまま使用）
`size`	新しいボリュームのサイズ。 CVS -パフォーマンス最小値は100GiBです。 CVS最小値は1GiBです。	CVS -パフォーマンスサービスのタイプはデフォルトで「100GiB」です。 CVSサービスのタイプではデフォルトが設定されませんが、1GiB以上が必要です。

CVS -パフォーマンスサービスの種類の例

次の例は、CVS -パフォーマンスサービスタイプの設定例を示しています。

例 **1** ：最小限の構成

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

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
```

```
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```

例**2**：サービスレベルの設定

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

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
```

```
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

例3：仮想プールの構成

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

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

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
```

```
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
    performance: extreme
    protection: extra
  serviceLevel: extreme
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
- labels:
    performance: extreme
    protection: standard
  serviceLevel: extreme
- labels:
    performance: premium
    protection: extra
  serviceLevel: premium
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
- labels:
    performance: premium
    protection: standard
  serviceLevel: premium
- labels:
    performance: standard
```

```
  serviceLevel: standard
```

ストレージクラスの定義

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

ストレージクラスの例

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

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

- 最初のストレージクラス (`cvs-extreme-extra-protection`）を最初の仮想プールにマッピングします。スナップショット予約が 10% の非常に高いパフォーマンスを提供する唯一のプールです。
- 最後のストレージクラス (`cvs-extra-protection`）スナップショット予約が10%のストレージプールを呼び出します。Tridentが、どの仮想プールを選択するかを決定し、スナップショット予約の要件が満たされていることを確認します。

CVSサービスタイプの例

次の例は、CVSサービスタイプの設定例を示しています。

例**1**：最小構成

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

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
```

```
  client_id: '123456789012345678901'
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
serviceLevel: standardsw
```

例**2**：ストレージプールの構成

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

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDaT+Oui9FBAw19
    L1AGEkrYU5xd9K5NlO5jMkIFND5wCD+Nv+jd1GvtFRLaLK5RvXyF5wzvztmODNS+
    qtScpQ+5cFpQkuGtv9U9+N6qtuVYYO3b504Kp5CtqVPJCgMJaK2j8pZTIqUiMum/
    5/Y9oTbZrjAHSMgJm2nHzFq2X0rqVMaHghI6ATm4DOuWx8XGWKTGIPlc0qPqJlqS
    LLaWOH4VIZQZCAyW5IUp9CAmwqHgdG0uhFNfCgMmED6PBUvVLsLvcq86X+QSWR9k
    ETqElj/sGCenPF7ti1DhGBFafd9hPnxg9PZY29ArEZwY9G/ZjZQX7WPgs0VvxiNR
    DxZRC3GXAgMBAAECggEACn5c59bG/qnVEVI1CwMAalM5M2z09JFhlLlljKwntNPj
    Vilw2eTW2+UE7HbJru/S7KQgA5Dnn9kvCraEahPRuddUMrD0vG4kTl/IODV6uFuk
    Y0sZfbqd4jMUQ21smvGsqFzwloYWS5qzO1W83ivXH/HW/iqkmY2eW+EPRS/hwSSu
    SscR+SojI7PB0BWSJhlV4yqYf3vcD/D95el2CVHfRCkL85DKumeZ+yHEnpiXGZAE
    t8xSs4a5OOPm6NHhevCw2a/UQ95/foXNUR450HtbjieJo5o+FF6EYZQGfU2ZHZO8
    37FBKuaJkdGW5xqaI9TL7aqkGkFMF4F2qvOZM+vy8QKBgQD4oVuOkJDlhkTHP86W
    esFlw1kpWyJR9ZA7LI0g/rVpslnX+XdDq0WQf4umdLNau5hYEH9LU6ZSGs1Xk3/B
    NHwR6OXFuqEKNiu83d0zSlHhTy7PZpOZdj5a/vVvQfPDMz7OvsqLRd7YCAbdzuQ0
    +Ahq0Ztwvg0HQ64hdW0ukpYRRwKBgQDgyHj98oqswoYuIa+pP1yS0pPwLmjwKyNm
    /HayzCp+Qjiyy7Tzg8AUqlH1Ou83XbV428jvg7kDhO7PCCKFq+mMmfqHmTpb0Maq
    KpKnZg4ipsqP1yHNNEoRmcailXbwIhCLewMqMrggUiLOmCw4PscL5nK+4GKu2XE1
    jLqjWAZFMQKBgFHkQ9XXRAJ1kR3XpGHoGN890pZOkCVSrqju6aUef/5KYlFCt8ew
    F/+aIxM2iQSvmWQYOvVCnhuY/F2GFaQ7d0om3decuwI0CX/xy7PjHMkLXa2uaZs4
    WR17sLduj62RqXRLX0c0QkwBiNFyHbRcpdkZJQujbYMhBa+7j7SxT4BtAoGAWMWT
    UucocRXZm/pdvz9wteNH3YDWnJLMxm1KC06qMXbBoYrliY4sm3ywJWMC+iCd/H8A
    Gecxd/xVu5mA2L2N3KMq18Zhz8Th0G5DwKyDRJgOQ0Q46yuNXOoYEjlo4Wjyk8Me
    +tlQ8iK98E0UmZnhTgfSpSNElbz2AqnzQ3MN9uECgYAqdvdVPnKGfvdtZ2DjyMoJ
    E89UIC41WjjJGmHsd8W65+3X0RwMzKMT6aZc5tK9J5dHvmWIETnbM+lTImdBBFga
    NWOC6f3r2xbGXHhaWSl+nobpTuvlo56ZRJVvVk7lFMsiddzMuHH8pxfgNJemwA4P
    ThDHCejv035NNV6KyoO0tA==
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
```

```
  auth_uri: https://accounts.google.com/o/oauth2/auth
  token_uri: https://oauth2.googleapis.com/token
  auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
  client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

次の手順

バックエンド構成ファイルを作成したら、次のコマンドを実行します。

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

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

```
tridentctl logs
```

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

NetApp HCI または SolidFire バックエンドを設定します

ネットアップが提供する Trident インストールで Element バックエンドを作成して使用する方法をご確認ください。

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"。

ボリュームモード

。 `solidfire-san` ストレージドライバは、ボリュームモード（fileとblock）の両方をサポートしています。をクリックします `Filesystem` volumeMode、Astra Tridentがボリュームを作成し、ファイルシステムを作成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
`solidfire-san`	iSCSI	ブロック	RWO 、 ROX 、 RWX	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。
`solidfire-san`	iSCSI	ブロック	RWO 、 ROX 、 RWX	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。
`solidfire-san`	iSCSI	ファイルシステム	RWO 、 ROX	`xfs`、 `ext3`、 `ext4`
`solidfire-san`	iSCSI	ファイルシステム	RWO 、 ROX	`xfs`、 `ext3`、 `ext4`

Astra Trident は強化された CSI プロビジョニング担当者として機能する場合、 CHAP を使用します。CSI のデフォルトである CHAP を使用している場合は、これ以上の準備は必要ありません。を明示的に設定することを推奨します `UseCHAP` CSI以外のTridentでCHAPを使用するオプション。それ以外は、を参照してください "こちらをご覧ください"。

ボリュームアクセスグループは、従来の非 CSI フレームワークである Astra Trident でのみサポートされています。CSI モードで動作するように設定されている場合、 Astra Trident は CHAP を使用します。

どちらでもない場合 `AccessGroups` または `UseCHAP` が設定され、次のいずれかのルールが適用されます。

- デフォルトの場合は `trident` アクセスグループが検出され、アクセスグループが使用されます。
- アクセスグループが検出されず、 Kubernetes バージョンが 1.7 以降の場合は、 CHAP が使用されます。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	常に「 solidfire-san-」
`backendName`	カスタム名またはストレージバックエンド	「 iSCSI_ 」 + ストレージ（ iSCSI ） IP アドレス SolidFire

パラメータ	説明	デフォルト
`Endpoint`	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP	
`SVIP`	ストレージ（ iSCSI ）の IP アドレスとポート	
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット。	「」
`TenantName`	使用するテナント名（見つからない場合に作成）	
`InitiatorIFace`	iSCSI トラフィックを特定のホストインターフェイスに制限します	デフォルト
`UseCHAP`	CHAP を使用して iSCSI を認証します	正しいです
`AccessGroups`	使用するアクセスグループ ID のリスト	「 trident 」という名前のアクセスグループの ID を検索します。
`Types`	QoS の仕様	
`limitVolumeSize`	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）
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true}	null

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。

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。その場合は、を使用して各ストレージクラスを使用するように定義します `IOPS` ストレージクラスのパラメータ。

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

例**2**：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。

Astra Tridentは、ストレージプール上にあるラベルを、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。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。

最初のストレージクラス (`solidfire-gold-four`）を選択すると、最初の仮想プールにマッピングされます。ゴールドのパフォーマンスを提供する唯一のプール `Volume Type QoS` 金の。最後のストレージクラス (`solidfire-silver`）Silverパフォーマンスを提供するストレージプールをすべて特定します。Tridentが、どの仮想プールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

```
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ドライバに関する重要な情報

Astra Controlは、で作成したボリュームに対して、シームレスな保護、ディザスタリカバリ、および移動（Kubernetesクラスタ間でボリュームを移動）を提供します `ontap-nas`、 `ontap-nas-flexgroup`および `ontap-san` ドライバ。を参照してください "Astra Controlレプリケーションの前提条件" を参照してください。

- を使用する必要があります `ontap-nas` データ保護、ディザスタリカバリ、モビリティを必要とする本番環境のワークロード向けのサービスです。
- 使用 `ontap-san-economy` 想定されるボリューム使用量がONTAP でサポートされる量よりも大幅に多い場合
- 使用 `ontap-nas-economy` 想定されるボリューム使用量が、ONTAP でサポートされるおよびよりも大幅に多い場合にのみ該当します `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

Tridentは、通常はを使用して、ONTAP 管理者またはSVM管理者のどちらかとして実行される必要があります `admin` クラスタユーザまたはです `vsadmin` SVMユーザ、または同じロールを持つ別の名前のユーザ。Amazon FSX for NetApp ONTAP 環境では、Astra Tridentは、クラスタを使用して、ONTAP 管理者またはSVM管理者のどちらかとして実行されるものと想定しています `fsxadmin` ユーザまたはです `vsadmin` SVMユーザ、または同じロールを持つ別の名前のユーザ。。 `fsxadmin` このユーザは、クラスタ管理者ユーザを限定的に置き換えるものです。

> を使用する場合 `limitAggregateUsage` クラスタ管理者権限が必要です。Amazon FSX for NetApp ONTAP をAstra Tridentとともに使用している場合は、を参照してください `limitAggregateUsage` パラメータはでは機能しません `vsadmin` および `fsxadmin` ユーザアカウント：このパラメータを指定すると設定処理は失敗します。

ONTAP 内では、 Trident ドライバが使用できるより制限的な役割を作成することもできますが、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。

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。

ONTAP バックエンドすべてに対して、 Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。

複数のドライバを実行し、 1 つまたは複数のドライバを参照するストレージクラスを作成することもできます。たとえば、を設定できます `san-dev` を使用するクラス `ontap-san` ドライバおよびA `san-default` を使用するクラス `ontap-san-economy` 1つ。

すべてのKubernetesワーカーノードに適切なiSCSIツール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

Astra Trident には、 ONTAP バックエンドを認証する 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。など、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 `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： Astra Trident は、バックエンドにインストールされた証明書を使用して ONTAP クラスタと通信することもできます。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式を切り替えることができます。ただし、一度にサポートされる認証方法は1つだけです。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

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

クレデンシャルベースの認証を有効にします

Trident が ONTAP バックエンドと通信するには、 SVM を対象とした管理者またはクラスタを対象とした管理者のクレデンシャルが必要です。などの標準の事前定義されたロールを使用することを推奨します `admin` または `vsadmin`。これにより、今後のリリースの ONTAP との互換性が今後のリリースの Astra Trident で使用される機能 API が公開される可能性があります。カスタムのセキュリティログインロールは Astra Trident で作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成または更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証を有効にします

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

- clientCertificate ： Base64 でエンコードされたクライアント証明書の値。
- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された CA を使用する場合は、このパラメータを指定する必要があります。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

手順

1. クライアント証明書とキーを生成します。生成時に、 ONTAP ユーザとして認証するように Common Name （ CN ；共通名）を設定します。

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

2. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っている可能性があります。信頼できる CA が使用されていない場合は無視します。

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

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

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

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

4. ONTAP セキュリティログインロールでサポートされていることを確認する `cert` 認証方式。

```
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、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 backend update`。

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

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

パスワード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した場合、 Astra 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ことを示しています。

双方向 **CHAP** を使用して接続を認証します

Astra Tridentは、に対して双方向CHAPを使用してiSCSIセッションを認証できます `ontap-san` および `ontap-san-economy` ドライバ。これには、を有効にする必要があり `useCHAP` バックエンド定義のオプション。に設定すると `true`、Astra Tridentは、SVMのデフォルトのイニシエータセキュリティを双方向CHAPに設定し、バックエンドファイルからのユーザ名とシークレットを設定します。接続の認証には双方向 CHAP を使用することを推奨します。次の設定例を参照してください。

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

。 `useCHAP` パラメータは、1回だけ設定できるブール値のオプションです。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。

に加えて `useCHAP=true`、 `chapInitiatorSecret`、 `chapTargetInitiatorSecret`、 `chapTargetUsername`および `chapUsername` フィールドはバックエンド定義に含める必要があります。を実行すると、バックエンドが作成されたあとでシークレットを変更できます `tridentctl update`。

動作の仕組み

を設定します `useCHAP` trueに設定すると、ストレージ管理者は、ストレージバックエンドでCHAPを設定するようにAstra Trident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、Astra Tridentはデフォルトのセキュリティタイプをに設定します `CHAP` CHAPイニシエータとターゲットのユーザ名および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ることはありません。
- 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、対応するがAstra Tridentによって作成されます `tridentbackend` CRDを実行し、CHAPシークレットとユーザ名をKubernetesシークレットとして保存します。このバックエンドの Astra Trident によって作成されたすべての PVS がマウントされ、 CHAP 経由で接続されます。

クレデンシャルをローテーションし、バックエンドを更新

CHAPクレデンシャルを更新するには、でCHAPパラメータを更新します `backend.json` ファイル。CHAPシークレットを更新し、を使用する必要があります `tridentctl update` 変更を反映するためのコマンドです。

バックエンドのCHAPシークレットを更新する場合は、を使用する必要があります `tridentctl` バックエンドを更新します。Astra Trident では変更を取得できないため、 CLI / ONTAP UI からストレージクラスタのクレデンシャルを更新しないでください。

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

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

既存の接続は影響を受けません。 SVM の Astra Trident でクレデンシャルが更新されても、引き続きアクティブです。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です。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 ドライバを作成して Astra Trident インストールで使用する方法をご確認ください。このセクションでは、バックエンド構成の例と、バックエンドをストレージクラスにマッピングする方法を詳しく説明します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + データ LIF
`managementLIF`	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス MetroClusterのシームレスなスイッチオーバーを実現するには、SVM管理LIFを指定する必要があります。 Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。 を使用してAstra Tridentをインストールした場合、IPv6アドレスを使用するようにを設定できます `--use-ipv6` フラグ。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。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
`dataLIF`	プロトコル LIF の IP アドレス。 * iSCSIには指定しないでください。* Astra Tridentが使用します "ONTAP の選択的LUNマップ" iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます `dataLIF` は明示的に定義されます。	SVMの派生物です
`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）。 をに設定します `true` Astra Tridentでは、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を設定して使用します。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" を参照してください。	`false`

パラメータ	説明	デフォルト
`chapInitiatorSecret`	CHAP イニシエータシークレット。の場合は必須です `useCHAP=true`	「」
`labels`	ボリュームに適用する任意の JSON 形式のラベルのセット	「」
`chapTargetInitiatorSecret`	CHAP ターゲットイニシエータシークレット。の場合は必須です `useCHAP=true`	「」
`chapUsername`	インバウンドユーザ名。の場合は必須です `useCHAP=true`	「」
`chapTargetUsername`	ターゲットユーザ名。の場合は必須です `useCHAP=true`	「」
`clientCertificate`	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	「」
`trustedCACertificate`	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。	「」
`username`	ONTAP クラスタとの通信に必要なユーザ名。クレデンシャルベースの認証に使用されます。	「」
`password`	ONTAP クラスタとの通信にパスワードが必要です。クレデンシャルベースの認証に使用されます。	「」
`svm`	使用する Storage Virtual Machine	SVMの場合に生成されます `managementLIF` を指定します
`storagePrefix`	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。 あとから変更することはできません。このパラメータを更新するには、新しいバックエンドを作成する必要があります。	`trident`

パラメータ	説明	デフォルト
`limitAggregateUsage`	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。 NetApp ONTAP バックエンドにAmazon FSXを使用している場合は、指定しないでください `limitAggregateUsage`。提供された `fsxadmin` および `vsadmin` アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。	“”（デフォルトでは適用されません）
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合、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`
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true} トラブルシューティングを行い、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リアンパラメータ。* テクニカルプレビュー * useREST は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると true`Astra 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へのアクセス権が必要です `ontap アプリケーション：これは事前定義されたによって満たされます vsadmin および cluster-admin ロール。 useREST は、MetroCluster ではサポートされていません。	false

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
spaceAllocation	space-allocation for LUN のコマンドを指定します	正しいです
spaceReserve	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
snapshotPolicy	使用する Snapshot ポリシー	なし

パラメータ	説明	デフォルト
qosPolicy	作成したボリュームに割り当てるQoS ポリシーグループ。ストレージプール / バックエンドごとにQOSPolicy または adaptiveQosPolicy のいずれかを選択します。 Trident が Astra 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。非共有のQoSポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	「」
adaptiveQosPolicy	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	「」
snapshotReserve	Snapshot「0」用にリザーブされているボリュームの割合	状況 snapshotPolicy は「none」、それ以外は「」です。
splitOnClone	作成時にクローンを親からスプリットします	いいえ
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです 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。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	いいえ
luksEncryption	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。	""
securityStyle	新しいボリュームのセキュリティ形式	unix

パラメータ	説明	デフォルト
`tieringPolicy`	「none」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」

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'
```

を使用して作成したすべてのボリューム `ontap-san` ドライバであるAstra Tridentが、FlexVolのメタデータに対応するために、さらに10%の容量を追加LUN は、ユーザが PVC で要求したサイズとまったく同じサイズでプロビジョニングされます。Astra Trident が FlexVol に 10% を追加（ ONTAP で利用可能なサイズとして表示）ユーザには、要求した使用可能容量が割り当てられます。また、利用可能なスペースがフルに活用されていないかぎり、 LUN が読み取り専用になることもありません。これは、 ONTAP と SAN の経済性には該当しません。

を定義するバックエンドの場合 `snapshotReserve`Tridentは、次のようにボリュームサイズを計算します。

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

1.1 は、 Astra Trident の 10% の追加料金で、 FlexVol のメタデータに対応します。の場合 `snapshotReserve` = 5%、PVC要求= 5GiB、ボリュームの合計サイズは5.79GiB、使用可能なサイズは5.5GiBです。。 `volume show` 次の例のような結果が表示されます。

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。

Amazon FSx on NetApp ONTAPとAstra 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 SANの最小構成の例

これは、 `ontap-san` ドライバ。

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

ONTAP SANエコノミーの最小構成の例

```
---
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>
```

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。

Astra Tridentは、[Comments]フィールドにプロビジョニングラベルを設定します。FlexVol にコメントが設定されます。Astra Tridentは、プロビジョニング時に仮想プール上にあるすべてのラベルをストレージボリュームにコピーします。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'
```

バックエンドを **StorageClasses** にマッピングします

次のStorageClass定義は、 [仮想プールを使用するバックエンドの例]。を使用する `parameters.selector` フィールドでは、各StorageClassがボリュームのホストに使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プール内で定義された要素があります。

- 。 `protection-gold` StorageClassは、 `ontap-san` バックエンド：ゴールドレベルの保護を提供する唯一のプールです。

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

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

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

- 。 `app-mysqldb` StorageClassは内の3番目の仮想プールにマッピングされます `ontap-san-economy` バックエンド：これは、mysqldbタイプアプリケーション用のストレージプール構成を提供する唯一のプールです。

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

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

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

- 。 `creditpoints-5k` StorageClassは内の3番目の仮想プールにマッピングされます `ontap-san` バックエンドと内の4番目の仮想プール `ontap-san-economy` バックエンド：これらは、5000クレジットポイントを持つ唯一のプールオファリングです。

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

Tridentが、どの仮想プールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

ONTAP NAS ドライバ

ONTAP NASドライバの概要

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

ONTAP NASドライバに関する重要な情報

Astra Controlは、で作成したボリュームに対して、シームレスな保護、ディザスタリカバリ、および移動（Kubernetesクラスタ間でボリュームを移動）を提供します `ontap-nas`、 `ontap-nas-flexgroup`および `ontap-san` ドライバ。を参照してください "Astra Controlレプリケーションの前提条件" を参照してください。

- を使用する必要があります `ontap-nas` データ保護、ディザスタリカバリ、モビリティを必要とする本番環境のワークロード向けのサービスです。
- 使用 `ontap-san-economy` 想定されるボリューム使用量がONTAP でサポートされる量よりも大幅に多い場合
- 使用 `ontap-nas-economy` 想定されるボリューム使用量が、ONTAP でサポートされるおよびよりも大幅に多い場合にのみ該当します `ontap-san-economy` ドライバは使用できません。
- 使用しないでください `ontap-nas-economy` データ保護、ディザスタリカバリ、モビリティのニーズが予想される場合。

ユーザ権限

Tridentは、通常はを使用して、ONTAP 管理者またはSVM管理者のどちらかとして実行される必要があります `admin` クラスタユーザまたはです `vsadmin` SVMユーザ、または同じロールを持つ別の名前のユーザ。

Amazon FSX for NetApp ONTAP 環境では、Astra Tridentは、クラスタを使用して、ONTAP 管理者またはSVM管理者のどちらかとして実行されるものと想定しています `fsxadmin` ユーザまたはです `vsadmin` SVMユーザ、または同じロールを持つ別の名前のユーザ。。 `fsxadmin` このユーザは、クラスタ管理者ユーザを限定的に置き換えるものです。

> を使用する場合 `limitAggregateUsage` クラスタ管理者権限が必要です。Amazon FSX for NetApp ONTAP をAstra Tridentとともに使用している場合は、を参照してください `limitAggregateUsage` パラメータはでは機能しません `vsadmin` および `fsxadmin` ユーザアカウント：このパラメータを指定すると設定処理は失敗します。

ONTAP内でTridentドライバが使用できる、より制限の厳しいロールを作成することは可能ですが、推奨しません。Trident の新リリースでは、多くの場合、考慮すべき API が追加で必要になるため、アップグレードが難しく、エラーも起こりやすくなります。

ONTAP NASドライバを使用してバックエンドを設定する準備をします

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

要件

- ONTAP バックエンドすべてに対して、 Astra Trident が SVM に少なくとも 1 つのアグリゲートを割り当てておく必要があります。
- 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：

- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート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 を参照してください。

ONTAPバックエンドの認証

Astra Trident には、 ONTAP バックエンドを認証する 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。など、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 `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： Astra Trident は、バックエンドにインストールされた証明書を使用して ONTAP クラスタと通信することもできます。この場合、バックエンド定義には、 Base64 でエンコードされたクライアント証明書、キー、および信頼された CA 証明書（推奨）が含まれている必要があります。

既存のバックエンドを更新して、クレデンシャルベースの方式と証明書ベースの方式を切り替えることができます。ただし、一度にサポートされる認証方法は1つだけです。別の認証方式に切り替えるには、バックエンド設定から既存の方式を削除する必要があります。

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

クレデンシャルベースの認証を有効にします

Trident が ONTAP バックエンドと通信するには、 SVM を対象とした管理者またはクラスタを対象とした管理者のクレデンシャルが必要です。などの標準の事前定義されたロールを使用することを推奨します `admin` または `vsadmin`。これにより、今後のリリースの ONTAP との互換性が今後のリリースの Astra Trident で使用される機能 API が公開される可能性があります。カスタムのセキュリティログインロールは Astra Trident で作成して使用できますが、推奨されません。

バックエンド定義の例は次のようになります。

YAML

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

JSON

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

バックエンド定義は、クレデンシャルがプレーンテキストで保存される唯一の場所であることに注意してください。バックエンドが作成されると、ユーザ名とパスワードが Base64 でエンコードされ、 Kubernetes シークレットとして格納されます。クレデンシャルの知識が必要なのは、バックエンドの作成と更新だけです。この処理は管理者専用で、 Kubernetes / ストレージ管理者が実行します。

証明書ベースの認証を有効にします

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

- clientCertificate ： Base64 でエンコードされたクライアント証明書の値。
- clientPrivateKey ： Base64 でエンコードされた、関連付けられた秘密鍵の値。
- trustedCACertifate: 信頼された CA 証明書の Base64 エンコード値。信頼された CA を使用する場合は、このパラメータを指定する必要があります。信頼された CA が使用されていない場合は無視してかまいません。

一般的なワークフローは次の手順で構成されます。

手順

1. クライアント証明書とキーを生成します。生成時に、 ONTAP ユーザとして認証するように Common

Name （ CN ；共通名）を設定します。

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

2. 信頼された CA 証明書を ONTAP クラスタに追加します。この処理は、ストレージ管理者がすでに行っている可能性があります。信頼できる CA が使用されていない場合は無視します。

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

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

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

4. ONTAP セキュリティログインロールでサポートされていることを確認する `cert` 認証方式。

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

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

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

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

認証方法を更新するか、クレデンシャルをローテーションして

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

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

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

パスワードのローテーションを実行する際には、ストレージ管理者が最初に ONTAP でユーザのパスワードを更新する必要があります。この後にバックエンドアップデートが続きます。証明書のローテーションを実行する際に、複数の証明書をユーザに追加することができます。その後、バックエンドが更新されて新しい証明書が使用されるようになります。この証明書に続く古い証明書は、 ONTAP クラスタから削除できます。

バックエンドを更新しても、すでに作成されているボリュームへのアクセスは中断されず、その後のボリューム接続にも影響しません。バックエンドの更新が成功した場合、 Astra 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ことを示しています。

NFS エクスポートポリシーを管理します

Astra 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。

Astra Trident には、エクスポートポリシーを使用する際に次の 2 つのオプションがあります。

- Astra Trident は、エクスポートポリシー自体を動的に管理できます。このモードでは、許容可能な 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。Astra Trident は、この範囲に含まれるノード IP をエクスポートポリシーに自動的に追加します。または、 CIDRs が指定されていない場

合は、ノード上で検出されたグローバルスコープのユニキャスト IP がエクスポートポリシーに追加されます。

- ストレージ管理者は、エクスポートポリシーを作成したり、ルールを手動で追加したりできます。構成に別のエクスポートポリシー名を指定しないと、 Astra Trident はデフォルトのエクスポートポリシーを使用します。

エクスポートポリシーを動的に管理

CSI Trident の 20.04 リリースでは、 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、この方法を使用すると、ストレージクラスタへのアクセスを指定した範囲内のIPを持つワーカーノードだけに制限できるため、きめ細かい管理が可能になります。

> エクスポートポリシーの動的管理は CSI Trident でのみ使用できます。ワーカーノードが NAT 処理されていないことを確認することが重要です。

例

2 つの設定オプションを使用する必要があります。バックエンド定義の例を次に示します。

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

> この機能を使用する場合は、SVMのルートジャンクションに、ノードのCIDRブロックを許可するエクスポートルール（デフォルトのエクスポートポリシーなど）を含む事前に作成したエクスポートポリシーがあることを確認する必要があります。ネットアップが推奨する、 Astra Trident 専用のベストプラクティスを常に守ってください。

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` がに設定されます `true`。これは、Astra Tridentがのエクスポートポリシーを作成することを示します `svm1` SVMで、を使用してルールの追加と削除を処理します `autoExportCIDRs` アドレスブロック。たとえば、UUID 403b5326-842-40dB-96d0-d83fb3f4daecのバックエンドです `autoExportPolicy` をに設定します `true` という名前のエクスポートポリシーを作成します `trident-403b5326-8482-40db-96d0-d83fb3f4daec` 指定します。
- `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は、 Astra Trident が、ワーカーノードで検出されたすべてのグローバルにスコープ指定されたユニキャストアドレスを追加します。

この例では、を使用しています `192.168.0.0/24` アドレススペースが指定されています。このアドレス範囲に含まれる Kubernetes ノードの IP が、 Astra 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。Astra Tridentは、実行されているノードを登録すると、ノードのIPアドレスを取得し、で指定されたアドレスブロックと照合してチェックします `autoExportCIDRs`。IP をフィルタリングすると、 Trident が検出したクライアント IP のエクスポートポリシールールを作成し、特定したノードごとに 1 つのルールが設定されます。

更新できます `autoExportPolicy` および `autoExportCIDRs` バックエンドを作成したあとのバックエンドの場合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。無効にすることもできます `autoExportPolicy` をバックエンドに追加し、手動で作成したエクスポートポリシーに戻します。これにはを設定する必要があります `exportPolicy` バックエンド構成のパラメータ。

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

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

Kubernetesクラスタにノードを追加してAstra Tridentコントローラに登録すると、既存のバックエンドのエクスポートポリシーが更新されます（に指定されたアドレス範囲に含まれる場合） `autoExportCIDRs` バックエンドの場合）をクリックします。

ノードを削除すると、 Astra Trident はオンラインのすべてのバックエンドをチェックして、そのノードのアクセスルールを削除します。管理対象の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、 Astra Trident は、この IP がクラスタ内の新しいノードによって再利用されないかぎり、不正なマウントを防止します。

以前のバックエンドの場合は、を使用してバックエンドを更新します `tridentctl update backend` では、Astra Tridentがエクスポートポリシーを自動的に管理します。これにより、バックエンドの UUID のあとにという名前の新しいエクスポートポリシーが作成され、バックエンドに存在するボリュームは、新しく作成したエクスポートポリシーを使用して、再びマウントします。

> 自動管理されたエクスポートポリシーを使用してバックエンドを削除すると、動的に作成されたエクスポートポリシーが削除されます。バックエンドが再作成されると、そのバックエンドは新しいバックエンドとして扱われ、新しいエクスポートポリシーが作成されます。

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

SMBボリュームをプロビジョニングする準備をします

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

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

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 2019を実行しているKubernetesクラスタ。Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- Active Directoryのクレデンシャルを含むAstra Tridentのシークレットが少なくとも1つ必要です。シークレットを生成します `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の場合は、必要に応じてSMB共有を作成するか、Astra TridentでSMB共有を作成できます。

   > 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` 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、Astra TridentでSMB共有を作成できる名前、ボリュームへ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することができます。 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 `nfs`。	`smb`	`securityStyle`
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`

ONTAP NASの設定オプションと例

Astra Tridentのインストール環境でONTAP NASドライバを作成して使用する方法をご確認ください。このセクションでは、バックエンド構成の例と、バックエンドをストレージクラスにマッピングする方法を詳しく説明します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	「 ONTAP-NAS 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-NAS-flexgroup 」、「 ONTAP-SAN 」、「 ONTAP-SAN-エコノミー 」
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + データ LIF
`managementLIF`	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス MetroClusterのシームレスなスイッチオーバーを実現するには、SVM管理LIFを指定する必要があります。 Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。 を使用してAstra Tridentをインストールした場合、IPv6アドレスを使用するようにを設定できます `--use-ipv6` フラグ。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。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」

パラメータ	説明	デフォルト
dataLIF	プロトコル LIF の IP アドレス。 を指定することを推奨します dataLIF。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。 初期設定後に変更できます。を参照してください 。 を使用してAstra Tridentをインストールした場合、IPv6アドレスを使用するようにを設定できます --use-ipv6 フラグ。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。	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）
autoExportPolicy	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]。 を使用する autoExportPolicy および autoExportCIDRs ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	いいえ
autoExportCIDRs	KubernetesのノードIPをいつからフィルタリングするかを示すCIDRsのリスト autoExportPolicy が有効になります。 を使用する autoExportPolicy および autoExportCIDRs ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	[0.0.0.0/0] 、 [::/0]
labels	ボリュームに適用する任意の JSON 形式のラベルのセット	「」
clientCertificate	クライアント証明書の Base64 エンコード値。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
`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	「」
`trustedCACertificate`	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	「」
`username`	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
`password`	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	
`svm`	使用する Storage Virtual Machine	SVMの場合に生成されます `managementLIF` を指定します
`storagePrefix`	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	Trident
`limitAggregateUsage`	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。 * Amazon FSX for ONTAP * には適用されません	“”（デフォルトでは適用されません）
`limitVolumeSize`	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）
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。 また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します `qtreesPerFlexvol` オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	""（デフォルトでは適用されません）
`lunsPerFlexvol`	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	100

パラメータ	説明	デフォルト
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例：{"API"：false 、 "method" ： true} 使用しないでください `debugTraceFlags` トラブルシューティングを実行していて、詳細なログダンプが必要な場合を除きます。	null
`nasType`	NFSボリュームまたはSMBボリュームの作成を設定 オプションはです `nfs`、 `smb` またはnull。nullに設定すると、デフォルトでNFSボリュームが使用されます。	`nfs`
`nfsMountOptions`	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は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、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。 ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的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	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、Astra TridentでSMB共有を作成できる名前、ボリュームへ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することができます。 オンプレミスのONTAPでは、このパラメータはオプションです。 このパラメータはAmazon FSx for ONTAPバックエンドで必須であり、空にすることはできません。	smb-share
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * useREST は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると true`Astra 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へのアクセス権が必要です `ontap アプリケーション：これは事前定義されたによって満たされます vsadmin および cluster-admin ロール。 useREST は、MetroCluster ではサポートされていません。	いいえ

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
spaceAllocation	space-allocation for LUN のコマンドを指定します	正しいです
spaceReserve	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
snapshotPolicy	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	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。	「」
snapshotReserve	Snapshot「0」用にリザーブされているボリュームの割合	状況 snapshotPolicy は「none」、それ以外は「」です。
splitOnClone	作成時にクローンを親からスプリットします	いいえ
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです 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。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	いいえ
tieringPolicy	「none」を使用する階層化ポリシー	ONTAP 9.5 よりも前の SVM-DR 構成の「スナップショットのみ」
unixPermissions	新しいボリュームのモード	NFSボリュームの場合は「777」、SMBボリュームの場合は空（該当なし）
snapshotDir	の表示/非表示を制御します .snapshot ディレクトリ	いいえ
exportPolicy	使用するエクスポートポリシー	デフォルト

パラメータ	説明	デフォルト
securityStyle	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： ntfs。

Trident が Astra 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。共有されない QoS ポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有 QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して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'
```

の場合 ontap-nas および ontap-nas-flexgroups`Tridentが新たに計算を使用して、FlexVol のサイズがsnapshotReserveの割合とPVCで正しく設定されていることを確認するようになりました。ユーザが

PVC を要求すると、 Astra Trident は、新しい計算を使用して、より多くのスペースを持つ元の FlexVol を作成します。この計算により、ユーザは要求された PVC 内の書き込み可能なスペースを受信し、要求されたスペースよりも少ないスペースを確保できます。v21.07 より前のバージョンでは、ユーザが PVC を要求すると（ 5GiB など）、 snapshotReserve が 50% に設定されている場合、書き込み可能なスペースは 2.5GiB のみになります。これは、ユーザが要求したボリューム全体とがであるためです `snapshotReserve には、その割合を指定します。Trident 21.07では、ユーザが要求したものが書き込み可能なスペースであり、Astra Tridentが定義します snapshotReserve ボリューム全体に対する割合として示されます。には適用されません ontap-nas-economy。この機能の仕組みについては、次の例を参照してください。

計算は次のとおりです。

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

snapshotReserve = 50% 、 PVC 要求 = 5GiB の場合、ボリュームの合計サイズは 2/0.5 = 10GiB であり、使用可能なサイズは 5GiB であり、これが PVC 要求で要求されたサイズです。。 volume show 次の例のような結果が表示されます。

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

以前のインストールからの既存のバックエンドは、 Astra Trident のアップグレード時に前述のようにボリュームをプロビジョニングします。アップグレード前に作成したボリュームについては、変更が反映されるようにボリュームのサイズを変更する必要があります。たとえば、が搭載されている2GiB PVCなどです snapshotReserve=50 以前は、書き込み可能なスペースが1GiBのボリュームが作成されていました。たとえば、ボリュームのサイズを 3GiB に変更すると、アプリケーションの書き込み可能なスペースが 6GiB のボリュームで 3GiB になります。

最小限の設定例

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

> ネットアップ ONTAP で Trident を使用している場合は、 IP アドレスではなく LIF の DNS 名を指定することを推奨します。

最小限の構成 **<code>ontap-nas-economy</code>**

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

最小限の構成 **<code>ontap-nas-flexgroup</code>**

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

SMBボリュームの最小限の設定

```
---
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

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

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 Astra Trident に指示する方法を示しています。これは、でも同様に機能します `ontap-nas-economy` および `ontap-nas-flexgroup` ドライバ。

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

IPv6アドレスを使用している

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**

。 `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_
```

仮想プールを使用するバックエンドの例

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

Astra Tridentは、[Comments]フィールドにプロビジョニングラベルを設定します。コメントは次のFlexVolに設定されています： `ontap-nas` またはFlexGroup for `ontap-nas-flexgroup`。Astra Tridentは、プロビジョニング時に仮想プール上にあるすべてのラベルをストレージボリュームにコピーします。ストレージ管理者は、仮想プールごとにラベルを定義したり、ボリュームをラベルでグループ化したりできます。

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

ONTAP NASの例

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

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

ONTAP NAS FlexGroupの例

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

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

ONTAP NASエコノミーの例

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

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

バックエンドを **StorageClasses** にマッピングします

次のStorageClass定義は、を参照してください。 [仮想プールを使用するバックエンドの例]。を使用する `parameters.selector` フィールドでは、各StorageClassがボリュームのホストに使用できる仮想プールを呼び出します。ボリュームには、選択した仮想プール内で定義された要素があります。

- 。 `protection-gold` StorageClassは、 `ontap-nas-flexgroup` バックエンド：ゴールドレベルの保護を提供する唯一のプールです。

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

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

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

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

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

- [t] `protection-silver-creditpoints-20k` StorageClassは、 `ontap-nas-flexgroup` バックエンド：シルバーレベルの保護と20000クレジットポイントを提供する唯一のプールです。

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

- 。 `creditpoints-5k` StorageClassは、 `ontap-nas` バックエンドと内の2番目の仮想プール `ontap-nas-economy` バックエンド：これらは、5000クレジットポイントを持つ唯一のプールオファリングです。

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

Tridentが、どの仮想プールを選択するかを判断し、ストレージ要件を確実に満たすようにします。

更新 `dataLIF` 初期設定後

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

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

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

NetApp ONTAP 対応の Amazon FSX

Amazon FSX for NetApp ONTAP で Astra Trident を使用

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

概要

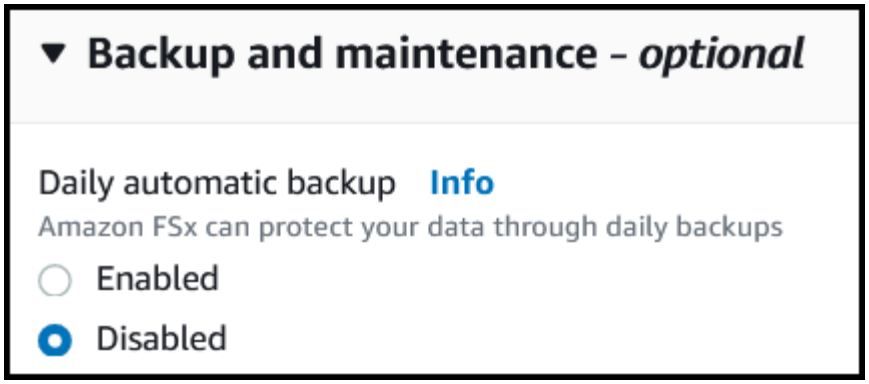
ファイルシステムは、オンプレミスの ONTAP クラスタに似た、 Amazon FSX のプライマリリソースです。各 SVM 内には、ファイルとフォルダをファイルシステムに格納するデータコンテナである 1 つ以上のボリュームを作成できます。Amazon FSX for NetApp ONTAP を使用すると、 Data ONTAP はクラウド内の管理対象ファイルシステムとして提供されます。新しいファイルシステムのタイプは * NetApp ONTAP * です。

Amazon Elastic Kubernetes Service （ EKS ）で実行されている Astra Trident と Amazon FSX for NetApp ONTAP を使用すると、 ONTAP がサポートするブロックボリュームとファイル永続ボリュームを確実にプロビジョニングできます。

NetApp ONTAP 用の Amazon FSX では、を使用します "FabricPool" ストレージ階層を管理します。データへのアクセス頻度に基づいて階層にデータを格納することができます。

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。
  - Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- 自動バックアップが有効になっているAmazon FSXファイルシステムで作成されたボリュームはTridentで削除できません。PVC を削除するには、 PV と ONTAP ボリュームの FSX を手動で削除する必要があります。この問題 を回避するには、次の手順
  - ONTAP ファイル・システム用の FSX を作成する場合は '**Quick create** を使用しないでくださいクイック作成ワークフローでは、自動バックアップが有効になり、オプトアウトオプションはありません。
  - **Standard create** を使用する場合は、自動バックアップを無効にしてください。自動バックアップを無効にすると、 Trident は手動操作なしでボリュームを正常に削除できます。



ドライバ

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

- `ontap-san`：プロビジョニングされる各PVは、NetApp ONTAP ボリューム用に独自のAmazon FSX内にあるLUNです。
- `ontap-san-economy`：プロビジョニングされる各PVは、Amazon FSXあたり、NetApp ONTAP ボリューム用に構成可能なLUN数を持つLUNです。
- `ontap-nas`：プロビジョニングされた各PVは、NetApp ONTAP ボリュームのAmazon FSX全体です。
- `ontap-nas-economy`：プロビジョニングされる各PVはqtreeで、NetApp ONTAP ボリュームのAmazon FSXごとに設定可能な数のqtreeがあります。
- `ontap-nas-flexgroup`：プロビジョニングされた各PVは、NetApp ONTAP FlexGroup ボリュームのAmazon FSX全体です。

ドライバーの詳細については、を参照してください "ONTAP ドライバ"。

認証

Astra Tridentは、2種類の認証モードを提供します。

- 証明書ベース： Astra Trident は、 SVM にインストールされている証明書を使用して、 FSX ファイルシステムの SVM と通信します。
- クレデンシャルベース：を使用できます `fsxadmin` ユーザが自身のファイルシステムまたはに割り当てられます `vsadmin` ユーザがSVM用に設定します。

> Astra Tridentは `vsadmin` SVMユーザまたは同じロールを持つ別の名前のユーザ。NetApp ONTAP 対応のAmazon FSXには、が搭載されています `fsxadmin` ONTAP を限定的に交換するユーザ `admin` クラスタユーザ：を使用することを強く推奨します `vsadmin` ネットアップが実現します。

証明書ベースの方法と証明書ベースの方法を切り替えるために、バックエンドを更新できます。ただし、*クレデンシャルと*証明書を入力しようとすると、バックエンドの作成に失敗します。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。

認証を有効にする方法の詳細については、使用しているドライバタイプの認証を参照してください。

- "ONTAP NAS認証"

- "ONTAP SAN認証"

詳細については、こちらをご覧ください

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

NetApp ONTAP 向けAmazon FSXを統合します

Amazon Elastic Kubernetes Service（EKS）で実行されているKubernetesクラスタが、ONTAP によってサポートされるブロックおよびファイルの永続ボリュームをプロビジョニングできるように、Amazon ONTAP ファイルシステム用のAmazon FSXをAstra Tridentに統合することができます。

要件

に加えて "Astra Trident の要件"FSX for ONTAP とAstra Tridentを統合するには、次のものが必要です。

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

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

- Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート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 を参照してください。

ONTAP SANとNASドライバの統合

> SMBボリュームについて設定する場合は、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 バックエンドを作成する前に。

手順

1. のいずれかを使用してAstra Tridentを導入 "導入方法"。
2. SVM管理LIFのDNS名を収集します。たとえば、AWS CLIを使用してを検索します `DNSName` の下のエントリ `Endpoints → Management` 次のコマンドを実行した後：

```
aws fsx describe-storage-virtual-machines --region <file system region>
```

3. 用の証明書を作成してインストールします "NASバックエンド認証" または "SANバックエンド認証"。

ファイルシステムにアクセスできる任意の場所から SSH を使用して、ファイルシステムにログイン（証明書をインストールする場合など）できます。を使用します `fsxadmin` user、ファイルシステムの作成時に設定したパスワード、およびの管理DNS名 `aws fsx describe-file-systems`。

4. 次の例に示すように、証明書と管理 LIF の DNS 名を使用してバックエンドファイルを作成します。

YAML

```
---
version: 1
storageDriverName: ontap-san
backendName: customBackendName
managementLIF: svm-XXXXXXXXXXXXXXXXX.fs-XXXXXXXXXXXXXXXXX.fsx.us-
east-2.aws.internal
svm: svm01
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

JSON

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "customBackendName",
  "managementLIF": "svm-XXXXXXXXXXXXXXXXX.fs-
XXXXXXXXXXXXXXXXX.fsx.us-east-2.aws.internal",
  "svm": "svm01",
  "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
  "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
  "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
 }
```

バックエンドの作成については、次のリンクを参照してください。

- "ONTAP NASドライバを使用したバックエンドの設定"
- "バックエンドに ONTAP SAN ドライバを設定します"

結果

導入後、を作成できます "ストレージクラスを定義してボリュームをプロビジョニングし、ポッドでボリュームをマウント"。

を使用してSMBボリュームをプロビジョニングできます `ontap-nas` ドライバ。をクリックしてください ONTAP SANとNASドライバの統合 次の手順を実行します。

作業を開始する前に

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

- Linuxコントローラノードと少なくとも1つのWindowsワーカーノードでWindows Server 2019を実行しているKubernetesクラスタ。Astra Tridentは、Windowsノードで実行されているポッドにマウントされたSMBボリュームのみをサポート
- Active Directoryのクレデンシャルを含むAstra Tridentのシークレットが少なくとも1つ必要です。シークレットを生成します `smbcreds`：

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

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

手順

1. SMB共有を作成SMB管理共有は、のいずれかの方法で作成できます "Microsoft管理コンソール" 共有フォルダスナップインまたはONTAP CLIを使用します。ONTAP CLIを使用してSMB共有を作成するには、次の手順を実行します
   a. 必要に応じて、共有のディレクトリパス構造を作成します。

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

   b. 指定したSVMに関連付けられているSMB共有を作成します。

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

   c. 共有が作成されたことを確認します。

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

   を参照してください "SMB 共有を作成" 詳細については、

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

（ONTAP の構成オプションと例）"。

パラメータ	説明	例
`smbShare`	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。 このパラメータは、Amazon FSx for ONTAPバックエンドに必要です。	`smb-share`
`nasType`	をに設定する必要があります **`smb`**. nullの場合、デフォルトはです `nfs`。	`smb`
`securityStyle`	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。*	""

FSX（**ONTAP** の構成オプションと例）

Amazon FSX for ONTAP のバックエンド構成オプションについて説明します。ここでは、バックエンドの設定例を示します。

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	例
`version`		常に 1
`storageDriverName`	ストレージドライバの名前	`ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`
`backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + データ LIF

パラメータ	説明	例
managementLIF	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス MetroClusterのシームレスなスイッチオーバーを実現するには、SVM管理LIFを指定する必要があります。 Fully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定できます。 を使用してAstra Tridentをインストールした場合、IPv6アドレスを使用するようにを設定できます --use-ipv6 フラグ。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。	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
dataLIF	プロトコル LIF の IP アドレス。 * ONTAP NASドライバ*：データLIFを指定することを推奨します。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。初期設定後に変更できます。を参照してください 。 * ONTAP SANドライバ*: iSCSIには指定しないでくださいTrident がONTAP の選択的LUNマップを使用して、マルチパスセッションの確立に必要なiSCSI LIFを検出します。データLIFが明示的に定義されている場合は警告が生成されます。 を使用してAstra Tridentをインストールした場合、IPv6アドレスを使用するようにを設定できます --use-ipv6 フラグ。IPv6アドレスは、[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]などの角かっこで定義する必要があります。	

パラメータ	説明	例
`autoExportPolicy`	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]。 を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	`false`
`autoExportCIDRs`	KubernetesのノードIPをいつからフィルタリングするかを示すCIDRsのリスト `autoExportPolicy` が有効になります。 を使用する `autoExportPolicy` および `autoExportCIDRs` ネットアップのAstra Tridentなら、エクスポートポリシーを自動的に管理できます。	「[0.0.0.0/0]、「::/0」」」
`labels`	ボリュームに適用する任意のJSON 形式のラベルのセット	""
`clientCertificate`	クライアント証明書の Base64 エンコード値。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	""
`trustedCACertificate`	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。	""
`username`	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`	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。	
`svm`	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。

パラメータ	説明	例
`storagePrefix`	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`
`limitAggregateUsage`	* Amazon FSx for NetApp ONTAP には指定しないでください。* 提供された `fsxadmin` および `vsadmin` アグリゲートの使用状況を取得し、Astra Tridentを使用して制限するために必要な権限が含まれていない。	使用しないでください。
`limitVolumeSize`	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。 また、qtreeおよびLUN用に管理するボリュームの最大サイズも制限します `qtreesPerFlexvol` オプションを使用すると、FlexVol あたりの最大qtree数をカスタマイズできます。	“”（デフォルトでは適用されません）
`lunsPerFlexvol`	FlexVol あたりの最大LUN数。有効な範囲は50、200です。 SANのみ。	`100`
`debugTraceFlags`	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true} 使用しないでください `debugTraceFlags` 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
nfsMountOptions	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は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、Astra Tridentはストレージバックエンドの構成ファイルで指定されているマウントオプションを使用します。 ストレージクラスや構成ファイルにマウントオプションが指定されていない場合、Astra Tridentは関連付けられた永続的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 あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	200
smbShare	次のいずれかを指定できます。Microsoft管理コンソールまたはONTAP CLIを使用して作成されたSMB共有の名前、またはAstra TridentでSMB共有を作成できるようにする名前。 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

パラメータ	説明	例
useREST	ONTAP REST API を使用するためのブーリアンパラメータ。* テクニカルプレビュー * useREST は、テクニカルプレビューとして提供されています。テスト環境では、本番環境のワークロードでは推奨されません。に設定すると true`Astra 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へのアクセス権が必要です `ontap アプリケーション：これは事前定義されたによって満たされます vsadmin および cluster-admin ロール。	false

更新 dataLIF 初期設定後

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

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

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

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
spaceAllocation	space-allocation for LUN のコマンドを指定します	true
spaceReserve	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
snapshotPolicy	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 が Astra 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。 非共有のQoSポリシーグループを使用して、各コンスティチュエントに個別にポリシーグループを適用することを推奨します。共有QoS ポリシーグループにより、すべてのワークロードの合計スループットに対して上限が適用されます。	「」
adaptiveQosPolicy	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。	「」
snapshotReserve	Snapshot「0」用にリザーブされているボリュームの割合	状況 snapshotPolicy はです none、 else 「」
splitOnClone	作成時にクローンを親からスプリットします	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです 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。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	false

パラメータ	説明	デフォルト
`luksEncryption`	LUKS暗号化を有効にします。を参照してください "Linux Unified Key Setup（LUKS；統合キーセットアップ）を使用"。 SANのみ。	""
`tieringPolicy`	使用する階層化ポリシー `none`	`snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合
`unixPermissions`	新しいボリュームのモード。 * SMBボリュームは空にしておきます。*	「」
`securityStyle`	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： `ntfs`。

例

を使用します `nasType`、 `node-stage-secret-name`および `node-stage-secret-namespace`を使用して、SMBボリュームを指定し、必要なActive Directoryクレデンシャルを指定できます。SMBボリュームは、を使用してサポートされます `ontap-nas` ドライバーのみ。

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

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、 Astra Trident とストレージシステムの関係を定義します。Trident がストレージシステムとの通信方法を Trident から指示し、 Astra Trident がボリュームをプロビジョニングする方法も解説します。Astra Trident のインストールが完了したら、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、TridentバックエンドをKubernetesインターフェイスから直接作成および管理できます。これは、を使用して実行できます `kubectl` または、Kubernetesディストリビューションと同等のCLIツールを使用します。

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`、 `tbconfig`、 `tbackendconfig`）は、Astra Tridentをバックエンドで管理できるフロントエンドで、名前を付けたCRDです `kubectl`。Kubernetesやストレージ管理者は、専用のコマンドラインユーティリティを使用せずに、Kubernetes CLIを使用してバックエンドを直接作成、管理できるようになりました (`tridentctl`）。

を作成したとき `TridentBackendConfig` オブジェクトの場合は次のようになります。

- バックエンドは、指定した構成に基づいて Astra Trident によって自動的に作成されます。これは、内部的にはとして表されます `TridentBackend` (`tbe`、 `tridentbackend`）CR。
- 。 `TridentBackendConfig` はに一意にバインドされます `TridentBackend` Astra Tridentによって作成されたのです。

各 `TridentBackendConfig` では、1対1のマッピングを保持します `TridentBackend`。前者はバックエンドの設計と構成をユーザに提供するインターフェイスで、後者は Trident が実際のバックエンドオブジェクトを表す方法です。

> `TridentBackend` CRSはAstra 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場合は、を変更して更新します `TridentBackendConfig` オブジェクト。

の形式については、次の例を参照してください `TridentBackendConfig` CR：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

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。使用するストレージドライバの

設定オプションの一覧については、を参照してください "ストレージドライバのバックエンド設定情報"。

。 spec セクションには、も含まれます credentials および deletionPolicy フィールドは、で新たに導入されました TridentBackendConfig CR：

- 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。
- deletionPolicy:このフィールドは、がどうなるかを定義します TridentBackendConfig が削除されました。次の 2 つの値のいずれかを指定できます。
  - delete:この結果、両方が削除されます TridentBackendConfig CRとそれに関連付けられたバックエンド。これがデフォルト値です。
  - retain:時 TridentBackendConfig CRが削除され、バックエンド定義は引き続き存在し、で管理できます tridentctl。削除ポリシーをに設定しています retain 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することができます。このフィールドの値は、のあとに更新できます TridentBackendConfig が作成されます。

バックエンドの名前は、を使用して設定されます spec.backendName。指定しない場合、バックエンドの名前はの名前に設定されます TridentBackendConfig オブジェクト（metadata.name）。を使用してバックエンド名を明示的に設定することを推奨します spec.backendName。

で作成されたバックエンド tridentctl が関連付けられていません TridentBackendConfig オブジェクト。このようなバックエンドの管理は、で選択できます kubectl を作成します TridentBackendConfig CR。同一の設定パラメータ（など）を指定するように注意する必要があります spec.backendName、 spec.storagePrefix、 spec.storageDriverName`など）。新しく作成したTridentがAstraに自動的にバインドされる `TridentBackendConfig 既存のバックエンドを使用します。

手順の概要

を使用して新しいバックエンドを作成します `kubectl`では、次の操作を実行する必要があります。

1. を作成します "Kubernetes Secret"。シークレットには、ストレージクラスタ / サービスと通信するために Trident から必要なクレデンシャルが含まれています。
2. を作成します TridentBackendConfig オブジェクト。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、を使用してそのステータスを確認できます kubectl get tbc <tbc-name> -n <trident-namespace> 追加の詳細情報を収集します。

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シークレットフィールドの説明	秘密	**Fields**概要
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID 。CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP
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。の場合 `ontap-san` および `ontap-san-economy`

ストレージプラットフォームシークレットフィールドの説明	秘密	**Fields**概要
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。の場合 `ontap-san` および `ontap-san-economy`
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。の場合 `ontap-san` および `ontap-san-economy`
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。の場合 `ontap-san` および `ontap-san-economy`

このステップで作成されたシークレットは、で参照されます `spec.credentials` のフィールド `TridentBackendConfig` 次のステップで作成されたオブジェクト。

手順2：を作成します `TridentBackendConfig` CR

これで、を作成する準備ができました `TridentBackendConfig` CR。この例では、を使用するバックエンド `ontap-san` ドライバは、を使用して作成されます `TridentBackendConfig` 以下のオブジェクト：

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

手順3：のステータスを確認します TridentBackendConfig CR

を作成しました TridentBackendConfig CRでは、ステータスを確認できます。次の例を参照してください。

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                       BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

バックエンドが正常に作成され、にバインドされました TridentBackendConfig CR。

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:を使用して表されます ""。。 TridentBackendConfig オブジェクトがバックエンドにバインドされていません。新しく作成されたすべてのファイル TridentBackendConfig CRSはデフォルトでこのフェーズになっています。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ボリューム要求（PVC）が存在しない場合は、を削除します TridentBackendConfig その結果、Astra Tridentによってバックエンドとが削除されます TridentBackendConfig CR。
  - 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。。 TridentBackendConfig CRはその後、削除フェーズにも入ります。バックエンドと TridentBackendConfig は、すべてのPVCが削除されたあとにのみ削除されます。
- Lost:に関連付けられているバックエンド TridentBackendConfig CRが誤って削除されたか、故意に削除された 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、に関係なく削除できます deletionPolicy 価値。
- Unknown：Astra Tridentは、に関連付けられているバックエンドの状態または存在を特定できません TridentBackendConfig CR。たとえば、APIサーバが応答していない場合や、が応答していない場合などです tridentbackends.trident.netapp.io CRDがありません。これには、ユーザの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、のYAML／JSONダンプを取得することもできます `TridentBackendConfig`。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: "2021-04-21T20:45:11Z"
  finalizers:
  - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo` が含まれます `backendName` および `backendUUID` に応答して作成されたバックエンドの `TridentBackendConfig` CR。。 `lastOperationStatus` フィールドは、の最後の操作のステータスを表します `TridentBackendConfig` CR。ユーザーがトリガすることができます（例えば、ユーザーがで何かを変更した場合など） `spec`）を使用するか、Astra Tridentによってトリガーされます（Astra Tridentの再起動時など）。Success または Failed のいずれかです。 `phase` は、間の関係のステータスを表します

TridentBackendConfig CRとバックエンド。上記の例では、 phase 値はバインドされています。これは、を意味します TridentBackendConfig CRはバックエンドに関連付けられています。

を実行できます kubectl -n trident describe tbc <tbc-cr-name> イベントログの詳細を確認するためのコマンドです。

> 関連付けられているが含まれているバックエンドは更新または削除できません TridentBackendConfig を使用するオブジェクト tridentctl。切り替えに関連する手順を理解する tridentctl および TridentBackendConfig、 "こちらを参照してください"。

kubectl を使用してバックエンド管理を実行します

を使用してバックエンド管理処理を実行する方法について説明します kubectl。

バックエンドを削除します

を削除する TridentBackendConfig`を使用して、Astra Tridentにバックエンドの削除と保持を指示します（ベースはです） `deletionPolicy）。バックエンドを削除するには、を確認します deletionPolicy は削除に設定されています。のみを削除します TridentBackendConfig`を参照してください `deletionPolicy はretainに設定されています。これにより、バックエンドがまだ存在し、を使用して管理できるようになります tridentctl。

次のコマンドを実行します。

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

Astra Tridentは、が使用していたKubernetesシークレットを削除しません 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
```

を実行することもできます tridentctl get backend -n trident または tridentctl get backend -o yaml -n trident 存在するすべてのバックエンドのリストを取得します。このリストには、で作成されたバックエンドも含まれます tridentctl。

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場合、で使用されるKubernetes Secret TridentBackendConfig オブジェクトを更新する必要があります。Astra Trident

が、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。
  この場合、 `TridentBackendConfig` オブジェクトはKubernetesを使用して直接更新できます。

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

または、既存のに変更を加えます `TridentBackendConfig` CRには次のコマンドを実行します。

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

バックエンドの更新に失敗した場合、バックエンドは最後の既知の設定のまま残ります。を実行すると、ログを表示して原因 を特定できます `kubectl get tbc <tbc-name> -o yaml -n trident` または `kubectl describe tbc <tbc-name> -n trident`。

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

tridentctl を使用してバックエンド管理を実行します

を使用してバックエンド管理処理を実行する方法について説明します `tridentctl`。

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
```

構成ファイルで問題を特定して修正したら、を実行するだけです `create` コマンドをもう一度実行します。

バックエンドを削除します

Astra 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
```

Astra Trident で、まだ存在しているこのバックエンドからボリュームとスナップショットをプロビジョニングしている場合、バックエンドを削除すると、新しいボリュームをプロビジョニングできなくなります。バックエンドは「削除」状態のままになり、 Trident は削除されるまでそれらのボリュームとスナップショットを管理し続けます。

既存のバックエンドを表示します

Trident が認識しているバックエンドを表示するには、次の手順を実行します。

- 概要を取得するには、次のコマンドを実行します。

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

- すべての詳細を確認するには、次のコマンドを実行します。

```
tridentctl get backend -o json -n trident
```

バックエンドを更新します

新しいバックエンド構成ファイルを作成したら、次のコマンドを実行します。

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

バックエンドの更新が失敗した場合、バックエンドの設定に問題があるか、無効な更新を試行しました。次のコマンドを実行すると、ログを表示して原因を特定できます。

```
tridentctl logs -n trident
```

構成ファイルで問題を特定して修正したら、を実行するだけです `update` コマンドをもう一度実行します。

バックエンドを使用するストレージクラスを特定します

以下は、回答 でできるJSON形式の質問の例です `tridentctl` バックエンドオブジェクトの出力。これにはを使用します `jq` 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}]'
```

を使用して作成されたバックエンドにも該当します `TridentBackendConfig`。

バックエンド管理オプション間を移動します

Astra Trident でバックエンドを管理するさまざまな方法をご確認ください。

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。

- を使用してバックエンドを作成可能 `tridentctl` で管理できます `TridentBackendConfig`？
- を使用してバックエンドを作成可能 `TridentBackendConfig` を使用して管理します `tridentctl`？

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、を使用して作成したバックエンドを管理するために必要な手順について説明します `tridentctl` を作成し、Kubernetesインターフェイスから直接実行 `TridentBackendConfig` オブジェクト。

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がありません `TridentBackendConfig` を使用して作成されたためです `tridentctl`。
- で作成された新しいバックエンド `tridentctl`、他の間 `TridentBackendConfig` オブジェクトが存在します。

どちらの場合も、 Trident でボリュームをスケジューリングし、処理を行っているバックエンドは引き続き存在します。管理者には次の 2 つの選択肢があります。

- の使用を続行します `tridentctl` を使用して作成されたバックエンドを管理します。
- を使用して作成したバックエンドをバインド `tridentctl` 新しい `TridentBackendConfig` オブジェクト。これにより、バックエンドはを使用して管理されます `kubectl` ではありません `tridentctl`。

を使用して、既存のバックエンドを管理します `kubectl`を作成する必要があります `TridentBackendConfig` これは既存のバックエンドにバインドします。その仕組みの概要を以下に示します。

1. Kubernetes Secret を作成します。シークレットには、ストレージクラスタ / サービスと通信するために Trident から必要なクレデンシャルが含まれています。
2. を作成します `TridentBackendConfig` オブジェクト。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。同一の設定パラメータ（など）を指定するように注意する必要があります `spec.backendName`、 `spec.storagePrefix`、 `spec.storageDriverName`など）。 `spec.backendName` 既存のバックエンドの名前に設定する必要があります。

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、を作成します `TridentBackendConfig` 既存のに自動的にバインドされるCR `ontap-nas-backend` （この例のように）。次の要件が満たされていることを確認します。

- に同じバックエンド名が定義されています `spec.backendName`。
- 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。

この場合は、を参照してください `TridentBackendConfig` 次のようになります。

```
cat backend-tbc-ontap-nas.yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'

kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

手順**3**：のステータスを確認します `TridentBackendConfig` **CR**

のあとに入力します `TridentBackendConfig` が作成されている必要があります `Bound`。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、バックエンドはを使用して完全に管理されます `tbc-ontap-nas-backend` `TridentBackendConfig` オブジェクト。

管理 `TridentBackendConfig` を使用してバックエンドを `tridentctl`

`tridentctl` を使用して、を使用して作成されたバックエンドを表示できます `TridentBackendConfig`。また、管理者は、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 `tridentctl` 削除します `TridentBackendConfig` そして確かめなさい `spec.deletionPolicy` がに設定されます `retain`。

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を使用して作成されたとします `TridentBackendConfig`：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

出力から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`

では、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ます `retain`。これにより、が確実に実行されます `TridentBackendConfig` CRが削除され、バックエンド定義は引き続き存在し、で管理できます `tridentctl`。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

それ以外の場合は、次の手順に進まないでください `deletionPolicy` がに設定されます `retain`。

手順2：を削除します `TridentBackendConfig` **CR**

最後の手順は、を削除することです `TridentBackendConfig` CR。確認が完了したら `deletionPolicy` がに設定されます `retain`をクリックすると、次のように削除されます。

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

が削除されたとき `TridentBackendConfig` Astra Tridentは、実際にバックエンド自体を削除することなく、単にオブジェクトを削除します。

ストレージクラスを管理する

ストレージクラスを作成したり、ストレージクラスを削除したり、既存のストレージクラスを表示したりできます。

ストレージクラスを設計する

を参照してください "こちらをご覧ください" ストレージクラスとその設定方法の詳細については、を参照してください。

ストレージクラスを作成する。

ストレージクラスファイルが作成されたら、次のコマンドを実行します。

```
kubectl create -f <storage-class-file>
```

`<storage-class-file>` は、ストレージクラスのファイル名に置き換えてください。

ストレージクラスを削除する

Kubernetes からストレージクラスを削除するには、次のコマンドを実行します。

```
kubectl delete storageclass <storage-class>
```

`<storage-class>` は、ストレージクラスで置き換える必要があります。

このストレージクラスで作成された永続ボリュームには変更はなく、 Astra Trident によって引き続き管理されます。

> Astra Tridentでは空白が強制される `fsType` を作成します。iSCSIバックエンドの場合は、適用することを推奨します `parameters.fsType` ストレージクラス。既存のストレージクラスを削除して、で再作成する必要があります `parameters.fsType` 指定された。

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
```

- Astra 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
```

- Astra Trident の同期されたストレージクラスの詳細を表示するには、次のコマンドを実行します。

```
tridentctl get storageclass <storage-class> -o json
```

デフォルトのストレージクラスを設定する

Kubernetes 1.6 では、デフォルトのストレージクラスを設定する機能が追加されています。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。

- アノテーションを設定してデフォルトのストレージクラスを定義します `storageclass.kubernetes.io/is-default-class` をtrueに設定してストレージクラスの定義に追加します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。

- 次のコマンドを使用して、既存のストレージクラスをデフォルトのストレージクラスとして設定できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 同様に、次のコマンドを使用して、デフォルトのストレージクラスアノテーションを削除できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、このアノテーションが含まれている Trident インストーラバンドルにも例があります。

クラスタには、常に 1 つのデフォルトストレージクラスだけを設定してください。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

以下は、回答 でできるJSON形式の質問の例です `tridentctl` Astra Tridentバックエンドオブジェクトの出力これにはを使用します `jq` ユーティリティ。先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}]'
```

ボリューム操作を実行する

CSI トポロジを使用します

Astra Trident では、を使用して、 Kubernetes クラスタ内にあるノードにボリューム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をプロビジョニングするために、 Astra Trident は CSI トポロジを使用します。

CSI トポロジ機能の詳細については、を参照してください "こちらをご覧ください"。

Kubernetes には、 2 つの固有のボリュームバインドモードがあります。

- を使用 `VolumeBindingMode` をに設定します `Immediate`トポロジを認識することなくボリュームを作成できます。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。これがデフォルトです `VolumeBindingMode` また、トポロジの制約を適用しないクラスタにも適しています。永続ボリュームは、要求側ポッドのスケジュール要件に依存せずに作成されます。
- を使用 `VolumeBindingMode` をに設定します `WaitForFirstConsumer`PVCの永続的ボリュームの作成とバインディングは、PVCを使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

。 `WaitForFirstConsumer` バインディングモードでは、トポロジラベルは必要ありません。これは CSI トポロジ機能とは無関係に使用できます。

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- を実行するKubernetesクラスタ "サポートされるKubernetesバージョン"

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

- クラスタ内のノードには、トポロジを認識するためのラベルが必要です (`topology.kubernetes.io/region` および `topology.kubernetes.io/zone`）。このラベル * は、 Astra Trident をトポロジ対応としてインストールする前に、クラスタ内のノードに存在する必要があります。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

手順 **1** ：トポロジ対応バックエンドを作成する

Astra 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ボリュームを選択的にプロビジョニングするように設計できます。各バックエンドはオプションで伝送できます `supportedTopologies` サポートする必要があるゾーンおよび領域のリストを表すブロック。ストレージクラスがそのようなバックエンドを使用する場合、ボリュームは、サポートされているリージョン / ゾーンでスケジュールされているアプリケーションから要求された場合にのみ作成されます。

バックエンド定義の例を次に示します。

YAML

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

JSON

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

`supportedTopologies` は、バックエンドごとのリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、 StorageClass で指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含む StorageClasses の場合、 Astra Trident がバックエンドにボリュームを作成します。

を定義できます `supportedTopologies` ストレージプールごとに作成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
  region: Iowa-DC
  zone: Iowa-DC-A
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
- labels:
    workload: dev
  region: Iowa-DC
  zone: Iowa-DC-B
  supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
```

この例では、を使用しています `region` および `zone` ラベルはストレージプールの場所を表します。 `topology.kubernetes.io/region` および `topology.kubernetes.io/zone` ストレージプールの使用場所を指定します。

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
```

上記のStorageClass定義で、 `volumeBindingMode` がに設定されます `WaitForFirstConsumer`。この StorageClass で要求された PVC は、ポッドで参照されるまで処理されません。および、 `allowedTopologies` 使用するゾーンとリージョンを提供します。。 `netapp-san-us-east1` StorageClassがにPVCを作成します `san-backend-us-east1` 上で定義したバックエンド。

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。

例を参照 `spec` 下記：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

このマニフェストを使用して PVC を作成すると、次のような結果になります。

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

Trident でボリュームを作成して PVC にバインドするには、ポッド内の PVC を使用します。次の例を参照してください。

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

このpodSpecにより、Kubernetesは、にあるノードにPODをスケジュールするように指示されます `us-east1` リージョンを選択し、にある任意のノードから選択します `us-east1-a` または `us-east1-b` ゾーン。

次の出力を参照してください。

```
kubectl get pods -o wide
NAME         READY    STATUS    RESTARTS    AGE    IP                NODE
NOMINATED NODE   READINESS GATES
app-pod-1    1/1      Running   0           19s    192.168.25.131    node2
<none>           <none>
kubectl get pvc -o wide
NAME       STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san    Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

バックエンドを更新して追加 `supportedTopologies`

既存のバックエンドを更新して、のリストを追加することができます `supportedTopologies` を使用します `tridentctl backend update`。これは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

- "コンテナのリソースを管理"
- "ノードセレクタ"
- "アフィニティと非アフィニティ"
- "塗料および耐性"

スナップショットを操作します

永続ボリューム（PVS）のKubernetesボリュームSnapshot（ボリュームSnapshot）を作成して、Astra Tridentボリュームのポイントインタイムコピーを保持できます。また、既存のボリュームSnapshotから、_ clone__という名前の新しいボリュームを作成することもできます。ボリュームSnapshotは、でサポートされます `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `gcp-cvs`および `azure-netapp-files` ドライバ。

作業を開始する前に

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コントローラを導入する。

> GKE環境でオンデマンドボリュームスナップショットを作成する場合は、スナップショットコントローラを作成しないでください。GKE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。

ステップ**1**： `VolumeSnapshotClass`

次の例は、ボリュームSnapshotクラスを作成します。

```
cat snap-sc.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 `driver` Astra Trident CSIドライバを指します。 `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。

詳細については、link：./trident-reference/objects.html#Kubernetes -volumesnapshotclass-objectsを参照してください[`VolumeSnapshotClass`]。

手順 **2** ：既存の **PVC** のスナップショットを作成します

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
```

この例では、という名前のPVCに対してスナップショットが作成されます `pvc1` Snapshotの名前はに設定されます `pvc1-snap`。

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

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

これでが作成されました `VolumeSnapshot` オブジェクト。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。

を識別できます `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。

```
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
.
.
```

。 `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、Snapshotを使用して新しいPVCを作成できることを示します。

手順 **3** ：ボリューム **Snapshot** から **PVC** を作成します

この例では、スナップショットを使用してPVCを作成します。

```
cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

`dataSource` に、という名前のボリュームSnapshotを使用してPVCを作成する必要があることを示します `pvc1-snap` データのソースとして。このコマンドを実行すると、 Astra Trident が Snapshot から PVC を作成するように指示します。作成された PVC は、ポッドに接続して、他の PVC と同様に使用できます。

> PVCは、と同じネームスペースに作成する必要があります `dataSource`。

Snapshotを含む**PV**を削除しています

スナップショットが関連付けられている永続ボリュームを削除すると、対応する Trident ボリュームが「削除状態」に更新されます。ボリュームSnapshotを削除してAstra Tridentボリュームを削除します。

ボリューム**Snapshot**コントローラを導入する

Kubernetesディストリビューションにスナップショットコントローラと CRDが含まれていない場合は、次のように導入できます。

手順

1. ボリュームのSnapshot作成

```
cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. スナップショットコントローラを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

必要に応じて、を開きます `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` およびを更新します `namespace` に移動します。

Snapshotを使用したボリュームデータのリカバリ

Snapshotディレクトリは、を使用してプロビジョニングされるボリュームの互換性を最大限に高めるため、デフォルトでは非表示になっています。 `ontap-nas` および `ontap-nas-economy` ドライバ。を有効にします `.snapshot` スナップショットからデータを直接リカバリするディレクトリ。

ボリュームを以前のSnapshotに記録されている状態にリストアするには、ボリュームSnapshotリストアONTAP CLIを使用します。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

Snapshotコピーをリストアすると、既存のボリューム設定が上書きされます。Snapshotコピーの作成後にボリュームデータに加えた変更は失われます。

関連リンク

- "ボリューム Snapshot"
- "ボリュームSnapshotクラス"

ボリュームを展開します

Astra Trident により、 Kubernetes ユーザは作成後にボリュームを拡張できます。ここでは、 iSCSI ボリュームと NFS ボリュームの拡張に必要な設定について説明します。

iSCSI ボリュームを展開します

CSI プロビジョニングを使用して、 iSCSI Persistent Volume （ PV ）を拡張できます。

> iSCSIボリューム拡張は、でサポートされます `ontap-san`、 `ontap-san-economy`、 `solidfire-san` ドライバとにはKubernetes 1.16以降が必要です。

概要

iSCSI PV の拡張には、次の手順が含まれます。

- StorageClass定義を編集してを設定します `allowVolumeExpansion` フィールドからに移動します `true`。
- PVC定義を編集してを更新します `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。
- サイズを変更するには、 PV をポッドに接続する必要があります。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、 Astra Trident はストレージバックエンドのボリュームを拡張し、デバイスを再スキャンし、ファイルシステムのサイズを変更します。
  - 未接続の PV のサイズを変更しようとすると、 Astra Trident がストレージバックエンドのボリュームを拡張します。PVC がポッドにバインドされると、 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。

次の例は、 iSCSI PVS の仕組みを示しています。

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

```
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
```

既存のストレージクラスの場合は、編集してを追加します `allowVolumeExpansion` パラメータ

手順 **2** ：作成した **StorageClass** を使用して **PVC** を作成します

```
cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Astra Trident が、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

この例では、を使用するポッドが作成されます `san-pvc`。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

ステップ **4** ： **PV** を展開します

1Giから2Giに作成されたPVのサイズを変更するには、PVCの定義を編集してを更新します `spec.resources.requests.storage` 2Giへ。

```
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 、 Astra Trident のボリュームのサイズを確認することで、拡張が正しく機能しているかどうかを検証できます。

```
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

Astra Tridentは、でプロビジョニングしたNFS PVSのボリューム拡張をサポートしています `ontap-nas`、`ontap-nas-economy`、 `ontap-nas-flexgroup`、 `gcp-cvs`および `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、管理者はまず、を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 `allowVolumeExpansion` フィールドからに移動します `true`：

```
cat storageclass-ontapnas.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

このオプションを指定せずにストレージクラスを作成済みの場合は、を使用して既存のストレージクラスを編集するだけです `kubectl edit storageclass` ボリュームを拡張できるようにするため。

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
```

Astra Trident が、この PVC に対して 20MiB の NFS PV を作成する必要があります。

```
kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

ステップ**3**：**PV**を拡張する

新しく作成した20MiBのPVのサイズを1GiBに変更するには、そのPVCを編集してを設定します `spec.resources.requests.storage` 1 GBに設定する場合：

```
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 、 Astra Trident の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ているかどうかを検証できます。

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+---------+---------+
```

ボリュームをインポート

`tridentctl import`を使用するか、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

Astra Tridentにボリュームをインポートすると、次のことが可能になります。

- アプリケーションをコンテナ化し、既存のデータセットを再利用する
- 一時的なアプリケーションに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。

- Astra 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Astra Tridentにインポートする前に、ミラー関係を解除する必要があります。
- アクティブな接続がないボリュームをインポートすることを推奨します。アクティブに使用されているボリュームをインポートするには、ボリュームのクローンを作成してからインポートを実行します。

Kubernetesは以前の接続を認識せず、アクティブなボリュームをポッドに簡単に接続できるため、これはブロックボリュームで特に重要です。その結果、データが破損する可能性があります。

- でもね `StorageClass` PVCに対して指定する必要があります。Astra Tridentはインポート時にこのパラメータを使用しません。ストレージクラスは、ボリュームの作成時に、ストレージ特性に基づいて使用可能なプールから選択するために使用されます。ボリュームはすでに存在するため、インポート時にプールを選択する必要はありません。そのため、PVCで指定されたストレージクラスと一致しないバックエンドまたはプールにボリュームが存在してもインポートは失敗しません。
- 既存のボリュームサイズはPVCで決定され、設定されます。ストレージドライバによってボリュームがインポートされると、 PV は ClaimRef を使用して PVC に作成されます。
  - 再利用ポリシーは、最初にににに設定されています `retain` PVにあります。Kubernetes が PVC と PV を正常にバインドすると、再利用ポリシーがストレージクラスの再利用ポリシーに合わせて更新されます。
  - ストレージクラスの再利用ポリシーがの場合 `delete`にすると、PVが削除されるとストレージボリュームが削除されます。
- デフォルトでは、Astra TridentがPVCを管理し、バックエンドでFlexVolとLUNの名前を変更します。を渡すことができます `--no-manage` 管理対象外のボリュームをインポートするフラグ。を使用する場合 `--no-manage`Astra Tridentは、オブジェクトのライフサイクルを通じてPVCやPVに対して追加の処理を実行することはありません。PVが削除されてもストレージボリュームは削除されず、ボリュームのクローンやボリュームのサイズ変更などのその他の処理も無視されます。

このオプションは、コンテナ化されたワークロードに Kubernetes を使用するが、Kubernetes 以外でストレージボリュームのライフサイクルを管理する場合に便利です。

- PVC と PV にアノテーションが追加されます。このアノテーションは、ボリュームがインポートされたこと、および PVC と PV が管理されていることを示す二重の目的を果たします。このアノテーションは変更または削除しないでください。

ボリュームをインポートします

`tridentctl import`を使用するか、`Trident`インポートアノテーションを使用して`PVC`を作成することで、ボリュームをインポートできます。

PVC アノテーションを使用する場合は、 `tridentctl`をダウンロードしたり使用してボリュームをインポートしたりする必要はありません。

tridentctlの使用

手順

1. PVCを作成するために使用するPVCファイル（例： `pvc.yaml`）を作成します。PVCファイルには `name`、 `namespace`、 `accessModes`、および `storageClassName`を含める必要があります。必要に応じて、PVC定義で `unixPermissions`を指定することもできます。

   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
   ```

   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. を使用します `tridentctl import` コマンドを使用して、ボリュームを含むAstra Tridentバックエンドの名前と、ストレージ上のボリュームを一意に識別する名前（ONTAP FlexVol、Elementボリューム、Cloud Volumes Serviceパスなど）を指定します。。 `-f` PVCファイルへのパスを指定するには、引数が必要です。

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

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

Astra Tridentでは、を使用したボリュームインポートがサポートされます `ontap-nas` および `ontap-nas-flexgroup` ドライバ。

- 。 `ontap-nas-economy` ドライバでqtreeをインポートおよび管理できない。
- 。 `ontap-nas` および `ontap-nas-flexgroup` ドライバでボリューム名の重複が許可されていません。

を使用して作成した各ボリューム `ontap-nas` driverはONTAP クラスタ上のFlexVol です。を使用してFlexVolをインポートする `ontap-nas` ドライバも同じように動作します。ONTAP クラスタにすでに存在するFlexVolは、としてインポートできます `ontap-nas` PVC。同様に、FlexGroup ボリュームはとしてインポートできます `ontap-nas-flexgroup` PVC

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

を使用する場合 `--no-manage` 引数に指定します。Astra 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>
```

ONTAP SAN

Astra Tridentでは、を使用したボリュームインポートがサポートされます `ontap-san` ドライバ。

Astra Tridentでは、単一のLUNを含むONTAP SAN FlexVolをインポートできます。これはと同じです `ontap-san` ドライバ。FlexVol 内の各PVCおよびLUNにFlexVol を作成します。Astra TridentがFlexVolをインポートし、PVCの定義に関連付けます。

次の例は、管理対象ボリュームと管理対象外ボリュームをインポートする方法を示しています：

管理対象ボリューム

管理対象ボリュームの場合、Astra TridentはFlexVolの名前をに変更します `pvc-<uuid>` およびFlexVol内のLUNをからにフォーマットします `lun0`。

次の例は、をインポートします `ontap-san-managed` にあるFlexVol `ontap_san_default` バックエンド：

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

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

管理対象外のボリューム

次に、をインポートする例を示します `unmanaged_example_volume` をクリックします `ontap_san` バックエンド：

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

次の例に示すように、KubernetesノードのIQNとIQNを共有するigroupにLUNをマッピングすると、エラーが表示されます。 `LUN already mapped to initiator(s) in this group`。ボリュームをインポートするには、イニシエータを削除するか、LUNのマッピングを解除する必要があります。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

要素（ **Element** ）

Astra Tridentでは、を使用したNetApp ElementソフトウェアとNetApp HCIボリュームのインポートがサポートされます `solidfire-san` ドライバ。

> 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はAstra Tridentからエラーが返されます。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    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud Platform の **1** つです

Astra Tridentでは、を使用したボリュームインポートがサポートされます `gcp-cvs` ドライバ。

> NetApp Cloud Volumes Serviceから作成されたボリュームをGoogle Cloud Platformにインポートするには、ボリュームパスでボリュームを特定します。ボリュームパスは、ボリュームのエクスポートパスののに続く部分です `:/`。たとえば、エクスポートパスがの場合などです `10.0.0.1:/adroit-jolly-swift`、ボリュームのパスはです `adroit-jolly-swift`。

Google Cloud Platformの例

次に、をインポートする例を示します `gcp-cvs` バックエンドのボリューム `gcpcvs_YEppr` を指定します `adroit-jolly-swift`。

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

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

Azure NetApp Files の特長

Astra Tridentでは、を使用したボリュームインポートがサポートされます `azure-netapp-files` および `azure-netapp-files-subvolume` ドライバ。

> Azure NetApp Filesボリュームをインポートするには、ボリュームパスでボリュームを特定します。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    |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
```

ネームスペース間でNFSボリュームを共有します

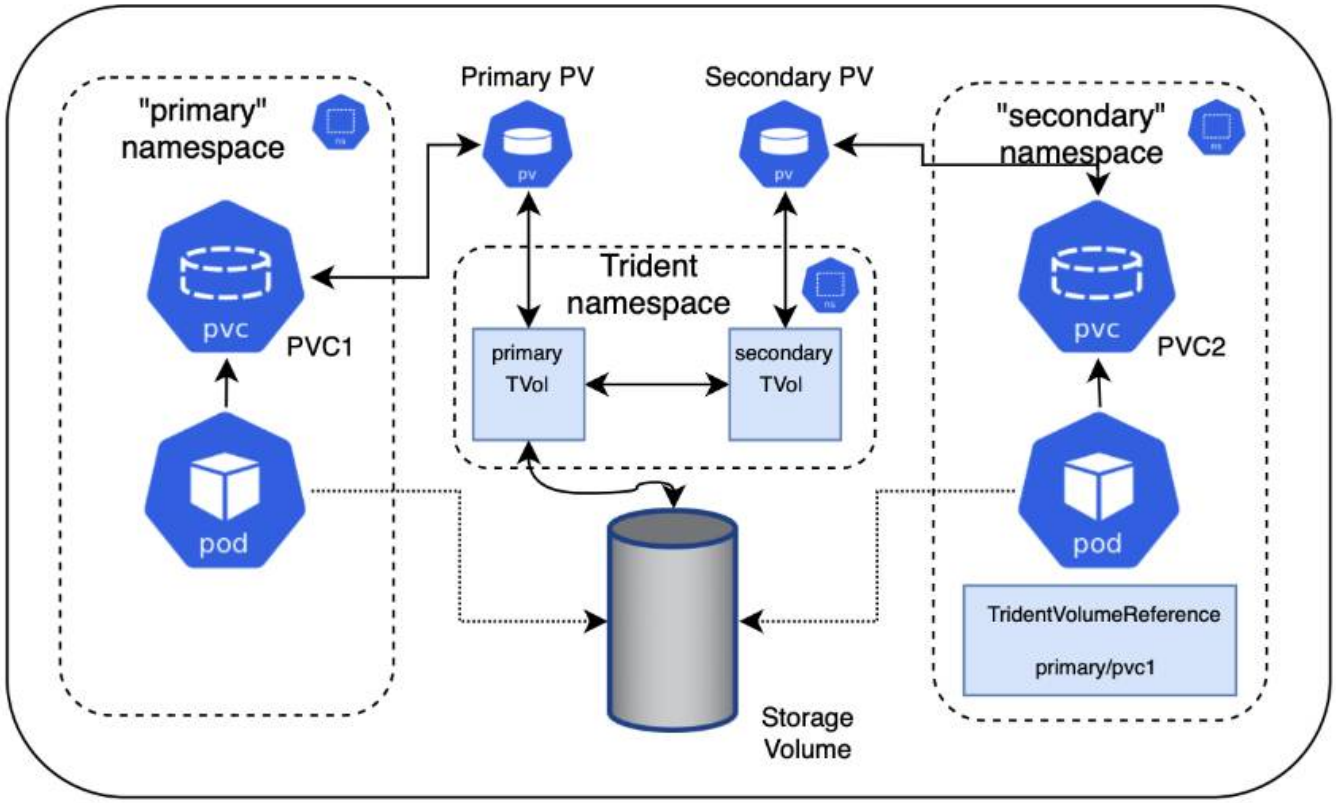
Tridentを使用すると、プライマリネームスペースにボリュームを作成し、1つ以上のセカンダリネームスペースで共有できます。

機能

Astra 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ボリュームをセキュアに共有できます。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
```

Astra TridentがPVとバックエンドのNFSストレージボリュームを作成

- カンマ区切りリストを使用して、複数の名前空間にPVCを共有できます。例： `trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- を使用して、すべてのネームスペースに共有できます `*`。例： `trident.netapp.io/shareToNamespace: *`
- PVCを更新してを含めることができます `shareToNamespace` アノテーションはいつでも作成できます。

2. *クラスタ管理者：*カスタムロールとkubeconfigを作成して、デスティネーションネームスペースの所有者にTridentVolumeReference CRを作成する権限を付与します。
3. *デスティネーションネームスペース所有者：*ソースネームスペースを参照するデスティネーションネームスペースにTridentVolumeReference CRを作成します `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 宛先名前空間の所有者： PVCを作成します (`pvc2`）をデスティネーションネームスペースに展開します(`namespace2`)を使用します `shareFromPVC` 送信元PVCを指定する注釈。

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

宛先PVCのサイズは、送信元PVCのサイズ以下である必要があります。

結果

Astra Tridentがを読み取り `shareFromPVC` デスティネーションPVCにアノテーションを設定し、ソースPVを参照するストレージリソースを持たない下位のボリュームとしてデスティネーションPVを作成し、ソースPVストレージリソースを共有します。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が、ソースネームスペースのボリュームへのアクセスを削除し、ボリュームを共有する他のネームスペースへのアクセスを維持します。ボリュームを参照するすべてのネームスペースが削除されると、Astra Tridentによってボリュームが削除されます。

使用 `tridentctl get` 下位のボリュームを照会する

を使用する[`tridentctl` ユーティリティを使用すると、を実行できます `get` コマンドを使用して下位のボリュームを取得します。詳細については、リンク:./trident-reference/tridentctl.htmlを参照してください [`tridentctl` コマンドとオプション]。

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

フラグ：

- `-h, --help`：ボリュームのヘルプ。
- `--parentOfSubordinate string`：クエリを下位のソースボリュームに制限します。
- `--subordinateOf string`:クエリをボリュームの下位に制限します。

制限

- Astra 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るのを防ぐことはできません。共有ボリュームのデータの上書きを防止するには、ファイルロックなどのプロセスを使用する必要があります。
- を削除しても、送信元PVCへのアクセスを取り消すことはできません `shareToNamespace` または `shareFromNamespace` 注釈またはを削除します `TridentVolumeReference` CR。アクセスを取り消すには、下位PVCを削除する必要があります。
- Snapshot、クローン、およびミラーリングは下位のボリュームでは実行できません。

を参照してください。

ネームスペース間のボリュームアクセスの詳細については、次の資料を参照してください。

- にアクセスします "ネームスペース間でのボリュームの共有：ネームスペース間のボリュームアクセスを許可する場合は「Hello」と入力します"。
- のデモをご覧ください "ネットアップTV"。

Astra Trident を監視

Astra Tridentは、Astra Tridentのパフォーマンス監視に使用できるPrometheus指標エン

ドポイントのセットを提供します。

概要

Astra Trident が提供する指標を使用すると、次のことが可能になります。

- Astra Trident の健常性と設定を保持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。Astra Trident がバックエンドと通信して処理を実行するのにどれくらいの時間がかかるかを調べることができます。

デフォルトでは、Tridentの指標はターゲットポートで公開されています `8001` で `/metrics` エンドポイント。これらの指標は、 Trident のインストール時にデフォルトで * 有効になります。

必要なもの

- Astra Trident がインストールされた Kubernetes クラスタ
- Prometheus インスタンス。これは a である場合もある "コンテナ化された Prometheus 環境" または、Prometheus をとして実行することもできます "ネイティブアプリケーション"。

手順 1 ： Prometheus ターゲットを定義する

Prometheus ターゲットを定義して指標を収集し、 Astra Trident が管理するバックエンド、作成するボリュームなどの情報を取得する必要があります。これ "ブログ" Prometheus と Grafana を Astra Trident とともに使用して指標を取得する方法について説明します。ブログでは、 Kubernetes クラスタでオペレータとして Prometheus を実行する方法と、 Astra Trident のメトリックを取得する ServiceMonitor を作成する方法について説明しています。

手順 2 ： Prometheus ServiceMonitor を作成します

Tridentの指標を利用するには、を監視するPrometheus ServiceMonitorを作成する必要があります `trident-csi` サービスおよびリッスン `metrics` ポート：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、から返されたメトリックを取得します `trident-csi` サービスとは、を特に探します `metrics` サービスのエンドポイント。その結果、PrometheusはAstra Tridentの
指標：

Astra Tridentから直接取得できる指標に加えて、kubeletは多くの指標を公開しています `kubelet_volume_*` 独自の指標エンドポイントを使用した指標。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "こちらをご覧ください"。

ステップ 3 ： PrompQL を使用して Trident 指標を照会する

PrompQL は、時系列データまたは表データを返す式を作成するのに適しています。

次に、 PrompQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **Astra 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
```

- **Astra Trident** からのステータスコードによる **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
```

- **Astra Trident** によって実行された処理の平均時間（ミリ秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Astra Trident の使用状況に関する情報を入手

- 平均体積サイズ

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各バックエンドによってプロビジョニングされた合計ボリューム容量

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

個々のボリュームの使用状況を取得する

これは、 kubelet 指標も収集された場合にのみ有効になります。

- 各ボリュームの使用済みスペースの割合

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Astra Trident AutoSupport の計測データ

デフォルトでは、 Astra Trident は Prometheus 指標と基本バックエンド情報を毎日定期的にネットアップに送信します。

- Astra TridentからPrometheus指標や基本バックエンド情報がネットアップに送信されないようにするには、を渡します `--silence-autosupport` Astra Tridentのインストール中にフラグを付ける。
- Tridentからネットアップサポートにコンテナログをオンデマンドで送信することもできます `tridentctl send autosupport`。Astra Trident をトリガーしてログをアップロードする必要があります。ログを送信する前に、ネットアップのに同意する必要があります "プライバシーポリシー"。
- 指定しないと、 Astra Trident は過去 24 時間からログを取得します。
- ログの保持期間は、で指定できます `--since` フラグ。例： `tridentctl send autosupport --since=1h`。この情報は、を介して収集および送信されます `trident-autosupport` コンテナこれはAstra Tridentと一緒にインストールされます。コンテナイメージは、で取得できます "Trident AutoSupport の略"。

- Trident AutoSupport は、個人情報（ PII ）や個人情報を収集または送信しません。Tridentコンテナイメージ自体には適用されないが付属して "EULA" います。データのセキュリティと信頼に対するネットアップの取り組みについて詳しくは、こちらをご覧 "こちらをご覧ください"ください。

Astra Trident から送信されるペイロードの例を次に示します。

```
---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true
```

- AutoSupport メッセージは、ネットアップの AutoSupport エンドポイントに送信されます。コンテナイメージの格納にプライベートレジストリを使用している場合は、を使用できます `--image-registry` フラグ。
- インストール YAML ファイルを生成してプロキシ URL を設定することもできます。これは、を使用して実行できます `tridentctl install --generate-custom-yaml` YAMLファイルを作成し、を追加します `--proxy-url` の引数 `trident-autosupport` にコンテナがあります `trident-deployment.yaml`。

Astra Trident の指標を無効化

**メトリックがレポートされないようにするには、を使用してカスタムYAMLを生成する必要があります `--generate-custom-yaml` フラグを付けて編集し、を削除します `--metrics` に対する呼び出し元からのフラグ `trident-main`
コンテナ：

Trident for Docker が必要です

導入の前提条件

Trident を導入するには、必要なプロトコルをホストにインストールして設定しておく必要があります。

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され、設定されていることを確認します。

プロトコル	オペレーティングシステム	コマンド
NFS	RHEL 8以降	`sudo yum install -y nfs-utils`
NFS	Ubuntu	`sudo apt-get install -y nfs-common`

プロトコル	オペレーティングシステム	コマンド

iSCSI	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`
		
		確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 `defaults`。
		
		5. を確認します `iscsid` および `multipathd` 実行中：
		
		`sudo systemctl enable --now iscsid multipathd`
		
		6. を有効にして開始します `iscsi`：
		
		`sudo systemctl enable --now iscsi`

プロトコル	オペレーティングシステム	コマンド

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`
		
		確認します `etc/multipath.conf` が含まれます `find_multipaths no` の下 `defaults`。
		
		5. を確認します `open-iscsi` および `multipath-tools` 有効になっていて実行中：
		
		`sudo systemctl status multipath-tools`

Astra Trident を導入

```
sudo systemctl enable
iscsi.service
sudo systemctl status
open-iscsi
```

Astra Trident for Docker は、ネットアップのストレージプラットフォーム向けの Docker エコシステムとの直接統合を実現します。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。

Astra Trident の複数のインスタンスを同じホストで同時に実行できます。これにより、複数のストレージシステム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、 Astra Trident を導入する準備ができました。

Docker Managed Plugin メソッド（バージョン 1.13 / 17.03 以降）

作業を開始する前に

従来のデーモン方式で Astra Trident 以前の Docker 1.13 / 17.03 を使用していた場合は、マネージドプラグイン方式を使用する前に Astra Trident プロセスを停止し、 Docker デーモンを再起動してください。

1. 実行中のインスタンスをすべて停止します。

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. Docker を再起動します。

```
systemctl restart docker
```

3. Docker Engine 17.03 （新しい 1.13 ）以降がインストールされていることを確認します。

```
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です `config.json`ただし、を指定すると、選択した任意の名前を使用できます `config` オプションを指定してファイル名を指定します構成ファイルはに格納されている必要があります `/etc/netappdvp` ホストシステム上のディレクトリ。

- `log-level`:ログレベルを指定します (`debug`、 `info`、 `warn`、 `error`、 `fatal`）。デフォルトはです `info`。
- `debug`:デバッグロギン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. マネージドプラグインシステムを使用して Astra Trident を起動交換してください `<version>` 使用しているプラグインのバージョン（xxx.xxx.xxx.xxx）を使用している必要があります。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. Astra Trident を使用して、構成したシステムのストレージを使用しましょう。

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/v23.04.0/trident-
      installer-23.04.0.tar.gz
      tar zxf trident-installer-23.04.0.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、必要な構成ファイルを使用して Trident デーモンを開始します。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

> 指定しないかぎり、ボリュームドライバのデフォルト名は「 netapp 」です。

デーモンが開始されたら、 Docker CLI インターフェイスを使用してボリュームを作成および管理できます

3. ボリュームを作成します

```
docker volume create -d netapp --name trident_1
```

4. コンテナの開始時に Docker ボリュームをプロビジョニング：

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
alpine ash
```

5. Docker ボリュームを削除します。

```
docker volume rm trident_1
docker volume rm trident_2
```

システム起動時に **Astra Trident** を起動

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、ファイルの名前によって異なります `/usr/lib/systemd/system` ディレクトリ。

   ```
   systemctl enable trident
   ```

5. サービスを開始します。

   ```
   systemctl start trident
   ```

6. ステータスを確認します。

   ```
   systemctl status trident
   ```

> 単位ファイルを変更する場合は、を実行します `systemctl daemon-reload` 変更を認識するためのコマンド。

Astra 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、 Astra 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、が短時間実行されます `docker volume` プラグインで指示されたコマンドは正常に実行されず、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

Astra 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
```

> nDVP に代わる、 Astra Trident の 18.01 リリース。から直接アップグレードする必要があります `netapp/ndvp-plugin` への画像 `netapp/trident-plugin` イメージ（Image）：

4. プラグインを有効にします。

```
docker plugin enable netapp:latest
```

5. プラグインが有効になっていることを確認します。

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. ボリュームが表示されることを確認します。

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

> 古いバージョンの Astra Trident （ 20.10 より前）から Astra Trident 20.10 以降にアップグレードすると、エラーが発生する場合があります。詳細については、を参照してください "既知の問題"。この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、次に追加のconfigパラメータを渡して、必要なAstra Tridentバージョンを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`

をアンインストールします

Astra 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

標準のを使用すると、ボリュームを簡単に作成、クローニング、および削除できます `docker volume` 必要に応じてAstra Tridentドライバ名を指定したコマンド。

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 Astra Trident インスタンスを使用してボリュームを作成します。

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

何も指定しない場合 "オプション（ Options ）"、ドライバのデフォルトが使用されます。

- デフォルトのボリュームサイズを上書きします。次の例を参照して、ドライバで 20GiB ボリュームを作成してください。

```
docker volume create -d netapp --name my_vol --opt size=20G
```

ボリュームサイズは、オプションの単位（ 10G 、 20GB 、 3TiB など）を含む整数値で指定します。単位を指定しない場合、デフォルトは g ですサイズの単位は、 2 の累乗（ B 、 KiB 、 MiB 、 GiB 、 TiB ）または 10 の累乗（ B 、 KB 、 MB 、 GB 、 TB ）のいずれかです。略記単位では、 2 の累乗が使用されます（ G=GiB 、 T=TiB 、…）。

ボリュームを削除します

- 他の Docker ボリュームと同様にボリュームを削除します。

```
docker volume rm firstVolume
```

を使用する場合 `solidfire-san` driver、上記の例では、ボリュームを削除およびパージします。

Astra 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

を使用する場合 `ontap-nas`、 `ontap-san`、 `solidfire-san`および `gcp-cvs storage drivers`Tridentがボリュームをクローニングできます。を使用する場合 `ontap-nas-flexgroup` または `ontap-nas-economy` ドライバ、クローニングはサポートされていません。既存のボリュームから新しいボリュームを作成すると、新しい Snapshot が作成されます。

- ボリュームを調べて Snapshot を列挙します。

```
docker volume inspect <volume_name>
```

- 既存のボリュームから新しいボリュームを作成します。その結果、新しい Snapshot が作成されます。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume>
```

- ボリューム上の既存の Snapshot から新しいボリュームを作成します。新しい Snapshot は作成されません。

```
docker volume create -d <driver_name> --name <new_name> -o
from=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

例

```
docker volume inspect firstVolume

[
    {
        "Driver": "ontap-nas",
        "Labels": null,
        "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
        "Name": "firstVolume",
        "Options": {},
        "Scope": "global",
        "Status": {
            "Snapshots": [
                {
                    "Created": "2017-02-10T19:05:00Z",
                    "Name": "hourly.2017-02-10_1505"
                }
            ]
        }
    }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

Tridentを使用すると、外部で作成されたブロックデバイス（またはそのクローン）にTrident *からアクセスできます。Tridentは、パーティションがなく、Astra Tridentでサポートされているファイルシステム（など）の場合にのみ利用できます `ext4`-フォーマット済み `/dev/sdc1` Astra Trident経由ではアクセスできません）。

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。を使用して、オプションと値を指定します `-o` CLI処理中の演算子。これらは、 JSON 構成ファイルの同等の値よりも優先されます。

ONTAP ボリュームのオプション

NFS と iSCSI のどちらの場合も、 volume create オプションには次のオプションがあります。

オプション	説明
`size`	ボリュームのサイズ。デフォルトは 1GiB です。
`spaceReserve`	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値はです `none` （シンプロビジョニング）および `volume` （シックプロビジョニング）。
`snapshotPolicy`	Snapshot ポリシーが目的の値に設定されます。デフォルトはです `none`つまり、ボリュームに対してSnapshotが自動的に作成されることはありません。ストレージ管理者によって変更されていない限り、「 `default` 」という名前のポリシーがすべての `ONTAP` システムに存在し、 `6` 個の時間単位 `Snapshot` 、 `2` 個の日単位 `Snapshot` 、および `2` 個の週単位 `Snapshot` を作成して保持します。Snapshotに保存されているデータは、にアクセスしてリカバリできます `.snapshot` ボリューム内の任意のディレクトリ内のディレクトリ。

オプション	説明
`snapshotReserve`	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。
`splitOnClone`	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すると、時間を大幅に節約できますが、ストレージの節約はほとんどできないため、クローンをすぐに分割することをお勧めします。
`encryption`	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです `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。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームが NAE に有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。
`tieringPolicy`	ボリュームに使用する階層化ポリシーを設定します。これにより、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、権限はに設定されます `---rwxr-xr-x`または数値表記`0755`、およびです `root` が所有者になります。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
snapshotDir	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。公式のMySQLイメージなどの一部のイメージは、の場合、期待どおりに機能しません .snapshot ディレクトリが表示されます。
exportPolicy	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。デフォルトはです default。
securityStyle	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。デフォルトはです unix。有効な値はです unix および mixed。

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	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。デフォルトはです ext4。有効な値はです ext3、 ext4`および `xfs。
spaceAllocation	これをに設定します false LUNのスペース割り当て機能をオフにします。デフォルト値はです `true`つまり、ボリュームのスペースが不足し、ボリューム内のLUNに書き込みを受け付けられなくなったときに、ONTAP からホストに通知されます。また、このオプションで ONTAP 、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 10GiB ボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- Snapshot を使用して 100GiB のボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- setuid ビットが有効になっているボリュームを作成します。

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

最小ボリュームサイズは 20MiB です。

Snapshotリザーブが指定されていない場合、Snapshotポリシーはです `none`Tridentは0%のSnapshotリザーブを使用します。

- Snapshot ポリシーがなく、 Snapshot リザーブがないボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- Snapshot ポリシーがなく、カスタムの Snapshot リザーブが 10% のボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- Snapshot ポリシーを使用し、カスタムの Snapshot リザーブを 10% に設定してボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- Snapshot ポリシーを設定してボリュームを作成し、 ONTAP のデフォルトの Snapshot リザーブ（通常は 5% ）を受け入れます。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element ソフトウェアのボリュームオプション

Element ソフトウェアのオプションでは、ボリュームに関連付けられているサービス品質（ QoS ）ポリシーのサイズと QoS を指定できます。ボリュームの作成時に、関連付けられているQoSポリシーをを使用して指定します `-o type=service_level` 名称。

Element ドライバを使用して QoS サービスレベルを定義する最初の手順は、少なくとも 1 つのタイプを作成し、構成ファイル内の名前に関連付けられた最小 IOPS 、最大 IOPS 、バースト IOPS を指定することです。

Element ソフトウェアのその他のボリューム作成オプションは次のとおりです。

オプション	説明
`size`	ボリュームのサイズ。デフォルト値は 1GiB または設定エントリ … 「 defaults 」： ｛「 size 」：「 5G 」｝ 。
`blocksize`	512 または 4096 のいずれかを使用します。デフォルトは 512 または config エントリ DefaultBlockSize です。

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

```
{
    "...": "..."
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

上記の構成では、 Bronze 、 Silver 、 Gold の 3 つのポリシー定義を使用します。これらの名前は任意です。

- 10GiB の Gold ボリュームを作成します。

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 100GiB Bronze ボリュームを作成します。

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

ログを収集します

トラブルシューティングに役立つログを収集できます。ログの収集方法は、 Docker プラグインの実行方法によって異なります。

トラブルシューティング用にログを収集する

手順

1. 推奨される管理プラグイン方法（を使用）でAstra Tridentを実行している場合 `docker plugin` コマンド）で表示される情報は次のとおりです。

```
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イネーブルにできます。

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. ホスト上でバイナリ自体を実行している場合、ログはホストので使用できます `/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。

  ```
  Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
  connect: no such file or directory
  ```

  これは、プラグインの起動に失敗したことを意味します。幸い、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。

- PVをコンテナにマウントする際に問題が発生する場合は、を確認してください `rpcbind` をインストールして実行しておきます。ホストOSに必要なパッケージマネージャを使用して、かどうかを確認します `rpcbind` を実行しています。rpcbindサービスのステータスを確認するには、を実行します `systemctl status rpcbind` またはそれと同等のものです。

複数の Astra Trident インスタンスを管理

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を作成するには、を使用して異なる名前を付けます `--alias` オプションにコンテナ化プラグインを指定するか、を指定します `--volume-driver` ホストでTridentをインスタンス化する際のオプション。

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

Astra Trident 構成で使用できる設定オプションを確認してください。

グローバル構成オプション

以下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 Astra Trident 構成に適用されます。

オプション	説明	例
version	構成ファイルのバージョン番号	1
storageDriverName	ストレージドライバの名前	ontap-nas、 ontap-san、ontap-nas-economy、ontap-nas-flexgroup、solidfire-san
storagePrefix	ボリューム名のオプションのプレフィックス。デフォルト：netappdvp_。	staging_
limitVolumeSize	ボリュームサイズに関するオプションの制限。デフォルト：「」（適用されていない）	10g

使用しないでください storagePrefix （デフォルトを含む）をElementバックエンドに使用します。デフォルトでは、が表示されます solidfire-san ドライバはこの設定を無視し、プレフィックスを使用しません。Docker ボリュームマッピングには特定の tenantID を使用するか、 Docker バージョン、ドライバ情報、名前の munging が使用されている可能性がある場合には Docker から取得した属性データを使用することを推奨します。

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。。 size オプションはすべてのコントローラタイプで使用できます。デフォルトのボリュームサイズの設定方法の例については、 ONTAP の設定に関するセクションを参照してください。

オプション	説明	例
size	新しいボリュームのオプションのデフォルトサイズ。デフォルト：1G	10G

ONTAP構成

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
managementLIF	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
dataLIF	プロトコル LIF の IP アドレス。 • ONTAP NASドライバ*:を指定することをお勧めします dataLIF。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。 • ONTAP SANドライバ*: iSCSIには指定しないでください Astra Tridentが使用 "ONTAP の選択的LUNマップ" iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます dataLIF は明示的に定義されます。	10.0.0.2
svm	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	svm_nfs
username	ストレージデバイスに接続するユーザ名	vsadmin
password	ストレージ・デバイスに接続するためのパスワード	secret
aggregate	プロビジョニング用のアグリゲート（オプション。設定する場合は SVM に割り当てる必要があります）。をクリックします ontap-nas-flexgroup ドライバ。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
limitAggregateUsage	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
nfsMountOptions	NFS マウントオプションのきめ細かな制御。デフォルトは「 -o nfsvers=3 」です。でのみ使用できます **ontap-nas** および **ontap-nas-economy** ドライバ。 "ここでは、 NFS ホストの設定情報を参照してください"。	-o nfsvers=4
igroupName	Astra Tridentはノード単位で作成、管理します igroups として netappdvp。 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	最大要求可能ボリュームサイズと qtree 親ボリュームサイズ。*のため ontap-nas-economy また、このオプションを使用すると、作成するFlexVol *のサイズも制限されます。	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

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
spaceReserve	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
snapshotPolicy	使用するSnapshotポリシー。デフォルトはです none	none

オプション	説明	例
snapshotReserve	Snapshotリザーブの割合。デフォルトはONTAP のデフォルトをそのまま使用する場合はです	10
splitOnClone	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	false
encryption	新しいボリュームでNetApp Volume Encryption（NVE）を有効にします。デフォルトはです 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。 NAEがバックエンドで有効になっている場合は、Astra Tridentでプロビジョニングされたすべてのボリュームがに有効になります。 詳細については、以下を参照してください。 "Astra TridentとNVEおよびNAEの相互運用性"。	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
snapshotDir	にアクセスするためのNASオプション .snapshot ディレクトリ。デフォルトはです false	true
exportPolicy	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
securityStyle	プロビジョニングされたNFSボリュームにアクセスするためのNASオプション。 NFS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

オプション	説明	例
`fileSystemType`	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`
`tieringPolicy`	使用する階層化ポリシー。デフォルトはです `none`; `snapshot-only` ONTAP 9.5より前のSVM-DR構成の場合	`none`

スケーリングオプション

。 `ontap-nas` および `ontap-san` ドライバによって、DockerボリュームごとにONTAP FlexVol が作成されます。ONTAP では、クラスタノードあたり最大 1 、 000 個の FlexVol がサポートされます。クラスタの最大 FlexVol 数は 12 、 000 です。この制限内にDockerボリュームの要件が収まる場合は、を参照してください `ontap-nas` FlexVolで提供されるDockerボリューム単位のSnapshotやクローニングなどの機能が追加されているため、NAS解決策 がドライバとして推奨されます。

FlexVol の制限で対応できない数のDockerボリュームが必要な場合は、を選択します `ontap-nas-economy` または `ontap-san-economy` ドライバ。

。 `ontap-nas-economy` ドライバによって、自動管理されるFlexVolのプール内に、Dockerボリュームがが作成される。qtree の拡張性は、クラスタノードあたり最大 10 、 000 、クラスタあたり最大 2 、 40 、 000 で、一部の機能を犠牲にすることで大幅に向上しています。。 `ontap-nas-economy` ドライバは、Dockerボリューム単位のスナップショットやクローニングをサポートしていません。

。 `ontap-nas-economy` ドライバは現在Docker Swarmではサポートされていません。Swarmは複数のノード間でのボリューム作成のオーケストレーションを行わないためです。

。 `ontap-san-economy` ドライバによって、自動で管理されるFlexVolの共有プール内にDockerボリュームがが作成される。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。

を選択します `ontap-nas-flexgroup` 数十億個のファイルを含むペタバイト規模に拡張可能な1つのボリュームへの並列処理能力を高めるドライバ。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 は、FlexGroup ボリュームのプロビジョニング時に SVM に割り当てられたすべてのアグリゲートを使用します。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。
- 推奨される最小 FlexGroup サイズは 100GB です。
- FlexGroup Volume ではクローニングはサポートされていません。

FlexGroup と FlexGroup に適したワークロードの詳細については、を参照してください "NetApp FlexGroup

Volume Best Practices and Implementation Guide"。

同じ環境で高度な機能と大規模な拡張性を実現するために、を使用して、Docker Volume Pluginの複数のインスタンスを実行できます `ontap-nas` を使用しています `ontap-nas-economy`。

ONTAP 構成ファイルの例

- NFSの例 `ontap-nas` ドライバ*

```
{
    "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"
    }
}
```

- NFSの例 `ontap-nas-flexgroup` ドライバ*

```
{
    "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"
    }
}
```

- NFSの例 `ontap-nas-economy` ドライバ*

```
{
    "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"
}
```

- iSCSIの例 `ontap-san` ドライバ*

```
{
    "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"
}
```

- NFSの例 `ontap-san-economy` ドライバ*

```
{
    "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"
}
```

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
`Endpoint`	\https：//<login>：<password>@<mvip>/ JSON -RPC /<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	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`
`InitiatorIFace`	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	`default`
`Types`	QoS の仕様	以下の例を参照してください
`LegacyNamePrefix`	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンの Trident を使用していて、既存のボリュームをアップグレードする場合は、この値を設定して、ボリューム名メソッドを使用してマッピングされた古いボリュームにアクセスする必要があります。	`netappdvp-`

。 `solidfire-san` ドライバはDocker Swarmをサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

既知の問題および制限事項

Astra Trident と Docker を使用する際の既知の問題と制限事項について説明しています。

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. 追加を指定してプラグインを再インストールします `config` パラメータ

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

ボリューム名は 2 文字以上にする必要があります。

これは Docker クライアントの制限事項です。クライアントは、 1 文字の名前を Windows パスと解釈します。 "バグ 25773 を参照"。

Docker Swarm には、 Astra Trident がストレージやドライバのあらゆる組み合わせでサポートしないようにする一定の動作があります。

- 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（ Astra 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。
  これは、ONTAP の仕組みとの仕組みによるものです `ontap-nas` および `ontap-san` ドライバ機能は、これらの制限内で動作できるようになる唯一の機能です。

その他のドライバには、競合状態などの問題があります。このような問題が発生すると、ボリュームを同じ名前で異なる ID にする機能が Element に備わっているため、「勝者」を明確にせずに 1 回の要求で大量のボリュームを作成できるようになります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。

よくある質問

Trident が提供する Astra 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 を掲載しています。

一般的な質問

Trident がリリースされる頻度を教えてください。

Trident は、 1 月、 4 月、 7 月、 10 月の 3 カ月ごとにリリースされます。Kubernetes のリリースから 1 カ月後です。

Astra 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。

Astra Trident は、通常、 Kubernetes で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。

Astra Trident には、他のネットアップ製品との依存関係はありますか。

Astra Trident は、他のネットアップソフトウェア製品に依存しないため、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。

Astra Trident の設定の詳細をすべて取得するにはどうすればよいですか。

を使用します `tridentctl get` コマンドを使用して、Astra Trident構成に関する詳細を確認できます。

Astra Trident を使用してストレージをプロビジョニングする方法に関するメトリクスを取得できますか。

はい。Trident 20.01 には、管理対象のバックエンドの数、プロビジョニングされたボリュームの数、消費されたバイト数など、 Astra Trident の動作に関する情報を収集するために使用できる Prometheus エンドポイントが導入されています。また、 Cloud Insights を使用して監視と分析を行うこともできます。

Astra Trident を CSI プロビジョニング担当者として使用すると、ユーザエクスペリエンスは変化しますか。

いいえユーザエクスペリエンスと機能に関する変更はありません。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新しい機能をすべて使用する場合は、 Astra Trident をインストールする方法を推奨します。

Kubernetes クラスタに Astra Trident をインストールして使用

のサポート対象のバージョンを指定します `etcd`？

Astra Tridentはもう必要ありません `etcd`。状態を維持するために CRD を使用します。

Astra 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、 Astra Trident はオフラインでインストールできます。を参照してください "こちらをご覧ください"。

Astra Trident はリモートからインストールできますか。

はい。Astra Trident 18.10以降では、を搭載した任意のマシンからリモートインストール機能がサポートされます `kubectl` クラスタへのアクセス。実行後 `kubectl` アクセスが検証されます（「開始」など） `kubectl get nodes` リモートマシンからコマンドを実行して確認）、インストール手順に従います。

Astra Trident でハイアベイラビリティを構成できますか。

Astra Trident は、 1 つのインスタンスで Kubernetes Deployment （ 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。導入環境内のレプリカの数は増やすべきではありません。Astra 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は、 Kubernetes によって、クラスタ内の正常なノードにポッドが自動的に再導入されます。Astra Trident はコントロールプレーンのみであるため、 Astra Trident を再導入しても、現在マウントされているポッドには影響しません。

Astra 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。

Astra Trident は Kubernetes API Server からデータを読み取り、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して、 kube-system へのアクセスを必要とします。

Astra Trident で使用されるロールと権限を教えてください。

Trident インストーラが Kubernetes ClusterRole を作成します。このロールには、 Kubernetes クラスタの PersistentVolume 、 PersistentVolumeClaim 、 StorageClass 、 Secret の各リソースへのアクセス権があります。を参照してください "こちらをご覧ください"。

Astra 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、マニフェストファイルである Astra Trident のインストールに使用するものをローカルで生成して変更できます。を参照してください "こちらをご覧ください"。

2 つの別々の Kubernetes クラスタに対して、同じ ONTAP バックエンド SVM を 2 つの別々の Astra Trident インスタンスに対して共有できますか。

推奨されませんが、同じバックエンド SVM を 2 つの Astra Trident インスタンスに使用できます。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、一意のボリューム名を指定します `StoragePrefix` のパラメータを指定します `setup/backend.json` ファイル。これは、両方のインスタンスで同じ FlexVol を使用しないためです。

ContainerLinux （旧 CoreOS ）に Astra Trident をインストールすることはできますか。

Astra Trident は Kubernetes ポッドとして機能し、 Kubernetes が実行されている場所に導入できます。

ネットアップの **Cloud Volumes ONTAP** で **Astra Trident** を使用できますか。

はい、 Astra Trident は AWS 、 Google Cloud 、 Azure でサポートされています。

Astra Trident は **Cloud Volume** サービスと連携していますか。

はい。 Astra Trident は、 Azure の Azure NetApp Files サービスと GCP の Cloud Volumes Service をサポートしています。

トラブルシューティングとサポート

ネットアップは **Astra Trident** をサポートしていますか。

Astra Trident はオープンソースであり、無償で提供されますが、ネットアップのバックエンドがサポートされていれば、完全にサポートされています。

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？

を実行すると、サポートバンドルを作成できます `tridentctl logs -a`。バンドルでキャプチャされたログに加えて、 kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、 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。

に問題を作成します "Astra Trident Github" そして、概要の件名と問題に「 * RFE * 」と明記してください。

不具合を発生させる場所

に問題を作成します "Astra Trident Github"。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。

ネットアップが **Trident** の **Astra** について簡単に質問できたらどうなりますか。コミュニティやフォーラムはありますか？

ご質問、ご質問、ご要望がございましたら、ネットアップのアストラからお問い合わせください "チャネルを外します" またはGitHub。

ストレージシステムのパスワードが変更され、 **Astra Trident** が機能しなくなった場合、どのように回復すればよいですか。

バックエンドのパスワードをで更新します `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。

Astra Trident が Kubernetes ノードを検出できない。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場合、次の 2 つのケースが考えられます。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。Astra Trident のインストール後にネットワークの変更が発生した場合、この問題が発生するのはクラスタに追加された新しい Kubernetes ノードだけです。

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。

Astra Trident をアップグレード

古いバージョンから新しいバージョンに直接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）？

ネットアップでは、 Astra Trident のメジャーリリースから次回のメジャーリリースへのアップグレードをサポートしています。バージョン 18.xx から 19.xx 、 19.xx から 20.xx 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。

ダウングレードする場合は、いくつかの要因を評価する必要があります。を参照してください "ダウングレードに関するセクション"。

バックエンドとボリュームを管理

ONTAP バックエンド定義ファイルに管理 LIF とデータ LIF の両方を定義する必要がありますか。

管理LIFは必須です。データLIFのタイプはさまざまです。

- ONTAP SAN：iSCSIには指定しないでください。Astra Tridentが使用 "ONTAP の選択的LUNマップ" iSCSI LIFを検出するには、マルチパスセッションを確立する必要があります。の場合は警告が生成されます `dataLIF` は明示的に定義されます。 を参照してください "ONTAP のSAN構成オプションと例" を参照してください。
- ONTAP NAS:を指定することを推奨します `dataLIF`。指定しない場合は、Astra TridentがSVMからデータLIFを取得します。NFSマウント処理に使用するFully Qualified Domain Name（FQDN；完全修飾ドメイン名）を指定して、ラウンドロビンDNSを作成して複数のデータLIF間で負荷を分散することができます。を参照してください "ONTAP NASの設定オプションと例" を参照してください

Astra Trident が ONTAP バックエンドに CHAP を設定できるか。

はい。20.04 以降、 Astra Trident は ONTAP バックエンドに対して双方向 CHAP をサポートします。これには設定が必要です `useCHAP=true` バックエンド構成

Astra 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。

Astra Trident では、バージョン 20.04 以降からエクスポートポリシーを動的に作成、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、 Astra Trident は特定の CIDR 内に IP アドレスが割り当てられたノードのルールの追加と削除を自動的に管理します。この機能には CSI Trident が必要です。

データ LIF にポートを指定できるか。

Astra Trident 19.01 以降では、 DataLIF にポートを指定できます。で設定します `backend.json` ファイルの形式 "`managementLIF": <ip address>:<port>`"。たとえば、管理LIFのIPアドレスが192.0.2.1で、ポートが1000の場合、を設定します `"managementLIF": "192.0.2.1:1000"`。

管理 LIF とデータ LIF に IPv6 アドレスを使用できますか。

Astra Tridentでは、次の機能に対してIPv6アドレスを定義できます。

- `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：

Astra Tridentは、を使用してインストールする必要があります `--use-ipv6` IPv6で動作するためのフラグ。

バックエンドの管理 LIF を更新できますか。

はい、を使用してバックエンドの管理LIFを更新できます `tridentctl update backend` コマンドを実行します

バックエンドのデータ LIF を更新できるか。

のデータLIFを更新できます `ontap-nas` および `ontap-nas-economy` のみ。

Kubernetes 向け Astra Trident で複数のバックエンドを作成できますか。

Astra Trident では、同じドライバまたは別々のドライバを使用して、多数のバックエンドを同時にサポートできます。

Astra 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。

Astra Trident では、バックエンドのクレデンシャルを Kubernetes のシークレットとして格納します。

Astra Trident で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したプールを自動的に選択できない場合は、を参照してください `storagePools` および `additionalStoragePools` パラメータは、特定のプールセットを選択するために使用します。

Astra 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。

。 `excludeStoragePools` パラメータを使用して、一連のプールをフィルタします。一連のプールがTridentからプロビジョニングに使用され、一致するプールは削除されます。

同じ種類のバックエンドが複数ある場合、 Astra Trident はどのバックエンドを使用するかをどのように選択しますか。

同じタイプのバックエンドが複数設定されている場合、Astra Tridentはにあるパラメータに基づいて適切なバックエンドを選択します `StorageClass` および `PersistentVolumeClaim`。たとえば、ONTAPとNASのドライババックエンドが複数ある場合、Astra Tridentは内のパラメータを照合しようとします `StorageClass` および `PersistentVolumeClaim` に記載された要件を提供できるバックエンドを組み合わせて組み合わせることができます `StorageClass` および `PersistentVolumeClaim`。この要求に一致するバックエンドが複数ある場合、 Astra Trident はいずれかのバックエンドからランダムに選択します。

Astra Trident は、 Element / SolidFire で双方向 CHAP をサポートしていますか。

はい。

Trident が ONTAP ボリュームに qtree を導入する方法を教えてください。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 に最大200個のqtreeを作成し（50~300で設定可能）、クラスタノードあたり100、000個のqtreeを、クラスタあたり240万個まで作成します。をクリックします `PersistentVolumeClaim` これは、エコノミードライバが対応しているため、ドライバは新しいqtreeを処理できるFlexVol がすでに存在するかどうかを調べます。qtree を提供できる FlexVol が存在しない場合は、新しい FlexVol が作成されます。

ONTAP NAS でプロビジョニングされたボリュームに UNIX アクセス権を設定するにはどうすればよいですか。

Astra Trident でプロビジョニングした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、デフォルトでマウントオプションが Kubernetes でどの値にも設定されていません。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、次の例を実行します "こちらをご覧ください"。

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、を使用します `exportPolicy` バックエンド定義ファイルで設定されたパラメータ。

ONTAP を使用して Astra Trident 経由でボリューム暗号化を設定する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。 "Astra TridentとNVEおよびNAEの相互運用性"

Trident 経由で ONTAP に QoS を実装するには、どのような方法が最適ですか。

使用 `StorageClasses` ONTAP にQoSを実装するには、次の手順を

Trident 経由でシンプロビジョニングや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を設定する必要があります `StorageClass`。両方が設定されている場合は、`StorageClass` 優先されます。ONTAP で次の項目を設定します。

1. オン `StorageClass`を設定します `provisioningType` シックとしての属性。
2. バックエンド定義ファイルで、を設定してシックボリュームを有効にします `backend spaceReserve parameter` ボリュームとして。

誤って PVC を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes 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。

Astra Trident によって作成された NFS PVC を拡張できますか。

はい。Astra Trident によって作成された PVC を拡張できます。ボリュームの自動拡張は ONTAP の機能であり、 Trident には適用されません。

Astra Trident の外部で作成したボリュームを Astra Trident にインポートできますか。

19.04 以降では、ボリュームインポート機能を使用してボリュームを Kubernetes に移行できます。

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.
```

Astra Trident によって作成された iSCSI PVC を拡張できますか。

Trident 19.10 は CSI プロビジョニング担当者を使用した iSCSI PVS の拡張をサポートしています。

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。容量不足が原因でネットアップストレージが Kubernetes のクォータ設定を受け入れられない場合、 Astra Trident はプロビジョニングを試みますがエラーになります。

Trident を使用してボリューム Snapshot を作成できますか。

はい。Trident が、 Snapshot からオンデマンドのボリューム Snapshot と永続的ボリュームを作成できるようになりました。スナップショットからPVSを作成するには、を確認してください `VolumeSnapshotDataSource` フィーチャーゲートが有効になりました。

Astra Trident のボリュームスナップショットをサポートするドライバを教えてください。

現在のところ、オンデマンドスナップショットがサポートされています `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `gcp-cvs`および `azure-netapp-files` バックエンドドライバ

ONTAP を使用して Astra Trident でプロビジョニングしたボリュームの Snapshot バックアップを作成する方法を教えてください。

これはで入手できます `ontap-nas`、 `ontap-san`および `ontap-nas-flexgroup` ドライバ。を指定することもできます `snapshotPolicy` をクリックします `ontap-san-economy` ドライバーはFlexVol レベルです。

この機能は、でも使用できます `ontap-nas-economy` ドライバの詳細は、FlexVol レベルではなく、qtreeレベルで表示されます。Astra Tridentによってプロビジョニングされたボリュームのスナップショット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設定します `snapshotPolicy` ONTAP バックエンドで定義されているSnapshotポリシーにコピーします。ストレージコントローラで作成された Snapshot は Astra Trident で認識されません。

Trident 経由でプロビジョニングしたボリュームの Snapshot リザーブの割合を設定できますか。

はい。を設定することで、Astra Tridentを使用して、Snapshotコピーを格納するためのディスクスペースの特定の割合を予約できます `snapshotReserve` バックエンド定義ファイルの属性。を設定している場合は `snapshotPolicy` および `snapshotReserve` バックエンド定義ファイルでは、に従ってSnapshotリザーブ

の割合が設定されます `snapshotReserve` バックエンドファイルに指定されている割合。状況に応じて `snapshotReserve` この割合は省略しています。ONTAP ではデフォルトでSnapshotリザーブの割合が5に設定されます。状況に応じて `snapshotPolicy` オプションがnoneに設定されている場合、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。Tridentがプロビジョニングしたボリュームの Snapshotディレクトリにアクセスするには、を設定します `snapshotDir` バックエンド定義ファイルのパラメータ。

Astra Trident を使用して、ボリューム用の **SnapMirror** をセットアップできますか。

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。

サポート

Astra Trident は、正式にサポートされているネットアップのプロジェクトです。任意の標準メカニズムを使用してネットアップに連絡し、必要なエンタープライズクラスのサポートを受けることができます。

また、ネットアップのアストラには、コンテナユーザ（Astra Trident開発者を含む）の活気あるパブリックコミュニティもあります "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。

トラブルシューティング

Astra Trident のインストール中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、ここに記載されているポインタを使用してください。

Astra Tridentのサポートを受けるには、を使用してサポートバンドルを作成してください `tridentctl logs -a -n trident` に送信します `NetApp Support <Getting Help>`。

トラブルシューティングに関する記事の包括的なリストについては、を参照してください "ネットアップナレッジベース（ログインが必要）"。また、 Astra に関連する問題のトラブルシューティングに関する情報も参照できます "こちらをご覧ください"。

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と（Tridentポッドがで停止した場合など） `ContainerCreating` 準備が完了したコンテナが2つ未満のフェーズ）を実行中であること `kubectl -n trident describe deployment trident` および `kubectl -n trident describe pod trident-**` 詳細な分析情報を提供できます。kubeletログの取得（例：Via `journalctl -xeu kubelet`)また有用である場合もある。
- Tridentのログに十分な情報がない場合は、にアクセスしてTridentのデバッグモードを有効にすることができます `-d` インストールオプションに基づいてインストールパラメータにフラグを設定します。

  次に、を使用してデバッグが設定されていることを `./tridentctl logs -n trident` を検索しています `level=debug msg` ログに記録されます。

  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、の出力の「経過時間」列を確認します `kubectl get pod -n trident`。

  Astra Trident 20.07および20.10では `tprov` の代わりに `torc`。

  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
  ```

- を含めて、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` バックエンドの定義に含まれています。たとえば、と指定します `debugTraceFlags: {"api":true, "method":true,}` Tridentログでapi呼び出しとメソッドの逆数を取得する。既存のバックエンドにはを追加できます `debugTraceFlags` を使用して設定します `tridentctl backend update`。
- Red Hat CoreOSを使用する場合は、次の点を確認してください `iscsid` はワーカーノードで有効になり、デフォルトで開始されます。この設定には、 OpenShift MachineConfig を使用するか、イグニッションテンプレートを変更します。
- Trident をで使用する際によく発生する問題です "Azure NetApp Files の特長" テナントとクライアントのシークレットが、必要な権限がないアプリケーションの登録から取得された場合です。Trident の要件の詳細については、を参照してください "Azure NetApp Files の特長" 設定
- PVをコンテナにマウントする際に問題が発生する場合は、を確認してください `rpcbind` をインストールして実行しておきます。ホストOSに必要なパッケージマネージャを使用して、かどうかを確認します `rpcbind` を実行しています。のステータスを確認できます `rpcbind` を実行してサービスを提供します `systemctl status rpcbind` またはそれと同等のものです。
- Tridentバックエンドがにあると報告した場合 `failed` 以前は対処したことがあるにもかかわらず、状況はバックエンドに関連付けられたSVM /管理者クレデンシャルの変更が原因であると考えられます。を使用したバックエンド情報の更新 `tridentctl update backend` Tridentポッドをバウンスすると、この問題 が修正されます。
- Kubernetes クラスタや Trident をアップグレードしてベータ版のボリューム Snapshot を使用する場合は、既存の alpha スナップショット CRS がすべて削除されていることを確認してください。その後、を使用できます `tridentctl obliviate alpha-snapshot-crd` アルファスナップショット作成の作成を削除するコマンド。を参照してください "この blog" アルファスナップショットの移行手順を理解する。
- TridentをDockerでコンテナランタイムとしてインストールする際に権限の問題が発生した場合は、Tridentのインストールをで試してください `--in cluster=false` フラグ。これはインストーラポッドを使用せず、に起因する許可の問題を回避する `trident-installer` ユーザ：
- を使用します `uninstall parameter <Uninstalling Trident>` 実行に失敗したあとにクリーンアップに使用します。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。
- Tridentの以前のバージョンにダウングレードする場合は、最初にを実行します `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新しい PVS が作成されておらず、既存の PVS / バックエンド / ストレージクラスに変更がない場合にのみ、ダウングレードを検討してください。Tridentは現在、状態を維持するためにSSDを使用しているため、作成されたすべてのストレージエンティティ（バックエンド、ストレージクラス、PVS、ボリュームSnapshot）にはが含まれています `associated CRD objects <Kubernetes CustomResourceDefinition Objects>` 以前にインストールされたTridentのバージョンで使用されていたPVに書き込まれるデータではありません。* 以前のバージョンに戻すと、新しく作成した PVS は使用できなくなります。 * バックエンド、 PVS 、ストレージクラス、ボリュームスナップショット（作成 / 更新 / 削除）などのオブジェクトに加えた変更は、ダウングレード時に Trident に表示されません * 。以前のバージョンの Trident で使用されていた PV は、 Trident から見ることができます。以前のバージョンに戻しても、アップグレードされていないかぎり、以前のリリースを使用してすでに作成された PVS へのアクセスは中断されません。
- Tridentを完全に削除するには、を実行します `tridentctl obliviate crd` コマンドを実行しますこれにより、すべての CRD オブジェクトが削除され、 CRD が定義解除されます。Trident は、すでにプロビジョニングされている PVS を管理しなくなります。

> Trident はその後、最初から再構成する必要があります。

- インストールが成功した後、PVCがにスタックしている場合 `Pending` 実行中のフェーズ `kubectl describe pvc` TridentがこのPVCのPVのプロビジョニングに失敗した理由について追加情報 に説明できる。

オペレータを使用して失敗した Trident の導入をトラブルシューティングします

オペレータを使用してTridentを導入している場合は、ステータスがになります `TridentOrchestrator` からの変更 `Installing` 終了： `Installed`。を確認した場合は `Failed` ステータスが表示され、オペレータが単独でリカバリできない場合は、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、
次の項目を参照してください： `TridentOrchestrator` ステータス。

```
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、がすでに存在することを示します `TridentOrchestrator`
これはTridentのインストールに使用された機能です。各Kubernetesクラスタで実行できるのは
Tridentのインスタンスを1つ使用することで、
アクティブな状態が1つだけ存在する時間 `TridentOrchestrator` それができること
を作成します。

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
```

ポッドを完全に初期化できないことがわかります。
1つ以上のコンテナイメージが取得されなかったためです。

問題に対処するには、を編集する必要があります `TridentOrchestrator` CR。
または、を削除することもできます `TridentOrchestrator`をクリックし、新しい
修正された正確な定義を持つもの。

を使用したTridentの導入に失敗した場合のトラブルシューティング `tridentctl`

何が問題になったかを確認するには、を使用してインストーラを再実行します `-d` 引数。デバッグモードをオンにして、問題の内容を理解するのに役立ちます。

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

問題に対処したら、次のようにインストールをクリーンアップし、を実行します `tridentctl install` コマンドの再実行：

```
./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.
```

ベストプラクティスと推奨事項

導入

Astra Trident の導入時には、ここに示す推奨事項を使用してください。

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できるため、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。Astra 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、権限が昇格されたサービスアカウントを利用します。

また、 Trident ポッドにアクセスすると、ユーザがストレージシステムのクレデンシャルやその他の機密情報にアクセスできるようになります。 アプリケーションユーザと管理アプリケーションが Trident オブジェクト定義またはポッド自体にアクセスできないようにすることが重要です。

クォータと範囲制限を使用してストレージ消費を制御します

Kubernetes には、 2 つの機能があります。これらの機能を組み合わせることで、アプリケーションによるリソース消費を制限する強力なメカニズムが提供されます。 。 "ストレージクォータメカニズム" 管理者は、グローバルおよびストレージクラス固有の、容量とオブジェクト数の使用制限をネームスペース単位で実装できます。さらに、を使用します "範囲制限" 要求がプロビジョニングツールに転送される前に、 PVC 要求が最小値と最大値の両方の範囲内にあることを確認します。

これらの値はネームスペース単位で定義されます。つまり、各ネームスペースに、リソースの要件に応じた値を定義する必要があります。の詳細については、こちらを参照してください "クォータの活用方法"。

ストレージ構成

ネットアップポートフォリオの各ストレージプラットフォームには、コンテナ化されたアプリケーションやそうでないアプリケーションに役立つ独自の機能があります。

プラットフォームの概要

Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の **SVM** を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。 SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします "SVM-DR" Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください "NetApp Hardware Universe の略" 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、 Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident `ontap-nas` および `ontap-san` ドライバによって、作成された各Kubernetes Persistent Volume（PV；永続ボリューム）用のFlexVolがプロビジョニングされます。。 `ontap-nas-economy` ドライバは、200 PVSごとに約1つのFlexVolを作成します（50~300で構成可能）。。 `ontap-san-economy` ドライバは、PVS 100個につきFlexVolを約1つ作成します（50~200の間で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

の値 `max-volumes` 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` 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ているボリュームの合計であり、個々のONTAP ノードではプロビジョニングされていません。その結果、 ONTAP クラスタノード

の Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、 2 ノードの ONTAP クラスタでは、最大 2 、 000 個の FlexVol をホストできます。最大ボリューム数を 1250 に設定していると、非常に妥当な結果が得られます。 ただし、のみの場合 "アグリゲート" あるノードから SVM に割り当てられている場合や、あるノードから割り当てられたアグリゲートをプロビジョニングできない場合（容量など）は、他のノードが Trident でプロビジョニングされたすべてのボリュームのターゲットになります。つまり、そのノードがボリューム数の上限に達するまでの可能性があります `max-volumes` の値に達したため、そのノードを使用するTridentと他のボリューム処理の両方に影響が生じます。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

Trident で作成できるボリュームの最大サイズを制限

Tridentで作成できるボリュームの最大サイズを設定するには、を使用します `limitVolumeSize` のパラメータ `backend.json` 定義（Definition）：

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。

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。双方向 CHAP のサポートは Trident 20.04 以降で利用できます。

SVM QoS ポリシーを作成して使用します

SVM に適用された ONTAP QoS ポリシーを使用すると、 Trident でプロビジョニングされたボリュームが使用できる IOPS の数が制限されます。 これはに役立ちます "Bully を防止します" Trident SVM 外のワークロードに影響を及ぼす、制御不能なコンテナ。

SVM の QoS ポリシーはいくつかの手順で作成します。正確な情報については、ご使用の ONTAP バージョンのマニュアルを参照してください。 次の例は、 SVM で使用可能な合計 IOPS を 5000 に制限する QoS ポリシーを作成します。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

また、使用しているバージョンの ONTAP でサポートされている場合は、最小 QoS を使用してコンテナ化されたワークロードへのスループットを保証することもできます。アダプティブ QoS は SVM レベルのポリシーには対応していません。

コンテナ化されたワークロード専用の IOPS は、さまざまな要素によって異なります。その中には、次のよう

なものがあります。

- ストレージアレイを使用するその他のワークロード。Kubernetes 環境とは関係なく、ストレージリソースを利用するほかのワークロードがある場合は、それらのワークロードが誤って影響を受けないように注意する必要があります。
- 想定されるワークロードはコンテナで実行されます。IOPS 要件が高いワークロードをコンテナで実行する場合は、 QoS ポリシーの値が低いとエクスペリエンスが低下します。

SVM レベルで割り当てた QoS ポリシーを使用すると、 SVM にプロビジョニングされたすべてのボリュームで同じ IOPS プールが共有されることに注意してください。コンテナ化されたアプリケーションの 1 つまたは少数のに高い IOPS が必要な場合、コンテナ化された他のワークロードに対する Bully になる可能性があります。その場合は、外部の自動化を使用したボリュームごとの QoS ポリシーの割り当てを検討してください。

ONTAP バージョン 9.8 より前の場合は、 QoS ポリシーグループを SVM * only * に割り当ててください。

Trident の **QoS** ポリシーグループを作成

Quality of Service （ QoS ；サービス品質）は、競合するワークロードによって重要なワークロードのパフォーマンスが低下しないようにします。ONTAP の QoS ポリシーグループには、ボリュームに対する QoS オプションが用意されており、ユーザは 1 つ以上のワークロードに対するスループットの上限を定義できます。QoS の詳細については、を参照してください "QoS によるスループットの保証"。
QoS ポリシーグループはバックエンドまたはストレージプールに指定でき、そのプールまたはバックエンドに作成された各ボリュームに適用されます。

ONTAP には、従来型とアダプティブ型の 2 種類の QoS ポリシーグループがあります。従来のポリシーグループは、最大スループット（以降のバージョンでは最小スループット）がフラットに表示されます。アダプティブ QoS では、ワークロードのサイズの変更に合わせてスループットが自動的に調整され、 TB または GB あたりの IOPS が一定に維持されます。これにより、何百何千という数のワークロードを管理する大規模な環境では大きなメリットが得られます。

QoS ポリシーグループを作成するときは、次の点に注意してください。

- を設定する必要があります `qosPolicy` キーを押します `defaults` バックエンド構成のブロック。次のバックエンド設定例を参照してください。

```
  ---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
- labels:
    performance: extreme
  defaults:
    adaptiveQosPolicy: extremely-adaptive-pg
- labels:
    performance: premium
  defaults:
    qosPolicy: premium-pg
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoS ポリシーグループの詳細については、を参照してください "ONTAP 9.8 QoS コマンド"。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Trident によって作成される NFS ボリュームと iSCSI LUN へのアクセスを制限することは、 Kubernetes 環境のセキュリティ体制に欠かせない要素です。これにより、 Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がないことです。つまり、 PVS はグローバルオブジェクトですが、 PVC にバインドされている場合は、同じネームスペース内のポッドからのみアクセス可能です。* 適切な場合は、名前空間を使用して分離することが重要です。 *

Kubernetes 環境でデータセキュリティを使用する場合、ほとんどの組織で最も懸念されるのは、コンテナ内のプロセスがホストにマウントされたストレージにアクセスできることですが、コンテナ用ではないためです。 "ネームスペース" この種の妥協を防ぐように設計されています。 ただし、特権コンテナという例外が 1 つあります。

権限付きコンテナは、通常よりもホストレベルの権限で実行されるコンテナです。デフォルトでは拒否されないため、を使用してこの機能を無効にしてください "ポッドセキュリティポリシー"。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にある

ターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（OpenShiftなど）や、ユーザアプリケーションをスケジュールできない他のノードを導入する場合は、ストレージリソースへのアクセスをさらに制限するために別 々 のエクスポートポリシーを使用する必要があります。これには、これらのインフラノードに導入されているサービス（ OpenShift Metrics サービスや Logging サービスなど）のエクスポートポリシーの作成と、非インフラノードに導入されている標準アプリケーションの作成が含まれます。

専用のエクスポートポリシーを使用します

Kubernetes クラスタ内のノードへのアクセスのみを許可するエクスポートポリシーが各バックエンドに存在することを確認する必要があります。Trident では、 20.04 リリース以降、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。

- を使用します `vserver export-policy create` ONTAP の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。
- 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 `vserver export-policy rule create` ONTAP CLIコマンド。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします `showmount` アプリケーション**SVM**用

。 `showmount` 機能を使用すると、NFSクライアントがSVMを照会して、使用可能なNFSエクスポートのリストを表示できます。Kubernetesクラスタに導入されたポッドは、問題 に対応しています `showmount -e` コマンドをデータLIFに対して実行し、アクセス権のないマウントも含めて使用可能なマウントのリストを取得します。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

を無効にする必要があります `showmount` SVMレベルのONTAP CLIコマンドを使用して、次の作業を行います。

```
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です	50です	15、000
最大 IOPS	パフォーマンスはこの制限を超えません。	50です	15、000	20万
バースト IOPS	短時間のバースト時に許容される最大 IOPS 。	50です	15、000	20万

Max IOPS と Burst IOPS は最大 200 、 000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロックサイズと帯域幅は、 IOPS に直接影響します。ブロックサイズが大きくなると、システムはそのブロックサイズを処理するために必要なレベルまで帯域幅を増やします。帯域幅が増えると、システムが処理可能な IOPS は減少します。を参照してください "SolidFire のサービス品質" QoS およびパフォーマンスの詳細については、を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（ VAG ）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、プロビジョニングするボリュームへのアクセスを制御します。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。

Trident と強化された CSI プロビジョニングツールは、 CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、 SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、 CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、 CHAP 認証の有無に

関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP 認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します "NetApp ライブラリ" 最新バージョンの場合。

- ONTAP *
- "NFS Best Practice and Implementation Guide"
- "SAN アドミニストレーションガイド" （ iSCSI の場合）
- "RHEL 向けの iSCSI のクイック構成"
- Element ソフトウェア *
- "SolidFire for Linux を設定しています"
- NetApp HCI *
- "NetApp HCI 導入の前提条件"
- "NetApp Deployment Engine にアクセスします"
- アプリケーションのベストプラクティス情報 *
- "ONTAP での MySQL に関するベストプラクティスです"
- "SolidFire での MySQL に関するベストプラクティスです"
- "NetApp SolidFire および Cassandra"
- "SolidFire での Oracle のベストプラクティス"
- "SolidFire での PostgreSQL のベストプラクティスです"

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です "NetApp ライブラリ" 最新のドキュメントを検索できます。

Astra Trident を統合

Astra Tridentを統合するには、設計とアーキテクチャに関する次の要素を統合する必要があります。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、ボリューム運用、Astra Tridentを使用したOpenShiftサービスの導入。

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、ONTAP のフルFlexVolです。
- `ontap-nas-economy`：PVがプロビジョニングされた各ボリュームはqtreeであり、FlexVolあたりのqtree数は設定可能です（デフォルトは200）。
- `ontap-nas-flexgroup`：すべてのONTAP FlexGroup としてプロビジョニングされたPVごとに、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-economy`：プロビジョニングされた各PVはLUNで、FlexVolあたりのLUN数は設定可能です（デフォルトは100）。

3 つの NAS ドライバの間で選択すると、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、 Astra Trident からすべての機能が提供されるわけではありません。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。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]
`ontap-nas-economy`	Yesfootnote: 3[]	Yesfootnote: 3[]	○脚注：5[]	はい。	Yesfootnote: 3[]	はい。	Yesfootnote: 3[]
`ontap-nas-flexgroup`	Yesfootnote: 1[]	いいえ	○脚注：5[]	はい。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]

Astra Trident は、 ONTAP 向けに 2 つの SAN ドライバを提供しています。このドライバ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。	Yesfootnote: 4[]	はい。	Yesfootnote: 1[]	はい。	Yesfootnote: 1[]
`ontap-san-economy`	はい。	はい。	Yesfootnote: 4[]	はい。	Yesfootnote: 3[]	はい。	Yesfootnote: 3[]

> 上記の表の脚注：
>
> Yes [1]：Astra Tridentで管理されない
>
> Yesfootnote: 2[]：Astra Tridentが管理しますが、PV Granularは管理しません
>
> Yesfootnote: 3[]：Astra Tridentで管理されず、PV Granularでは管理されない
>
> Yes [4]:raw-blockボリュームでサポート
>
> Yesfootnote: 5[]：CSI Tridentによるサポート

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、の機能の多くはです `ontap-nas` および `ontap-nas-economy` は同じです。しかし、だからです `ontap-nas-economy` ドライバは、PV単位でスケジュールを制御する機能を制限します。これは、ディザスタリカバリやバックアップ計画に特に影響を与える可能性があります。ONTAP ストレージでPVCクローン機能を利用したい開発チームの場合、この方法はを使用する場合にのみ使用できます `ontap-nas`、 `ontap-san` または `ontap-san-economy` ドライバ。

。 `solidfire-san` また、ドライバはPVCをクローニングすることもできます。

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です `ontap-nas`、 `ontap-nas-economy`、 `ontap-san` および `ontap-san-economy`。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 ONTAP を使用すると、お客様は使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、 AWS にデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用できます。FSX for ONTAP は、 ONTAP のファイルシステム機能と管理 API の多く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

。 `solidfire-san` NetApp HCI / SolidFireプラットフォームで使用されるドライバ。管理者は、QoS制限に基づいてTrident用にElementバックエンドを設定できます。Tridentで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、を使用してください `type` バックエンドファイル内のパラメータ。また、管理者は、を使用してストレージに作成できるボリュームサイズを制限することもできます `limitVolumeSize` パラメータ現在のところ、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、ではサポートされていません `solidfire-san` ドライバ。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
solidfire-san	はい。	はい。	○脚注： 2 []	はい。	はい。	はい。	Yesfootnote: 1[]

> 脚注：
>
> Yes [1]：Astra Tridentで管理されない
>
> Yes [2]:raw-blockボリュームでサポート

Azure NetApp Files バックエンドドライバ

Astra Tridentがを使用 `azure-netapp-files` を管理するドライバ "Azure NetApp Files の特長" サービス

このドライバの詳細と設定方法については、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	はい。	はい。	はい。	はい。	はい。	Yesfootnote: 1[]

> 脚注：
>
> Yes [1]：Astra Tridentで管理されない

Google Cloudバックエンドドライバ上の**Cloud Volumes Service**

Astra Tridentがを使用 `gcp-cvs` Google CloudのCloud Volumes Service にリンクするドライバ。

。 `gcp-cvs` ドライバは仮想プールを使用してバックエンドを抽象化し、Astra Tridentでボリュームの配置を判断できるようにします。管理者が、で仮想プールを定義します `backend.json` ファイル。ストレージクラスには、ラベルで仮想プールを識別するセレクタが使用されます。

- バックエンドに仮想プールが定義されている場合、Astra Tridentは、その仮想プールが制限されているGoogle Cloudストレージプール内にボリュームを作成しようとします。
- バックエンドに仮想プールが定義されていない場合、Astra Tridentは、リージョン内の使用可能なストレージプールからGoogle Cloudストレージプールを選択します。

Astra TridentでGoogle Cloudバックエンドを設定するには、と指定する必要があります `projectNumber`、`apiRegion`および `apiKey` バックエンドファイル内。プロジェクト番号はGoogle Cloudコンソールで確認できます。APIキーは、Google CloudでCloud Volumes Service のAPIアクセスを設定するときに作成したサービスアカウントの秘密鍵ファイルから取得されます。

Google Cloudのサービスタイプとサービスレベルに関するCloud Volumes Service の詳細については、を参照してください "CVS for GCPのAstra Tridentサポートについてご確認ください"。

Cloud Volumes Service for Google Cloudドライバ	Snapshot	クローン	マルチアタッチ	QoS	を展開します	レプリケーション
gcp-cvs	はい。	はい。	はい。	はい。	はい。	CVS -パフォーマンスサービスタイプでのみ利用できます。

レプリケーションに関する注意事項

- レプリケーションはAstra Tridentで管理されていません。
- クローンは、ソースボリュームと同じストレージプールに作成されます。

ストレージクラスの設計

Kubernetes ストレージクラスオブジェクトを作成するには、個々のストレージクラスを設定して適用する必要があります。このセクションでは、アプリケーション用のストレージクラスの設計方法について説明します。

特定のバックエンド使用率

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは、次の3セットのフィルタを設定できます。 `storagePools`、 `additionalStoragePools`または `excludeStoragePools`。

。 `storagePools` パラメータを指定すると、指定した属性に一致するプールのセットだけにストレージが制限されます。。 `additionalStoragePools` パラメータは、属性とで選択されたプールのセットに加えて、Astra 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されます `storagePools` パラメータ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` パラメータを使用すると、属性に一致する一連のプールが具体的に除外されます。

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QoSポリシーをエミュレートする場合は、でストレージクラスを作成します `media` 属性の形式 `hdd` または `ssd`。に基づきます `media` ストレージクラスで説明されている属性の中から、Tridentが提供する適切なバックエンドを選択します `hdd` または `ssd` media属性に一致するアグリゲートを作成し、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転送します。そこで、Premiumストレージクラスを作成します `media` 属性をとして設定します `ssd` Premium 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Astra Tridentバックエンドで利用可能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。セレクタで特定のラベルを設定したストレージクラスがある場合、 Astra 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した場合、Astra Tridentが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を1つ選択します。

仮想プールの設計

バックエンドの作成時に、一般に一連のパラメータを指定できます。管理者が、同じストレージクレデンシャルと異なるパラメータセットを使用して別のバックエンドを作成することはできませんでした。仮想プールの導入により、この問題 は軽減されました。仮想プールは、バックエンドとKubernetesストレージクラスの間に導入されたレベル抽象化です。管理者は、Kubernetes Storage Classesでセレクターとして参照できるラベルとともにパラメータをバックエンドに依存しない方法で定義できます。仮想プールは、サポートされているすべてのネットアップバックエンドにAstra Tridentを使用して定義できます。リストには、 SolidFire / NetApp HCI 、 ONTAP 、 GCP 上の Cloud Volumes Service 、 Azure NetApp Files が含まれます。

> ⓘ 仮想プールを定義する場合は、バックエンド定義で既存の仮想プールの順序を変更しないことをお勧めします。また、既存の仮想プールの属性を編集または変更したり、新しい仮想プールを定義したりしないことを推奨します。

さまざまなサービスレベル/**QoS**のエミュレート

サービスクラスをエミュレートするための仮想プールを設計できます。Cloud Volume Service for Azure NetApp Files の仮想プール実装を使用して、さまざまなサービスクラスをセットアップする方法を見ていきましょう。さまざまなパフォーマンスレベルを表す複数のラベルで ANF バックエンドを設定します。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たパラメータの中には、PVCを作成する際にAstra Tridentプロビジョニングの判断プロセスに影響するものがあります。

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。

Astra Trident は、次のマトリックスで指定された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側面が Kubernetes 以外で変更されることが防止されるわけではありません。特定のアプリケーション用にボリュームをカスタマイズしたり、誤って容量が消費されないようにしたり、何らかの理由でボリュームを別のストレージコントローラに移動したりする場合に便利です。

> Kubernetes のツリー内プロビジョニングツールは、現時点では NFS または iSCSI PVS のボリュームサイズ変更処理をサポートしていません。Astra Trident では、 NFS ボリュームと iSCSI ボリュームの両方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

Astra Trident は、 CSI フレームワークを使用して、オンデマンドでボリュームスナップショットを作成し、スナップショットから PVC を作成できます。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

Astra Trident は、ボリューム Snapshot からの PersistentVolumes の作成もサポートしています。これを実現するには、PersistentVolumeClaimを作成し、を指定します `datasource` ボリュームの作成元となる必要があるSnapshot。Astra Trident がこの PVC を処理するには、 Snapshot にデータが存在するボリュームを作成します。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、デスティネーションアグリゲートが Trident が使用している SVM からアクセス可能なアグリゲートであるかぎり、 Astra Trident または Kubernetes クラスタには影響しません。この点が重要なのは、アグリゲートが SVM に新たに追加された場合、 Astra Trident に再追加してバックエンドを更新する必要があることです。これにより、 Astra Trident が SVM のインベントリを再作成し、新しいアグリゲートが認識されるようになります。

ただし、バックエンド間でのボリュームの移動は Astra Trident では自動では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上の SVM 間が含まれます（たとえストレージシステムが Trident から Astra に接続されている場合でも）。

ボリュームが別の場所にコピーされた場合、ボリュームインポート機能を使用して現在のボリュームを Astra Trident にインポートできます。

ボリュームを展開します

Astra Trident は、 NFS と iSCSI PVS のサイズ変更をサポートしています。これにより、ユーザは Kubernetes レイヤを介してボリュームのサイズを直接変更できます。ボリュームを拡張できるのは、 ONTAP 、 SolidFire / NetApp HCI 、 Cloud Volumes Service バックエンドなど、主要なすべてのネットアップストレージプラットフォームです。あとで拡張できるようにするには、をに設定します `allowVolumeExpansion` 終了： `true` ボリュームに関連付けられているストレージクラス内のストレージクラス。永続ボリュームのサイズを変更する必要がある場合は、を編集します `spec.resources.requests.storage` Persistent Volume Claimのアノテーションを、必要なボリュームサイズに設定します。Tridentによって、ストレージクラスタ上のボリュームのサイズが自動的に変更されます。

既存のボリュームを **Kubernetes** にインポートする

Volume Import では、既存のストレージボリュームを Kubernetes 環境にインポートできます。これは現在、でサポートされています `ontap-nas`、 `ontap-nas-flexgroup`、 `solidfire-san`、 `azure-netapp-files`および `gcp-cvs` ドライバ。この機能は、既存のアプリケーションを Kubernetes に移植する場合や、ディザスタリカバリシナリオで使用する場合に便利です。

ONTAP およびを使用する場合 `solidfire-san` ドライバの場合は、コマンドを使用します `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml` 既存のボリュームをKubernetesにインポートしてAstra Tridentで管理import volume コマンドで使用した PVC YAML または JSON ファイルは、 Astra 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。

状況に応じて `azure-netapp-files` または `gcp-cvs` ドライバを使用する場合は、コマンドを使用します `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml` からKubernetesにボリュームをインポートしてAstra Tridentで管理。これにより、ボリューム参照が一意になります。

上記のコマンドを実行すると、 Astra Trident がバックエンド上にボリュームを検出し、サイズを確認します。設定された PVC のボリュームサイズが自動的に追加（必要に応じて上書き）されます。次に Astra Trident が新しい PV を作成し、 Kubernetes が PVC を PV にバインド

特定のインポートされた PVC を必要とするようにコンテナを導入した場合、ボリュームインポートプロセスによって PVC/PV ペアがバインドされるまで、コンテナは保留状態のままになります。PVC/PV ペアがバイ

ンドされると、他に問題がなければコンテナが起動します。

OpenShift サービスを導入します

OpenShift の付加価値クラスタサービスは、クラスタ管理者とホストされているアプリケーションに重要な機能を提供します。これらのサービスが使用するストレージはノードローカルリソースを使用してプロビジョニングできますが、これにより、サービスの容量、パフォーマンス、リカバリ性、持続可能性が制限されることがよくあります。エンタープライズストレージアレイを活用してこれらのサービスに容量を提供することで、劇的に向上したサービスを実現できます。ただし、すべてのアプリケーションと同様に、 OpenShift とストレージ管理者は、緊密に連携してそれぞれに最適なオプションを決定する必要があります。Red Hat のドキュメントは、要件を決定し、サイジングとパフォーマンスのニーズを確実に満たすために大きく活用する必要があります。

レジストリサービス

レジストリのストレージの導入と管理については、に記載されています "netapp.io のコマンドです" を参照してください "ブログ"。

ロギングサービス

他の OpenShift サービスと同様に、ログ記録サービスは、 Ansible と、インベントリファイル（別名）で提供される構成パラメータを使用して導入されますホスト。プレイブックに含まれています。インストール方法には、OpenShiftの初期インストール時にログを導入する方法と、OpenShiftが終了した後にログを導入する方法の2つがあります。
インストール済み。

> 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 の NFS サーバにはこのような問題はなく、簡単にロギング環境を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、Ansible変数を設定する必要があります `openshift_enable_unsupported_configurations` 終了： `true` インストーラが失敗しないようにします。

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。操作ログを配置する場合は、変数を指定します `openshift_logging_use_ops` として `true`サービスのインスタンスが2つ作成されます。操作のロ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。

導入方法に基づいて Ansible 変数を設定することは、基盤のサービスが正しいストレージを利用できるようにするために重要です。各導入方法のオプションを見てみましょう。

> 以下の表には、ロギングサービスに関連するストレージ構成に関連する変数のみが含まれています。その他のオプションは、で確認できます "Red Hat OpenShift のロギングに関するドキュメント" 導入環境に応じて、確認、設定、使用する必要があります。

次の表の変数では、入力した詳細を使用してロギングサービスの PV と PVC を作成する Ansible プレイブッ

クが作成されます。この方法は、 OpenShift インストール後にコンポーネントインストールプレイブックを使用するよりもはるかに柔軟性に劣るが、既存のボリュームがある場合はオプションとなります。

変数（ **Variable** ）	詳細
`openshift_logging_storage_kind`	をに設定します `nfs` ログ記録サービス用のNFS PVを作成するため。
`openshift_logging_storage_host`	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。
`openshift_logging_storage_nfs_directory`	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合などです `/openshift_logging`この変数には、このパスを使用します。
`openshift_logging_storage_volume_name`	名前。例 `pv_ose_logs`作成するPVの。
`openshift_logging_storage_volume_size`	たとえば、NFSエクスポートのサイズ `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_logging_es_pvc_dynamic`	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
`openshift_logging_es_pvc_storage_class_name`	PVC で使用されるストレージクラスの名前。
`openshift_logging_es_pvc_size`	PVC で要求されたボリュームのサイズ。
`openshift_logging_es_pvc_prefix`	ロギングサービスで使用される PVC のプレフィックス。
`openshift_logging_es_ops_pvc_dynamic`	をに設定します `true` 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。
`openshift_logging_es_ops_pvc_storage_class_name`	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`	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、 OpenShift のバージョンが異なるためわずかに変更される場合があるので、必ず読んで従うようにしてください "Red Hat OpenShift Container Platform 3.11 のドキュメント" 使用しているバージョンに対応した

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動拡張機能にも必要であり、多くの組織では、チャージバックやショーバックのアプリケーションに指標サービスのデータを使用しています。

ロギングサービスや OpenShift 全体と同様に、 Ansible を使用して指標サービスを導入します。また、ロギングサービスと同様に、メトリックサービスは、クラスタの初期セットアップ時またはコンポーネントのインストール方法を使用して運用可能になった後に導入できます。次の表に、指標サービスに永続的ストレージを設定する際に重要となる変数を示します。

以下の表には、指標サービスに関連するストレージ構成に関連する変数のみが含まれています。このドキュメントには、他にも導入環境に応じて確認、設定、使用できるオプションが多数あります。

変数（ **Variable** ）	詳細
`openshift_metrics_storage_kind`	をに設定します `nfs` ログ記録サービス用のNFS PVを作成するため。
`openshift_metrics_storage_host`	NFS ホストのホスト名または IP アドレス。これは SVM のデータ LIF に設定されている必要があります。
`openshift_metrics_storage_nfs_directory`	NFS エクスポートのマウントパス。たとえば、ボリュームがとしてジャンクションされている場合などです `/openshift_metrics`この変数には、このパスを使用します。
`openshift_metrics_storage_volume_name`	名前、 例： `pv_ose_metrics`作成するPVの。
`openshift_metrics_storage_volume_size`	たとえば、NFSエクスポートのサイズ `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_pvc_prefix`	メトリック PVC に使用するプレフィックス。
`openshift_metrics_cassandra_pvc_size`	要求するボリュームのサイズ。
`openshift_metrics_cassandra_storage_type`	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`	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、 OpenShift のインストール後に Ansible によって必要な PVC が作成

されます。また、 Trident 用のストレージをプロビジョニングしたあとにサービスを導入します。

上記の変数と導入プロセスは、 OpenShift の各バージョンで変更される可能性があります。必ず見直しを行ってください "RedHat OpenShift 導入ガイド" をバージョンに合わせて設定し、環境に合わせて設定します。

データ保護とディザスタリカバリ

Astra TridentとAstra 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。

Astra Tridentのレプリケーションとリカバリ

災害発生時にAstra Tridentをリストアするバックアップを作成できます。

Astra Tridentのレプリケーション

Astra Tridentは、Kubernetes CRDを使用して独自の状態の格納と管理を行い、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. 次のコマンドを使用してKubernetesクラスタetcdをバックアップします。 "Kubernetes：etcdクラスタのバックアップ"。
2. バックアップアーティファクトをFlexVolに配置します。

> FlexVolが配置されているSVMを別のSVMへのSnapMirror関係で保護することを推奨します。

Astra Tridentのリカバリ

Kubernetes CRDとKubernetesクラスタetcd Snapshotを使用して、Astra 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 `/etc/kubernetes/pki` 以下のetcdメンバーファイル `/var/lib/etcd`。
3. 次のコマンドを使用して、etcdバックアップからKubernetesクラスタをリストアします。 "Kubernetes：etcdクラスタのリストア"。
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レプリケーションとリカバリ

Astra Tridentではレプリケーション関係を設定できないが、ストレージ管理者は "ONTAP SnapMirror" SVMをレプリケートするため。

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 SVM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。実行時に選択したオプションを把握しておく必要があります。 Astra Tridentを使用したSVMのリカバリ。

- "-identity-preserve true" 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。

Astra Tridentを使用した**SVM**のリカバリ

Astra Trident では、 SVM の障害は自動では検出されない。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. シテイシタシヨウコウ `-identity-preserve false` または `-discard-config network` SVMレプリケーションを設定する際に、 `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 `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 `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. シテイシタシヨウコウ `-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。

シテイシタシヨウコウ `-identity-preserve true`デスティネーションSVMがアクティブ化されると、Astra Tridentで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

Astra TridentではSnapMirrorレプリケーション関係を設定できないが、ストレージ管理者は "ONTAPのSnapMirrorレプリケーションとリカバリ" Astra Tridentで作成されたボリュームをレプリケート

リカバリしたボリュームは、次のコマンドを使用してAstra Tridentにインポートできます： "tridentctlボリュームインポート"。

インポートは `ontap-nas-economy`、 `ontap-san-economy`または `ontap-flexgroup-economy` ドライバ。

Snapshotによるデータ保護

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"

Astra Control Centerアプリケーションのレプリケーション

Astra Controlを使用すると、SnapMirrorの非同期レプリケーション機能を使用して、データやアプリケーションの変更をクラスタ間でレプリケートできます。

"Astra Control：SnapMirrorテクノロジを使用してアプリケーションをリモートシステムにレプリケート"

セキュリティ

セキュリティ

ここに記載された推奨事項を参考に、Astra Tridentのインストールを安全に行ってください。

Astra Trident を独自のネームスペースで実行

アプリケーション、アプリケーション管理者、ユーザ、および管理アプリケーションが Astra Trident オブジェクト定義またはポッドにアクセスしないようにして、信頼性の高いストレージを確保し、悪意のあるアクティビティをブロックすることが重要です。

他のアプリケーションやユーザをAstra Tridentから分離するには、Astra Tridentを必ず独自のKubernetesネームスペースにインストールしてください (`trident`）。Astra Trident を独自の名前空間に配置することで、Kubernetes 管理担当者のみが Astra Trident ポッドにアクセスでき、名前空間 CRD オブジェクトに格納されたアーティファクト（バックエンドや CHAP シークレット（該当する場合）にアクセスできるようになります。
Astra Tridentのネームスペースにアクセスできるのは管理者だけであることを確認してから、にアクセスできるようにしてください `tridentctl` アプリケーション：

ONTAP SAN バックエンドで CHAP 認証を使用します

Astra Tridentは、ONTAP SANワークロードに対して（を使用して）CHAPベースの認証をサポート `ontap-san` および `ontap-san-economy` ドライバ）。ネットアップでは、ホストとストレージバックエンドの間の認証に、双方向 CHAP と Astra Trident 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、Astra Tridentは双方向CHAPを設定し、を使用してCHAPユーザ名とシークレットを管理できます `tridentctl`。
を参照してください "こちらをご覧ください" ONTAP バックエンドで Trident が CHAP を構成する方法をご確認ください。

ONTAP バックエンドの CHAP サポートは Trident 20.04 以降で利用可能

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。Astra 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CSIプロビジョニングツールとしてインストールすると、CHAPシークレットが管理され、に格納されます `tridentvolume` 対応するPVのCRオブジェクト。PV を作成すると、CSI Astra 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。

CSI Trident によって作成されたボリュームは、どのボリュームアクセスグループにも関連付けられていません。

CSI 以外のフロントエンドでは、ワーカーノード上のデバイスとしてのボリュームの接続は Kubernetes で処理されます。ボリュームの作成後、 Astra Trident が NetApp HCI / SolidFire システムに対して API 呼び出しを実行し、テナントのシークレットがない場合はシークレットを取得します。Trident が Kubernetes にシークレットを渡します。各ノード上の kubelet は Kubernetes API を介してシークレットにアクセスし、ボリュームにアクセスする各ノードとボリュームが配置されている NetApp HCI / SolidFire システム間で CHAP を実行 / 有効化するために使用します。

NVEおよびNAEでAstra Tridentを使用する

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"。

- NAEがバックエンドで有効になっている場合は、Astra TridentでプロビジョニングされたすべてのボリュームがNAEに対応します。
- NAEがバックエンドで有効になっていない場合、NVE暗号化フラグをに設定していないかぎり、Astra TridentでプロビジョニングされたすべてのボリュームがNVE対応になります `false` バックエンド構成

NAE対応バックエンドのAstra Tridentで作成されるボリュームは、NVEまたはNAEで暗号化されている必要があります。

- NVE暗号化フラグはに設定できます `true` Tridentバックエンド構成でNAE暗号化を無効にし、ボリューム単位で特定の暗号化キーを使用します。
- NVE暗号化フラグをに設定する `false` NAEが有効なバックエンドでは、NAEが有効なボリュームが作成されます。NAE暗号化を無効にするには、NVE暗号化フラグをに設定します `false`。

- 明示的にNVE暗号化フラグをに設定することで、Astra TridentでNVEボリュームを手動で作成できます `true`。

バックエンド構成オプションの詳細については、以下を参照してください。

- "ONTAP のSAN構成オプション"
- "ONTAP NASの構成オプション"

Linux Unified Key Setup（LUKS；統合キーセットアップ）

Linux Unified Key Setup（LUKS；ユニファイドキーセットアップ）を有効にして、Astra Trident上のONTAP SANおよびONTAP SANエコノミーボリュームを暗号化できます。Astra Tridentは、LUKS暗号化ボリュームのパスフレーズローテーションとボリューム拡張をサポートしています。

Astra Tridentでは、で推奨されるとおり、LUKSによって暗号化されたボリュームがAES-XTS -原64定型とモードを使用します "NIST"。

作業を開始する前に

- ワーカーノードにはcryptsetup 2.1以上（3.0よりも下位）がインストールされている必要があります。詳細については、を参照してください "Gitlab: cryptsetup"。
- パフォーマンス上の理由から、ワーカーノードでAdvanced Encryption Standard New Instructions（AES-NI）をサポートすることを推奨します。AES-NIサポートを確認するには、次のコマンドを実行します。

```
grep "aes" /proc/cpuinfo
```

何も返されない場合、お使いのプロセッサはAES-NIをサポートしていません。AES-NIの詳細については、以下を参照してください。 "Intel：Advanced Encryption Standard Instructions（AES-NI）"。

LUKS暗号化を有効にします

ONTAP SANおよびONTAP SANエコノミーボリュームでは、Linux Unified Key Setup（LUKS；Linux統合キーセットアップ）を使用して、ボリューム単位のホスト側暗号化を有効にできます。

手順

1. バックエンド構成でLUKS暗号化属性を定義します。ONTAP SANのバックエンド構成オプションの詳細については、を参照してください "ONTAP のSAN構成オプション"。

```
"storage": [
    {
        "labels":{"luks": "true"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "true"
        }
    },
    {
        "labels":{"luks": "false"},
        "zone":"us_east_1a",
        "defaults": {
            "luksEncryption": "false"
        }
    },
]
```

2. 使用 `parameters.selector` LUKS暗号化を使用してストレージプールを定義する方法。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. LUKSパスフレーズを含むシークレットを作成します。例：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

制限

LUKSで暗号化されたボリュームは、ONTAP の重複排除と圧縮を利用できません。

LUKSボリュームをインポートするためのバックエンド構成

LUKSボリュームをインポートするには、を設定する必要があります `luksEncryption` 終了：(`true` バックエンドにあります。。 `luksEncryption` optionを指定すると、ボリュームがLUKS準拠かどうかがAstra Tridentに通知されます (`true`）またはLUKS準拠ではありません (`false`）をクリックします。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

LUKSパスフレーズをローテーションします

LUKSのパスフレーズをローテーションしてローテーションを確認できます。

> パスフレーズは、ボリューム、Snapshot、シークレットで参照されなくなることを確認するまで忘れないでください。参照されているパスフレーズが失われた場合、ボリュームをマウントできず、データが暗号化されたままアクセスできなくなることがあります。

このタスクについて

LUKSパスフレーズのローテーションは、ボリュームをマウントするポッドが、新しいLUKSパスフレーズの指定後に作成されたときに行われます。新しいポッドが作成されると、Astra TridentはボリュームのLUKSパスフレーズをシークレット内のアクティブなパスフレーズと比較します。

- ボリュームのパスフレーズがシークレットでアクティブなパスフレーズと一致しない場合、ローテーションが実行されます。
- ボリュームのパスフレーズがシークレットのアクティブなパスフレーズと一致する場合は、を参照してください `previous-luks-passphrase` パラメータは無視されます。

手順

1. を追加します `node-publish-secret-name` および `node-publish-secret-namespace` StorageClassパラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. ボリュームまたはSnapshotの既存のパスフレーズを特定します。

   ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

   スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. ボリュームのLUKSシークレットを更新して、新しいパスフレーズと前のパスフレーズを指定します。確認します `previous-luke-passphrase-name` および `previous-luks-passphrase` 前のパスフレーズと同じにします。

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. ボリュームをマウントする新しいポッドを作成します。これはローテーションを開始するために必要です。

5. パスフレーズがローテーションされたことを確認します。

ボリューム

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

スナップショット

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

結果

パスフレーズは、ボリュームとSnapshotに新しいパスフレーズのみが返されたときにローテーションされました。

> たとえば、2つのパスフレーズが返された場合などです `luksPassphraseNames: ["B", "A"]`回転が不完全です。回転を完了するために、新しいポッドをトリガできます。

ボリュームの拡張を有効にします

LUKS暗号化ボリューム上でボリューム拡張を有効にできます。

手順

1. を有効にします `CSINodeExpandSecret` 機能ゲート（ベータ1.25+）。を参照してください "Kubernetes 1.25：CSIボリュームのノードベースの拡張にシークレットを使用します" を参照してください。
2. を追加します `node-expand-secret-name` および `node-expand-secret-namespace` StorageClass パラメータ。例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: netapp.io/trident
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。

参照

Astra Trident ポート

Tridentが通信に使用するポートの詳細をご確認ください。

Astra Trident ポート

Astra Trident は次のポート経由で通信：

ポート	目的
8443	バックチャネル HTTPS
8001	Prometheus 指標エンドポイント
8、000	Trident REST サーバ
17546	Trident デミ作用 / レディネスプローブポートは、 Trident デミ作用ポッドで使用されます

活性/レディネスプローブポートは、を使用して設置するときに変更できます `--probe-port` フラグ。このポートがワーカーノード上の別のプロセスで使用されていないことを確認することが重要です。

Astra Trident REST API

間 "tridentctl コマンドとオプション" Trident REST APIを使用するには、RESTエンドポイントを直接使用する方法が最も簡単です。

REST APIを使用する状況

REST APIは、Kubernetes以外の環境でAstra 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。

セキュリティ強化のため、Astra Tridentをぜひご利用ください `REST API` ポッド内で実行されている場合は、デフォルトでlocalhostに制限されます。この動作を変更するには、Astra Tridentを設定する必要があります `-address` 引数をポッド構成で指定します。

REST APIを使用する

API は次のように機能します。

`GET`

- `GET <trident-address>/trident/v1/<object-type>`:そのタイプのすべてのオブジェクトを一覧表示します。
- `GET <trident-address>/trident/v1/<object-type>/<object-name>`:名前付きオブジェクトの詳細を取得します。

`POST`

`POST <trident-address>/trident/v1/<object-type>`:指定した型のオブジェクトを作成します。

- オブジェクトを作成するには JSON 構成が必要です。各オブジェクトタイプの仕様については、tridentctl.htmlを参照してください[`tridentctl` コマンドとオプション]。
- 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。

`DELETE`

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`:指定したリソースを削除します。

> バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、tridentctl.htmlを参照してください[`tridentctl` コマンドとオプション]。

これらのAPIの呼び出し方法の例については、デバッグを渡してください (`-d`）フラグ。詳細については、tridentctl.htmlを参照してください[`tridentctl` コマンドとオプション]。

コマンドラインオプション

Trident 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をいくつか公開しています。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`:デバッグ出力をイネーブルにします。
- `-loglevel <level>`:ログレベルを設定します(デバッグ、情報、警告、エラー、致命的)。デフォルトは info です。

Kubernetes

- `-k8s_pod`：このオプションまたはを使用します `-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ます `-k8s_pod` をクリックしてKubernetesのサポートを有効にし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することができますが、サポートされるのは API サーバへのセキュアでない接続だけです。セキュアに接続するには、Tridentをポッドに搭載し、を使用して導入します `-k8s_pod` オプション
- `-k8s_config_path <file>`:必須。このパスをKubeConfigファイルに指定する必要があります。

Docker です

- `-volume_driver <name>`：Dockerプラグインの登録時に使用するドライバ名。デフォルトはです `netapp`。
- `-driver_port <port-number>`: UNIXドメインソケットではなく、このポートでリッスンします。
- `-config <file>`:必須。バックエンド構成ファイルへのパスを指定する必要があります。

REST

- `-address <ip-or-host>`：TridentのRESTサーバがリスンするアドレスを指定します。デフォルトは localhost です。localhost で聞いて Kubernetes ポッド内で実行しているときに、 REST インターフェイスにポッド外から直接アクセスすることはできません。使用 `-address ""` RESTインターフェイスにポッドのIPアドレスからアクセスできるようにするため。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

- `-port <port-number>`：TridentのRESTサーバがリスンするポートを指定します。デフォルトは 8000 です。
- `-rest`：RESTインターフェイスを有効にします。デフォルトは true です。

ネットアップの製品が Kubernetes と統合されます

ネットアップ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、高度なデータ管理機能を提供して、 Kubernetes 環境の機能、機能、パフォーマンス、可用性を強化します。

アストラ

"アストラ" Kubernetes 上で実行される大量のデータコンテナ化ワークロードを、パブリッククラウドとオンプレミスの間で簡単に管理、保護、移動できます。Astra は、ネットアップの実績のある拡張可能なストレージポートフォリオから、パブリッククラウドとオンプレミスに提供される Trident を使用して、永続的なコンテナストレージをプロビジョニングし、提供します。また、 Snapshot 、バックアップとリストア、アクティビティログ、アクティブクローニングによるデータ保護、ディザスタ / データリカバリ、データ監査、Kubernetes ワークロードの移行のユースケースなど、アプリケーションに対応した高度なデータ管理機能も豊富に搭載されています。

ONTAP

ONTAP は、あらゆるアプリケーションに高度なデータ管理機能を提供する、ネットアップ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す。ONTAP システムには、オールフラッシュ、ハイブリッド、オール HDD のいずれかの構成が採用されており、自社開発のハードウェア（ FAS と AFF ）、ノーブランド製品（ ONTAP Select ）、クラウドのみ（ Cloud Volumes ONTAP ）など、さまざまな導入モデルが用意されています。

> Trident は、上記の ONTAP 導入モデルをすべてサポートしています。

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。Cloud Volumes ONTAP は、本番ワークロード、ディザスタリカバリ、 DevOps 、ファイル共有、データベース管理に使用できます。ストレージ効率、高可用性、データレプリケーション、データ階層化、アプリケーションの整合性を提供することで、エンタープライズストレージをクラウドに拡張します。

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・システムを搭載したファイル・システムの起動と実行を可能にする、フルマネージドの AWS サービスです。FSX for ONTAP を使用すると、お客様は使い慣れたネットアップの機能、パフォーマンス、管理機能を活用しながら、 AWS にデータを格納する際のシンプルさ、即応性、セキュリティ、拡張性を活用できます。FSX for ONTAP は、ONTAP のファイルシステム機能と管理 API の多くをサポートしています。

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。Element と API を組み合わせることでストレージ管理のあらゆる要素を自動化できるため、ストレージ管理者は少ない労力で多くの作業を行うことができます。

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。

NetApp HCI は Trident によって完全にサポートされています。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。

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 Service for Google Cloud

"NetApp Cloud Volumes Service for Google Cloud" は、 NFS や SMB 経由で NAS ボリュームにオールフラッシュのパフォーマンスを提供する、クラウドネイティブのファイルサービスです。このサービスにより、従来型アプリケーションを含むあらゆるワークロードをGCPクラウドで実行できます。フルマネージドサービスを提供し、一貫したハイパフォーマンス、瞬時のクローニング、データ保護、 Google Compute Engine （ GCE ）インスタンスへのセキュアなアクセスを実現します。

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. ユーザがを作成します `PersistentVolumeClaim` 新しいを要求しています `PersistentVolume` 特定のサイズのものをKubernetesから取得します `StorageClass` 以前に管理者によって設定されていたもの。
2. Kubernetes `StorageClass` Tridentをプロビジョニングツールとして特定し、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設定します。
3. Tridentはその外観を独自にしています `StorageClass` 一致するものと同じ名前を使用します `Backends` および `StoragePools` を使用して、クラスのボリュームをプロビジョニングできます。
4. Tridentは、一致するバックエンドにストレージをプロビジョニングし、2つのオブジェクトを作成します。A `PersistentVolume` Kubernetesで、ボリュームとTrident内のボリュームを検出、マウント、処理し、間の関係を保持する方法を指示します `PersistentVolume` 実際のストレージをサポートします。
5. Kubernetesがをバインド `PersistentVolumeClaim` を新しいに変更します `PersistentVolume`。を含むポッド `PersistentVolumeClaim` このPersistentVolumeを、実行されている任意のホストにマウントします。
6. ユーザがを作成します `VolumeSnapshot` を使用した既存のPVCの `VolumeSnapshotClass` Tridentを指しています。
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、を作成します `VolumeSnapshotContent` これにより、Snapshotの識別方法をKubernetesに指示します。
8. ユーザはを作成できます `PersistentVolumeClaim` を使用します `VolumeSnapshot` をソースとして使用します。
9. Tridentが必要なSnapshotを特定し、の作成と同じ手順を実行します `PersistentVolume` および `Volume`。

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を `PersistentVolumeClaim` オブジェクトは、Kubernetesクラスタユーザが作成する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

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/cloneFromPVC	cloneSourceVolume の実行中です	ONTAP-NAS、 ontap-san、solidfire-san、azure-netapp-files、gcp-cvs、 ONTAP - SAN - 経済性
trident.netapp.io/splitOnClone	splitOnClone	ONTAP - NAS 、 ONTAP - SAN
trident.netapp.io/protocol	プロトコル	任意
trident.netapp.io/exportPolicy	エクスポートポリシー	ONTAP-NAS、 ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAP-NAS、 ontap-nas-economy、ontap-nas-flexgroup、ontap-SAN
trident.netapp.io/snapshotReserve	Snapshot リザーブ	ONTAP-NAS、 ontap-nas-flexgroup、ontap-san 、gcp-cvs
trident.netapp.io/snapshotDirectory	snapshotDirectory の略	ONTAP-NAS、 ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ONTAP-NAS、 ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/blockSize	ブロックサイズ	solidfire - SAN

作成されたPVにがある場合 `Delete` ポリシーを再利用すると、PVが解放されたとき（つまり、ユーザがPVCを削除したとき）に、Tridentは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がを使用している場合 `Retain` Tridentはポリシーを無視し、管理者がKubernetesとバックエンドからクリーンアップすることを前提としているため、ボリュームを削除する前にバックアップや検査を実行できます。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を使用して削除する必要があります (`tridentctl`）。

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。を参照してください `On-Demand Volume Snapshots` これがどのように機能するかを確認します。

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アノテーションを使用すると、CSI実装（Kubernetes 1.13以前）を使用せずにPVCをクローニングできます。また、KubernetesリリースでボリュームSnapshotのベータ版がサポートされていない場合（Kubernetes 1.16以前）も同様です。Trident 19.10 は、 PVC からのクローニングの CSI ワークフローをサポートしていることに注意してください。

を使用できます `cloneFromPVC` および `splitOnClone` CSI Tridentの注釈と従来のCSI以外のフロントエンド。

次に例を示します。ユーザがすでにというPVCを持っている場合 `mysql`を使用すると、ユーザはという新しいPVCを作成できます `mysqlclone などのアノテーションを使用する trident.netapp.io/cloneFromPVC: mysql。このアノテーションセットを使用すると、 Trident はボリュームをゼロからプロビジョニングするのではなく、 MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- アイドルボリュームのクローンを作成することを推奨します。
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- を使用 ontap-nas および ontap-san ドライバが必要な場合は、PVC注釈を設定することをお勧めします trident.netapp.io/splitOnClone と組み合わせて使用します trident.netapp.io/cloneFromPVC。を使用 trident.netapp.io/splitOnClone をに設定します true`Tridentでは、クローニングされたボリュームを親ボリュームからスプリットするため、ストレージ効率を維持しないまま、クローニングされたボリュームのライフサイクルを完全に分離します。設定されていません `trident.netapp.io/splitOnClone またはに設定します false 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のではなく、バックエンドのスペース消費が削減されます。そのため、クローンを先に削除しないかぎり親ボリュームを削除できません。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。Trident ボリュームに関連するパラメータと設定の完全な概要については、 Trident ボリュームオブジェクトを参照してください。

Kubernetes PersistentVolume オブジェクト

Kubernetesを PersistentVolume オブジェクトは、Kubernetesクラスタで使用可能になるストレージを表します。ポッドに依存しないライフサイクルがあります。

Tridentが実現 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Kubernetesクラスタに自動的に登録します。自分で管理することは想定されていません。

Tridentベースを参照するPVCを作成する場合 `StorageClass`Tridentは、対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しい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 StorageClass オブジェクトは、の名前で指定します PersistentVolumeClaims 一連のプロパティを指定して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。

Kubernetesを `StorageClass` Tridentを使用するオブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、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	バックエンド名とリストのマッピング ストレージプール
AdditionalStoragePools	[string] StringList をマップします	いいえ	バックエンド名のマップ ストレージプールノリスト
excludeStoragePools	[string] StringList をマップします	いいえ	ハツクエントメイノマツヒンク ストレージプールノリスト

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック：All ONTAP ；thin ：All ONTAP & solidfire-san-SAN
backendType	文字列	ONTAPNAS、ONTAPNASエコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN、solidfire-san-SAN、solidfire-san-SAN、GCP-cvs、azure-NetApp-files 、ONTAP-SAN-bエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、 Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、 ONTAP-NAS-エコノミー 、 ONTAP-NAS-FlexArray グループ、 ONTAP-SAN

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
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、を使用できます `attributes` 特定のクラスのニーズを満たすために必要なストレージの品質をモデル化することだけを目的としています。Tridentは、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` を指定します。

自分が使用できない場合は `attributes` クラスに適したプールを自動的に選択するには、を使用します `storagePools` および `additionalStoragePools` プールをさらに細かく指定するためのパラメータ、または特定のプールセットを選択するためのパラメータ。

を使用できます `storagePools` 指定したパラメータに一致するプールをさらに制限します `attributes`。つまり、Tridentはによって識別されたプールの交点を使用します `attributes` および `storagePools` プロビジョニングのパラメータ。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

を使用できます `additionalStoragePools` Tridentがプロビジョニングに使用する一連のプールを、で選択されているプールに関係なく拡張するためのパラメータ `attributes` および `storagePools` パラメータ

を使用できます `excludeStoragePools` 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するためのパラメータ。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。

を参照してください `storagePools` および `additionalStoragePools` パラメータを指定すると、各エントリの形式がになります `<backend>:<storagePoolList>`、ここで `<storagePoolList>` は、指定したバックエンドのストレージプールをカンマで区切ったリストです。たとえば、の値などです `additionalStoragePools` 次のように表示されます `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`。
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。を使用できます `tridentctl get backend` バックエンドとそのプールのリストを取得します。

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に 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 バージョン

FSstype （英語）	文字列	ext4 、 ext3 、 xfs など	ブロックのファイルシステムタイプ 個のボリューム	solidfire-san-group、ontap/nas、ontap -nas-エコノミー、ontap -nas-flexgroup 、ontap -san 、ONTAP - SAN -経済性	すべて
allowVolumeExpansion の略	ブール値	true false	PVC サイズの拡張のサポートをイネーブルまたはディセーブルにします	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 ONTAP-SAN-エコノミー 、 solidfire-san-, gcvs, azure-netapp-files	1.11 以上
volumeBindingMode のようになりました	文字列	即時、 WaitForFirstConsumer	ボリュームバインドと動的プロビジョニングを実行するタイミングを選択します	すべて	1.19 ~ 1.26

- 。 `fsType` パラメータは、SAN LUNに必要なファイルシステムタイプを制御する場合に使用します。また、Kubernetesでは、の機能も使用されます `fsType` ファイルシステムが存在することを示すために、ストレージクラスに格納します。ボリューム所有権は、を使用して制御できます `fsGroup` ポッドのセキュリティコンテキスト（使用する場合のみ） `fsType` が設定されます。を参照してください "Kubernetes ：ポッドまたはコンテナのセキュリティコンテキストを設定します" を使用したボリューム所有権の設定の概要については、を参照してください `fsGroup` コンテキスト（Context）。Kubernetesでが適用されます `fsGroup` 次の場合のみ値を指定します

  - `fsType` はストレージクラスで設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 NFS エクスポートにはファイルシステムがすでに存在します。を使用します `fsGroup` ストレージクラスでは、引き続きを指定する必要があります `fsType`。に設定できます `nfs` またはnull以外の値。

- 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のインストーラバンドルには、でTridentで使用するストレージクラス定義の例がいくつか含まれています `sample-input/storage-class-*.yaml`。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

Kubernetes VolumeSnapshotClass オブジェクト

Kubernetes VolumeSnapshotClass オブジェクトはに似ています StorageClasses。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。

A VolumeSnapshotClass Snapshotを作成するには、管理者によって定義されている必要があります。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 のボリュームSnapshotを要求するKubernetesに指定します csi-snapclass クラスはTridentによって処理されます。。 deletionPolicy Snapshotを削除する必要がある場合に実行する処理を指定します。いつ deletionPolicy がに設定されます Delete`を指定すると、Snapshotが削除されたときに、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が削除されます。または、に設定します `Retain はそのことを示します VolumeSnapshotContent 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を VolumeSnapshot objectは、ボリュームのSnapshotを作成する要求です。PVC がボリュームに対するユーザ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Snapshot要求が開始されると、Tridentはバックエンドでのボリュームの Snapshotの作成を自動的に管理し、一意のを作成してSnapshotを公開します VolumeSnapshotContent オブジェクト。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。

> VolumeSnapshot のライフサイクルはソース PVC とは無関係です。ソース PVC が削除されても、スナップショットは維持されます。スナップショットが関連付けられている PVC を削除すると、 Trident はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes VolumeSnapshotContent オブジェクト

Kubernetesを 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作成されたSnapshotを表します。これはに似ています PersistentVolume とは、ストレージクラスタにプロビジョニングされたSnapshotを表します。に似ています PersistentVolumeClaim および PersistentVolume オブジェクト。スナップショットが作成されると、が表示されます VolumeSnapshotContent オブジェクトは、への1対1のマッピングを保持します VolumeSnapshot オブジェクト。オブジェクトはSnapshotの作成を要求しました。

Tridentが実現 `VolumeSnapshotContent` 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Kubernetesクラスタに自動的に登録します。自分で管理することは想定されていません。

。 `VolumeSnapshotContent` Objectには、など、Snapshotを一意に識別する詳細が含まれます `snapshotHandle`。これ `snapshotHandle` は、PVの名前との名前を一意に組み合わせたものです `VolumeSnapshotContent` オブジェクト。

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によってが設定されます `VolumeSnapshotContent` オブジェクトを作成することで、Snapshot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。を実行すると、これらのリソース定義を取得できます `kubectl get crds`。

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。

19.07リリース以降、Tridentはいくつかのを使用します `CustomResourceDefinition` Tridentバックエンド、Tridentストレージクラス、Tridentボリュームなど、TridentオブジェクトのIDを保持するオブジェクト。これらのオブジェクトは Trident によって管理されます。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、を使用してバックエンドを作成する場合を示します `tridentctl`に対応します `tridentbackends` CRDオブジェクトは、Kubernetesによって消費されるために作成されます。

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。
- 以前のバージョンのTrident（使用していたもの）からアップグレードする場合 `etcd` ステートを維持するために）、Tridentインストーラがからデータを移行します `etcd` キーバリューデータストアと対応するCRDオブジェクトの作成。
- 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/netapp.io/trident` プロビジョニング担当者のフィールドに入力します。ストレージクラス名がKubernetesの名前と一致していること `StorageClass` 表すオブジェクト。

Kubernetesでは、これらのオブジェクトはKubernetesのときに自動的に作成されます `StorageClass` Tridentをプロビジョニングツールとして使用していることが登録されます。

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環境では、新しいKubernetesを登録するときにKubernetes環境が作成されることを想定しています `StorageClass` オブジェクト。

Trident バックエンドオブジェクト

バックエンドとは、 Trident がボリュームをプロビジョニングする際にストレージプロバイダを表します。 1 つの Trident インスタンスであらゆる数のバックエンドを管理できます。

これは、自分で作成および管理する 2 つのオブジェクトタイプのうちの 1 つです。もう1つはKubernetesです `StorageClass` オブジェクト。

これらのオブジェクトの作成方法の詳細については、を参照してください "バックエンドの設定"。

Trident `StoragePool` オブジェクト

ストレージプールは、各バックエンドでのプロビジョニングに使用できる個別の場所を表します。ONTAP の場合、これらは SVM 内のアグリゲートに対応します。NetApp HCI / SolidFire では、管理者が指定した QoS 帯域に対応します。Cloud Volumes Service の場合、これらはクラウドプロバイダのリージョンに対応します。各ストレージプールには、パフォーマンス特性とデータ保護特性を定義するストレージ属性があります。

他のオブジェクトとは異なり、ストレージプールの候補は常に自動的に検出されて管理されます。

Trident `Volume` オブジェクト

ボリュームは、 NFS 共有や iSCSI LUN などのバックエンドエンドエンドエンドポイントで構成される、プロビジョニングの基本単位です。Kubernetesでは、これらはに直接対応します `PersistentVolumes`。ボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前
splitOnClone	文字列	いいえ	ONTAP （ NAS 、 SAN ）：クローンを親からスプリットします
Snapshot ポリシー	文字列	いいえ	ONTAP - * ：使用する Snapshot ポリシー
Snapshot リザーブ	文字列	いいえ	ONTAP - * ： Snapshot 用にリザーブされているボリュームの割合
エクスポートポリシー	文字列	いいえ	ONTAP-NAS* ：使用するエクスポートポリシー
snapshotDirectory の略	ブール値	いいえ	ONTAP-NAS* ： Snapshot ディレクトリが表示されているかどうか
unixPermissions	文字列	いいえ	ONTAP-NAS* ：最初の UNIX 権限
ブロックサイズ	文字列	いいえ	SolidFire - * ：ブロック / セクターサイズ
ファイルシステム	文字列	いいえ	ファイルシステムのタイプ

Tridentが生成 `internalName` ボリュームを作成する場合。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（デフォルトのプレフィックス）を先頭に追加します `trident` またはバックエンド構成内のプレフィックス）をボリューム名に変更して、形式の名前を指定します `<prefix>-<volume-name>`。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 バックエンドの場合、ハイフンをアンダースコアに置き換えます（内部名はになります）

<prefix>_<volume-name>）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム構成を使用してREST APIを使用してボリュームを直接プロビジョニングできますが、Kubernetes環境ではほとんどのユーザが標準のKubernetesを使用することを想定しています PersistentVolumeClaim メソッドプロビジョニングの一環としてTridentがこのボリュームオブジェクトを自動的に作成
プロセス：

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、これらはに直接対応します VolumeSnapshotContent オブジェクト。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。

各 Snapshot オブジェクトには、次のプロパティが含まれます。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン（「 1 」）
名前	文字列	はい。	Trident Snapshot オブジェクトの名前
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetesを導入したとき VolumeSnapshot オブジェクト要求が作成されると、Tridentはバッキングストレージシステムにsnapshotオブジェクトを作成することで機能します。。 internalName このSnapshotオブジェクトのプレフィックスを組み合わせると、が生成されます snapshot- を使用 UID の VolumeSnapshot オブジェクト（例： snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660）。 volumeName および volumeInternalName 裏付けの詳細を取得することで入力されます。
ボリューム：

Astra Trident ResourceQuota オブジェクト

Tridentのデーモンは、を消費します system-node-critical 優先度クラス：Kubernetesで最も高い優先度クラスです。Astra Tridentは、ノードの正常なシャットダウン中にボリュームを識別してクリーンアップし、Tridentのデミスタポッドがリソースの負荷が高いクラスタでより低い優先度でワークロードをプリエンプトできるようにします。

そのために、Astra Tridentはを採用しています ResourceQuota Tridentのデミスタに対する「システムノー

ドクリティカル」の優先クラスを満たすことを保証するオブジェクト。導入とデマ作用の開始前に、Astra Tridentがを探します `ResourceQuota` オブジェクトを検出し、検出されない場合は適用します。

デフォルトのリソースクォータ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、を生成できます `custom.yaml` またはを設定します `ResourceQuota` Helmチャートを使用するオブジェクト。

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、のあとにインストールが失敗する場合があります `ResourceQuota` オブジェクトが作成されました。最初に実行してください "アンインストール中です" を再インストールします。

うまくいかない場合は、を手動で削除します `ResourceQuota` オブジェクト。

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、Astra Tridentを削除できます `ResourceQuota` 次のコマンドを使用したオブジェクトの削除：

```
kubectl delete quota trident-csi -n trident
```

tridentctl コマンドとオプション

。 "Trident インストーラバンドル" コマンドラインユーティリティを搭載しています。`tridentctl`Astra Tridentに簡単にアクセスできます。十分な権限を持つ Kubernetes ユーザは、このロールを使用して Astra Trident をインストールしたり、 Astra Trident ポッドが含まれるネームスペースを直接管理したりできます。

使用可能なコマンドとオプション

使用方法については、を実行してください `tridentctl --help`。

使用可能なコマンドとグローバルオプションは次のとおりです。

```
Usage:
  tridentctl [command]
```

使用可能なコマンド：

- `create`：Astra Tridentにリソースを追加
- `delete`：Astra Tridentから1つ以上のリソースを削除
- `get`：Astra Tridentから1つ以上のリソースを入手
- `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を表で印刷
- `import`：既存のリソースをAstra Tridentにインポート
- `install`：Astra Tridentをインストール
- `logs`：Astra Tridentからログを印刷
- `send`：Astra Tridentからリソースを送信
- `uninstall`：TridentからAstraをアンインストール。
- `update`：Astra Tridentのリソースを変更
- `upgrade`：Astra Tridentのリソースをアップグレード
- `version`：Astra Tridentのバージョンを印刷

フラグ：

- `-d, --debug`:デバッグ出力。
- `-h, --help`:のヘルプ `tridentctl`。
- `-n, --namespace string`: Astra Tridentのネームスペースの導入。
- `-o, --output string`:出力形式。JSON の 1 つ | yaml | name | wide | ps （デフォルト）。
- `-s, --server string`：Astra Trident RESTインターフェイスのアドレス/ポート。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）のみをリスンして処理するように設定できます。

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。

create

を実行します `create` Astra Tridentにリソースを追加するコマンド。

```
Usage:
  tridentctl create [option]
```

使用可能なオプション：
`backend`：Astra Tridentにバックエンドを追加

delete

を実行できます `delete` コマンドを使用して、Astra Tridentから1つ以上のリソースを削除します。

```
Usage:
  tridentctl delete [option]
```

使用可能なオプション：

- `backend`：Tridentから1つ以上のストレージバックエンドを削除
- `snapshot`：Astra Tridentから1つ以上のボリュームSnapshotを削除
- `storageclass`：Astra Tridentから1つ以上のストレージクラスを削除
- `volume`：Astra Tridentから1つ以上のストレージボリュームを削除

get

を実行できます `get` Astra Tridentから1つ以上のリソースを取得するためのコマンドです。

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

使用可能なオプション：

- `backend`：Tridentから1つ以上のストレージバックエンドを取得
- `snapshot`：Astra Tridentから1つ以上のスナップショットを取得
- `storageclass`：Astra Tridentから1つ以上のストレージクラスを取得
- `volume`：Astra Tridentから1つ以上のボリュームを取得

`volume` 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。

images

を実行できます `images` Astra Tridentが必要とするコンテナイメージの表を印刷するためのフラグ。

```
Usage:
  tridentctl images [flags]
```

フラグ：
* `-h, --help`: Help for images.
* -v ,--k8s-version string`：Kubernetesクラスタのセマンティックバージョン。

import volume

を実行できます `import volume` コマンドを使用して、既存のボリュームをAstra Tridentにインポートします。

```
Usage:
  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。

install

を実行できます `install` Astra Tridentのインストールにフラグを付けます。

```
Usage:
  tridentctl install [flags]
```

フラグ：

- `--autosupport-image string`：AutoSupport テレメトリのコンテナイメージ（デフォルトは「NetApp / Trident autosupport：20.07.0」）。
- `--autosupport-proxy string`：AutoSupport テレメトリを送信するプロキシのアドレス/ポート。
- `--csi`：CSI Tridentをインストールします（Kubernetes 1.13のみを上書きします。機能ゲートが必要です）。

- `--enable-node-prep：ノードに必要なパッケージをインストールします。
- `--generate-custom-yaml:インストールを行わずにYAMLファイルを生成します。
- `-h, --help:インストールのヘルプ。
- `--http-request-timeout：TridentコントローラのREST APIのHTTP要求タイムアウトを上書きします（デフォルトは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処理のタイムアウト（デフォルトは3分0）。
- `--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: Astra Tridentのログ形式(テキスト、JSON](デフォルトは「text」)。
- `--pv string: Astra Tridentが使用するレガシーPVの名前は、存在しないことを確認します(デフォルトは"trident")。
- `--pvc string：Astra Tridentで使用されている従来のPVCの名前。このPVCが存在しないことを確認します（デフォルトは「trident」）。
- `--silence-autosupport：AutoSupport バンドルを自動的にネットアップに送信しない（デフォルトはtrue）。
- `--silent:インストール中は、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Astra Tridentのイメージ
- `--use-custom-yaml: setupディレクトリに存在する既存のYAMLファイルを使用します。
- `--use-ipv6：Astra Tridentの通信にIPv6を使用

logs

を実行できます `logs` Astra Tridentからログを印刷するためのフラグ。

```
Usage:
  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：Astra Tridentのログが表示されます。trident | auto | trident-operator | all （デフォルトは「 auto 」）のいずれかです。
- `--node string：ノードポッドログの収集元のKubernetesノード名。
- `-p, --previous:以前のコンテナインスタンスのログが存在する場合は、それを取得します。
- `--sidecars:サイドカーコンテナのログを取得します。

send

を実行できます `send` Astra Tridentからリソースを送信するコマンド。

```
Usage:
  tridentctl send [option]
```

使用可能なオプション：
`autosupport`：ネットアップにAutoSupport アーカイブを送信します。

uninstall

を実行できます `uninstall` Astra Tridentをアンインストールするためのフラグ。

```
Usage:
  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。

update

を実行できます `update` Astra Tridentのリソースを変更するコマンド。

```
Usage:
  tridentctl update [option]
```

使用可能なオプション：
`backend`：Astra Tridentのバックエンドを更新。

upgrade

を実行できます `upgrade` Astra Tridentのリソースをアップグレードするためのコマンド。

```
Usage:
tridentctl upgrade [option]
```

使用可能なオプション：
`volume`：1つ以上の永続ボリュームをNFS/iSCSIからCSIにアップグレードします。

`version`

を実行できます `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

```
Usage:
  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。

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、Astra 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、Astra Tridentによって設定された権限の詳細を説明します。

PSSは、Podセキュリティポリシー（PSP）に代わるものです。PSPはKubernetes v1.21で廃止され、v1.25で削除されます。詳細については、を参照してください "Kubernetes：セキュリティ"。

必須の**Kubernetes Security Context**と関連フィールド

アクセス権	説明
権限があります	CSIでは、マウントポイントが双方向である必要があります。つまり、Tridentノードポッドで特権コンテナを実行する必要があります。詳細については、を参照してください "Kubernetes：マウントの伝播"。
ホストネットワーク	iSCSIデーモンに必要です。 `iscsiadm` 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	開始する必要があります `rpc-statd` NFSの場合：Astra Tridentがホストプロセスを照会して、状況を特定 `rpc-statd` を実行してからNFSボリュームをマウントしてください。

アクセス権	説明

機能	。 `SYS_ADMIN` この機能は、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は特権コンテナに次の機能を設定します。
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccom	Seccompプロファイルは、権限のあるコンテナでは常に「制限なし」なので、Astra Tridentでは有効にできません。
SELinux	OpenShiftでは、特権のあるコンテナがで実行されます `spc_t` （「スーパー特権コンテナ」）ドメインおよび非特権コンテナは、で実行されます `container_t` ドメイン：オン `containerd`を使用 `container-selinux` インストールすると、すべてのコンテナがで実行されます `spc_t` domain 。SELinuxは無効になります。そのため、Astra Tridentは機能しません `seLinuxOptions` コンテナへ。
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.kubernetes.io/enforce`
 	
`pod-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: privileged`
		
		`enforce-version: <version of the current cluster or highest version of PSS tested.>`

名前空間ラベルを変更すると、ポッドがスケジュールされず、「Error creating：…」または「Warning：trident-csi-…」が表示される場合があります。その場合は、のネームスペースラベルを確認してください `privileged` が変更されました。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。	`true`
`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

フィールド	説明	デフォルト
allowedCapabilities	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	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	Tridentノードポッドでノードのルートファイルシステムがマウントされるため、このリストを設定してもメリットはありません。	空です
allowedProcMountTypes	Tridentでは使用していません ProcMountTypes。	空です
allowedUnsafeSysctls	Tridentでは安全でないリソースは不要です sysctls。	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。	false
forbiddenSysctls	いいえ sysctls 許可されています。	空です
fsGroup	Tridentコンテナはrootとして実行されます。	RunAsAny
hostIPC	NFSボリュームをマウントするには、ホストIPCがと通信する必要があります nfsd	true
hostNetwork	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。	true
hostPID	ホストPIDが必要かどうかを確認します rpc-statd ノードで実行されている。	true
hostPorts	Tridentはホストポートを使用しません。	空です
privileged	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。	true
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。	false

フィールド	説明	デフォルト
requiredDropCapabilities	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は使用しません RuntimeClasses。	空です
seLinux	Tridentが設定されていません seLinuxOptions 現在のところ、コンテナの実行時間とKubernetesのディストリビューションでのSELinuxの処理に違いがあるためです。	空です
supplementalGroups	Tridentコンテナはrootとして実行されます。	RunAsAny
volumes	Tridentポッドには、こ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。	true
allowHostIPC	NFSボリュームをマウントするには、ホストIPCがと通信する必要があります nfsd。	true
allowHostNetwork	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。	true
allowHostPID	ホストPIDが必要かどうかを確認します rpc-statd ノードで実行されている。	true
allowHostPorts	Tridentはホストポートを使用しません。	false
allowPrivilegeEscalation	特権コンテナは、特権昇格を許可する必要があります。	true
allowPrivilegedContainer	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。	true

ラベル	説明	デフォルト
`allowedUnsafeSysctls`	Tridentでは安全でないリソースは不要です `sysctls`。	`none`
`allowedCapabilities`	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`	Tridentコンテナはrootとして実行されます。	`RunAsAny`
`groups`	このSCCはTridentに固有で、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。	`false`
`requiredDropCapabilities`	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が設定されていません `seLinuxOptions` 現在のところ、コンテナの実行時間とKubernetesのディストリビューションでのSELinuxの処理に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`	Tridentコンテナはrootとして実行されます。	`RunAsAny`
`users`	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
`volumes`	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

ネットアップの Astra Trident 向けソフトウェアで使用されているサードパーティの著作権とライセンスは、の各リリースの通知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。