Astra Trident 22.01 のドキュメント

Astra Trident

NetApp
April 16, 2024

This PDF was generated from https://docs.netapp.com/ja-jp/trident-2201/index.html on April 16, 2024.
Always check docs.netapp.com for the latest.

目次

Astra Trident 22.01 のドキュメント 1
リリースノート 2
22.01.1の新機能 2
22.01.0 の変更点（ 22.10.1 以降） 2
Astra Trident 21.10.1 の変更点 3
21.10.0 の変更点（ Astra Trident 21.07 以降） 3
既知の問題 4
詳細については、こちらをご覧ください 5
概念 6
Astra Trident の概要 6
ONTAP ドライバ 7
プロビジョニング 8
ボリューム Snapshot 8
仮想ストレージプール 9
ボリュームアクセスグループ 10
はじめに 12
ぜひお試しください 12
要件 12
導入の概要 17
Trident オペレータとともに導入 20
tridentctl を使用して導入します 28
次の手順 31
Trident で Astra を管理 37
Astra Trident をアップグレード 37
オペレータにアップグレードしてください 39
tridentctl を使用してアップグレードします 46
Astra Trident をアンインストール 50
Trident をダウングレード 52
Astra Trident を使用 56
バックエンドを設定 56
kubectl を使用してバックエンドを作成します 124
kubectl を使用してバックエンド管理を実行します 131
tridentctl を使用してバックエンド管理を実行します 132
バックエンド管理オプション間を移動します 134
ストレージクラスを管理する 140
ボリューム操作を実行する 142
ワーカーノードを準備します 167
ワーカーノードの自動準備 171
Astra Trident を監視 171

Trident for Docker が必要です 176
導入の前提条件 176
Astra Trident を導入 179
Astra Trident をアップグレードまたはアンインストールする 183
ボリュームを操作します 185
ログを収集します 195
複数の Astra Trident インスタンスを管理 196
ストレージ構成オプション 197
既知の問題および制限事項 213
よくある質問 215
一般的な質問 215
Kubernetes クラスタに Astra Trident をインストールして使用 215
トラブルシューティングとサポート 217
Astra Trident をアップグレード 218
バックエンドとボリュームを管理 218
サポート 224
トラブルシューティング 225
全般的なトラブルシューティング 225
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 227
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` 229
ベストプラクティスと推奨事項 230
導入 230
ストレージ構成 230
Astra Trident を統合 237
データ保護 248
セキュリティ 253
参照 255
Astra Trident ポート 255
Astra Trident REST API 255
コマンドラインオプション 256
ネットアップの製品が Kubernetes と統合されます 257
Kubernetes オブジェクトと Trident オブジェクト 258
tridentctl コマンドとオプション 270
以前のバージョンのドキュメント 276
法的通知 277
著作権 277
商標 277
特許 277
プライバシーポリシー 277
オープンソース 277

Astra Trident 22.01 のドキュメント

リリースノート

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。

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

22.01.1の新機能

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"。

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

の修正

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

22.01.0 の変更点（ 22.10.1 以降）

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 のサポートが追加されました。

- Trident Operator または Helm 経由でインストールした場合、 Trident ポッドのスケジュールオプションを追加します。（"GitHub 問題 #651"）

- GCP ドライバでリージョン間のボリュームを許可します。（"GitHub 問題 #633"）
- 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

Astra Trident 21.10.1 の変更点

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

の修正

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

21.10.0 の変更点（ Astra Trident 21.07 以降）

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：
  - スナップショットを作成するときに 'ONTAP-NAS' および 'ONTAP-NAS-flexgroup ドライバ（ GitHub 問題 645 ）を使用して ' ボリュームの使用済み領域を最小リストアサイズとして返します
  - ボリュームのサイズ変更後に 'Failed to expand filesystem エラーがログに記録された問題を修正しました (GitHub 問題 560)
  - POD が「 Terminating 」状態で停止する可能性がある固定問題（ GitHub 問題 572 ）。
  - 「 ONTAP-SAN-エコノミー 」問題がスナップショット FlexVol （ GitHub 533 ）でいっぱいになる場合があるという問題を修正しました。
  - 異なるイメージを持つ固定カスタム YAML インストーラ問題（ GitHub 問題 613 ）。
  - Snapshot サイズの計算方法を固定（ GitHub 問題 611 ）。

- 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

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

実験的な機能強化

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

既知の問題

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

- Astra Trident は ' ストレージクラスに fsType が指定されていないボリュームに対して ' ブランクの fsType(`fsType="") を適用するようになりましたKubernetes 1.17 以降を使用する場合、 Trident は NFS ボリュームに空の「 fsType 」を提供できます。iSCSI ボリュームの場合 ' セキュリティコンテキストを使用して fsGroup を実行するときに 'fsType' を StorageClass 上に設定する必要があります
- 複数の Astra Trident インスタンス間でバックエンドを使用する場合、各バックエンド構成ファイルの ONTAP バックエンドに異なる「 toragePrefix 」値を指定するか、 SolidFire バックエンドに異なる「 tenantname 」を使用する必要があります。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。「 toragePrefix 」または「 tenantname 」が異なる場合は、同じバックエンド上に作成されたボリュームに名前の衝突が発生する可能性があります。
- Astra Trident をインストールするときに (tridentctl または Trident Operator を使用) 、 tridentctl を使用して Astra Trident を管理するときは、「 KUBECONFIG 」環境変数が設定されていることを確認する必要があります。これは 'tridentctl が機能するべき Kubernetes クラスタを示すために必要です複数の Kubernetes 環境で作業する場合は、「 KBECONFIG 」ファイルが正確にソースされていることを確認する必要があります。
- iSCSI PVS のオンラインスペース再生を実行するには、作業者ノード上の基盤となる OS がボリュームにマウントオプションを渡す必要があります。これは、「 discard 」を必要とする RHEL/RedHat CoreOS インスタンスに当てはまります "マウントオプション"; discard mountOption がに含まれていることを確認します "d4b9b9554fd820f43eae492d33e41167" オンラインブロックの破棄をサポートするため。

- 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。
- Trident がオフラインのときに Astra Trident ベースの「 torageClass 」オブジェクトが Kubernetes から削除された場合、 Astra Trident は、対応するストレージクラスをオンラインに戻ってもデータベースから削除しません。これらのストレージクラスは、「 tridentctl 」または REST API を使用して削除してください。
- 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。tridentctl または REST API を使用してボリュームを削除してください。
- 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、バックエンド定義内の角かっこ内に「 managementlif 」と「 datalif 」を指定する必要があります。例えば、「 [fd20 ： 8b1e ： b258 ： 2000 ： f816 ： 3eff ： feec ： 0]`` 」のようになります。
- OpenShift 4.5 で「 olidfire-san 」ドライバを使用する場合は、基になるワーカーノードが CHAP 認証アルゴリズムとして MD5 を使用していることを確認します。

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"

概念

Astra Trident の概要

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。

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 ）、 Astra データストア、 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 データストアのドキュメント"
- "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。各ドライバーがボリュームの作成、アクセス制御、機能をどのように処理するかについて、詳細をご覧ください。

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	ファイルシステム	RWO 、 RWX 、 ROX	"" 、 NFS です
「 ONTAP - NAS - エコノミー」	NFS	ファイルシステム	RWO 、 RWX 、 ROX	"" 、 NFS です
「 ONTAP-NAS-flexgroup 」	NFS	ファイルシステム	RWO 、 RWX 、 ROX	"" 、 NFS です
「 ontap - 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
「 ontap - san 」	iSCSI	ファイルシステム	RWO 、 ROX	「 xfs 」、「 ext3 」、「 ext4 」
「 ONTAP - SAN - エコノミー」	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
「 ONTAP - SAN - エコノミー」	iSCSI	ファイルシステム	RWO 、 ROX	「 xfs 」、「 ext3 」、「 ext4 」

ONTAP バックエンドは、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て認証されます。既存のバックエンドを更新して 'tridentctl update backend' を使用して認証モードから別の認証モードに移行できます

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。

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 'toragePools’additionalStoragePools'`excludeStoragePools' リストの両方が必要ですストレージクラスを作成すると、 Trident はバックエンドごとに提供される属性とプールを、ストレージクラスから要求された属性とプールと比較します。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、属性がストレージクラスの要求された属性のすべてまたは一部を満たしていない場合でも、 Astra Trident は、そのセットに「 additionalStoragePools 」リストにリストされているすべてのストレージプールを追加します。ストレージ・クラスのストレージ・プールを上書きして ' ストレージ・クラスに使用しないようにするには 'excludeStoragePools' リストを使用する必要があります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。

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。

- 「 ONTAP-NAS」 、「 ONTAP-SAN」 、「 GCP-CVS 」、「 azure-NetApp-files 」ドライバの場合、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。
- 「 ONTAP-NAS-flexgroup 」ドライバの場合、各永続ボリューム（ PV ）は FlexGroup にマッピングされます。その結果、ボリューム 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。
- 「 ONTAP-SAN-エコノミー 」ドライバの場合、 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。
- 「 olidfire-SAN' ドライバの場合、各 PV は NetApp Element ソフトウェア / NetApp HCI クラスタ上に作成

された LUN にマッピングされます。ボリューム Snapshot は、基盤となる LUN の Element Snapshot で表されます。これらの Snapshot はポイントインタイムコピーであり、消費するシステムリソースとスペースはごくわずかです。

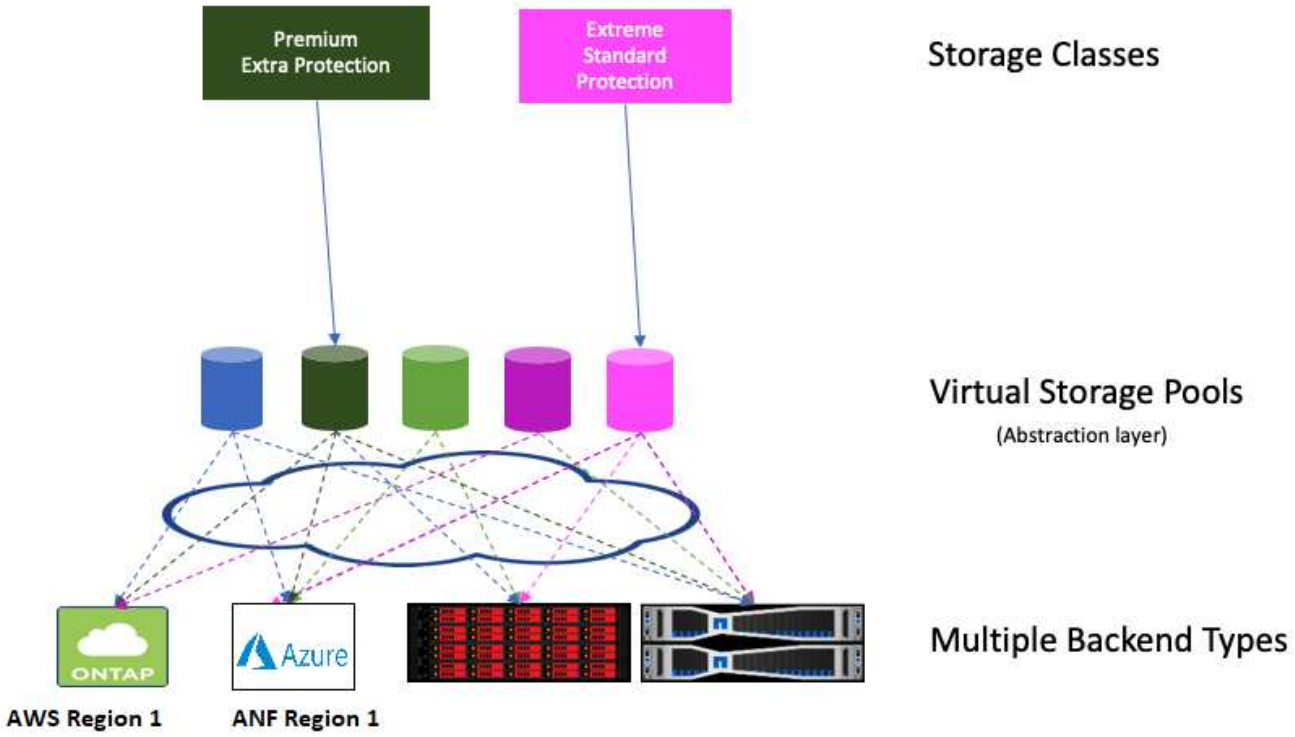
- ONTAP スナップショットは 'ONTAP-NAS' および「 ONTAP-SAN' 」ドライバと連携して動作する場合 ' FlexVol のポイント・イン・タイム・コピーであり ' FlexVol 自体の領域を消費しますその結果、ボリューム内の書き込み可能なスペースが、 Snapshot の作成やスケジュール設定にかかる時間を短縮できます。この問題に対処する簡単な方法の 1 つは、 Kubernetes を使用してサイズを変更することでボリュームを拡張することです。もう 1 つの方法は、不要になった Snapshot を削除することです。Kubernetes で作成されたボリューム Snapshot 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。'olidfire-SAN' ドライバは ' スナップショットから PVS を作成するために Element ソフトウェアボリュームクローンを使用しますここで、 Element Snapshot からクローンを作成します。

仮想ストレージプール

仮想ストレージプールは、 Astra Trident の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。管理者は ' どの物理バックエンド ' バックエンドプール ' またはバックエンドタイプを ' 目的の基準を満たすために使用するかを ' 「 torageClass 」で指定することなく ' バックエンドに依存しない共通の方法で ' 各バックエンドのロケーション ' パフォーマンス ' 保護などの側面を定義できます

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ストレージ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）。

> 仮想ストレージ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でください。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。

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公開されず、「 torageClasses 」でのマッチングには使用できません。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。

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。

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、構成に 1 つ以上のアクセスグループ ID を指定しない限り、「 trident 」というアクセスグループが検索されます。

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"。

デフォルトの trident' アクセスグループを使用している構成から ' 他のアクセスグループを使用している構成に変更する場合は ' リストに trident' アクセスグループの ID を含めます

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"。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

サポートされるフロントエンド、バックエンド、およびホスト構成を確認することから始めましょう。

> Trident が使用するポートについては、を参照してください "こちらをご覧ください"。

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 は、ベアメタル 1.8 、 1.9 、 1.10 上に実装されています
- Kubernetes 1.17 以降（最新： 1.23 ）
- Mirantis Kubernetes Engine 3.4
- OpenShift 4.7 、 4.8 、 4.9

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 は、ベアメタル 1.8 、 1.9 、 1.10 上に実装されています
- Kubernetes 1.17 以降（最新： 1.23 ）
- OpenShift 4.7 、 4.8 、 4.9

> Red Hat OpenShift Container Platform のユーザは、 4.1.8 よりも前のバージョンを使用している場合、 initiatorname.iscsi ファイルが空白になる可能性があります。これは RedHat によって識別されているバグで、 OpenShift 4.1.8 で修正されています。を参照してください "バグ修正のお知らせ"。ネットアップでは、 OpenShift 4.6.8 以降で Astra Trident を使用することを推奨しています。

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。

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
- Astra データストア
- Cloud Volumes ONTAP
- Cloud Volumes Service for GCP
- FAS/AFF / Select 9.3 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	1.17 以降	いいえ
ボリューム Snapshot	1.17 以降	いいえ
ボリューム Snapshot からの PVC	1.17 以降	いいえ
iSCSI PV のサイズ変更	1.17 以降	いいえ
ONTAP 双方向 CHAP	1.17 以降	いいえ
動的エクスポートポリシー	1.17 以降	いいえ
Trident のオペレータ	1.17 以降	いいえ
自動ワーカーノード準備（ベータ版）	1.17 以降	いいえ
CSI トポロジ	1.17 以降	いいえ

テスト済みのホストオペレーティングシステム

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。

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るわけではありませんが、次の Linux ディストリビューションが動作することはわかっています。

- OpenShift Container Platform でサポートされている Red Hat CoreOS （ RHCOS ）バージョン
- RHEL または CentOS 7.4 以降
- Ubuntu 18.04 以降

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

ホストの設定

使用しているバックエンドによっては、 NFS や iSCSI のユーティリティをクラスタ内のすべてのワーカーにインストールする必要があります。を参照してください "こちらをご覧ください" を参照してください。

ストレージシステムの構成：

Trident を使用するには、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、一部の変更が必要になることがあります。を参照してください "こちらをご覧ください" を参照してください。

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。tridentctl images コマンドを使用して ' 必要なコンテナイメージのリストを確認します

Kubernetes のバージョン	コンテナイメージ
v1.17.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • k81.gcr.io/sig-storage/csi-Provisioner ： v2.2.2 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）

Kubernetes のバージョン	コンテナイメージ
v1.18.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • k81.gcr.io/sig-storage/csi-Provisioner ： v2.2.2 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）
v1.19.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • k81.gcr.io/sig-storage/csi-Provisioner ： v2.2.2 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）
v1.20.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • K83.GCR.IO/sig-storage/csi-Provisioner ： v3.1.0 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）

Kubernetes のバージョン	コンテナイメージ
v1.21.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • K83.GCR.IO/sig-storage/csi-Provisioner ： v3.1.0 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）
v1.22.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • K83.GCR.IO/sig-storage/csi-Provisioner ： v3.1.0 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）
v1.3.0	• NetApp / Trident ： 22.01.1 • NetApp / Trident -autosupport ： 22.01. • K83.GCR.IO/sig-storage/csi-Provisioner ： v3.1.0 • k83.GCR.IO/sig-storage/csi-attacher: v3.4.0 • k81.gcr.io/sig-storage/csi-resizer ： v1.3.0 • k83.gcr.io/sig-storage/csi-snapshotter ： v3.0.3 • k8s 。 gcr.io/sig-storage/csi-node-driver-registrar: v2.4.0 • NetApp/trident-operator ： 22.01.1 （オプション）

Kubernetes バージョン 1.20 以降では、検証済みの「 k8es.cr.io/sig-storage/csi-snapshotter ： v4.x 」イメージを使用します。これは、「 v1' バージョンが「 volumesnapshotes.snapshot.storage.k88.io`CRD 」に対応している場合にのみ使用します。v1beta` のバージョンが v1beta` のバージョンの有無に関わらず CRD にサービスを提供している場合は、検証済みの「 k83.gcr.io/sig-storage/csi-snapshotter: v3.x' 」イメージを使用します。

導入の概要

Trident の導入には、 Trident のオペレータまたは tridentctl を使用できます。

導入方法を選択します

使用する導入方法を決定するには、次の点を考慮してください。

Trident のオペレータが必要な理由

。 "Trident オペレータ" は、 Astra Trident のリソースを動的に管理し、セットアップフェーズを自動化する優れた方法です。いくつかの前提条件を満たす必要があります。を参照してください "要件"。

Trident オペレータには、以下に示すような利点があります。

自己回復機能

Trident の Astra 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、問題に対処する手段を積極的に講じることができます。オペレータが配備として設定されると 'trident-operator-<generated-id>'POD が作成されますこのポッドは 'TridentOrchestrator CR と Astra Trident のインストールを関連付けて ' 常にアクティブな TridentOrchestrator が 1 つだけになるようにしますつまり、オペレータは、 Astra Trident のインスタンスがクラスタ内に 1 つしかないことを確認し、セットアップを制御して、インストールがべきでないことを確認します。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。

既存のインストール環境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。「 TridentOrchestrator 」 CR を編集するだけで、インストールを更新できます。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、「 TridentOrchestrator 」に「 pec.debug` 」を「 true 」に設定するパッチを適用します。

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

TridentOrchestrator が更新されると、オペレータは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 のアップグレードを自動的に処理

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。

Helm を使用する必要があるのはなぜですか？

Helm を使用して管理している他のアプリケーションが Astra Trident 21.01 以降である場合は、 Helm を使用して導入を管理することもできます。

いつを使用すればよいか `tridenctl`？

既存の導入環境をに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場合は、のを参照してください "Tridentctl"。これは、従来の方法であった Astra Trident を導入する方法です。

導入方法間での移動に関する考慮事項

導入方法を切り替える必要があるシナリオを想像するのは難しいことではありません。tridentctl 配備からオペレータベースの配備に移行する前に ' またはその逆を行う前に ' 次の点を考慮してください

- Astra Trident のアンインストールには、常に同じ方法を使用します。tridentctl を使用して導入した場合は 'tridentctl' バイナリの適切なバージョンを使用して Astra Trident をアンインストールする必要があります同様に、オペレータとともに展開する場合は、「 TridentOrchestrator 」 CR を編集し、「 PEC.uninstall=true 」を設定して Astra Trident をアンインストールする必要があります。
- Astra Trident を削除して tridentctl を使用して Astra Trident を導入するオペレータベースの導入環境の場合は、最初に「 TridentOrchestrator 」を編集し、「 PEC.uninstall=true 」を設定して Astra Trident をアンインストールする必要があります。次に 'TridentOrchestrator とオペレータの配備を削除します次に 'tridentctl を使用してインストールできます
- 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。tridentctl ベースの配備を使用している場合は 'Helm ベースの配備を問題なく使用できます

導入モードを理解する

Trident を導入する方法は 3 種類あります。

標準的な導入

Trident を Kubernetes クラスタに導入すると、 Astra Trident インストーラで次の 2 つの作業を実行できます。

- インターネット経由でコンテナイメージを取得しています
- 導入環境と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する。

このような標準的な導入は、次の 2 つの方法で実行できます。

- tridentctl install を使用
- Trident 演算子を使用する。Trident オペレータは、手動で導入することも、 Helm を使用して導入することもできます。

このインストールモードは、 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。

オフラインでの導入

エアギャップ導入を実行するには 'tridentctl install を呼び出すときに --image-registry フラグを使用して ' プライベートイメージレジストリを参照しますTrident 演算子を使用して展開する場合は、「 TridentOrchestrator 」で「 PEC.imageRegistry 」を指定することもできます。このレジストリにはが含まれている必要があります "Trident の画像"、 "Trident AutoSupport の画像"および CSI のサイドカーイメージ（ Kubernetes バージョンで必要な場合）

導入環境をカスタマイズするには 'tridentctl' を使用して Trident のリソースのマニフェストを生成します導入、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。

導入環境のカスタマイズの詳細については、次のリンクを参照してください。

- "オペレータベースの展開をカスタマイズします"
- "85cda507040c129ee606d4d8df583b90"

> プライベート・イメージ・リポジトリを使用している場合は '1.17 より前のバージョンの Kubernetes には '/k8scsi' を ' プライベート・レジストリ URL の末尾には '1.17 より後のバージョンの Kubernetes には '/si' を追加する必要がありますtridentctl 配備にプライベートレジストリを使用する場合は '--trident-image' と --autosupport.image' を --image-registry' とともに使用する必要がありますTrident 演算子を使用して Astra Trident を導入する場合は、 Orchestrator CR にインストールパラメータに「 tridentImage 」と「 autosupportImage 」が含まれていることを確認してください。

リモート導入

次に、リモート導入プロセスの概要を示します。

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

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

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

Trident オペレータとともに導入

Trident のオペレータが、 Astra Trident を導入できます。Trident オペレータは、手動または Helm を使用して導入できます。

をまだ理解していない場合は、を参照してください "基本概念"今こそ、そのための絶好の機会です。

必要なもの

Astra Trident を導入するには、次の前提条件を満たしている必要があります。

- Kubernetes 1.17 以降を実行するサポート対象の Kubernetes クラスタに対するすべての権限が必要です。
- サポートされているネットアップストレージシステムを利用できるようにしておきます。
- すべての Kubernetes ワーカーノードからボリュームをマウントできます。
- 使用する Kubernetes クラスタを管理するために 'kubectl(OpenShift を使用している場合は 'OC) をインストールして構成した Linux ホストがあります
- 「 KUBECONFIG 」環境変数を、 Kubernetes クラスタ構成を指すように設定しておきます。
- を有効にしておきます "Astra Trident に必要な機能ゲート"。
- 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"。

それはすべてですか？最高！それでは始めましょう。

Helm を使用して Trident オペレータを導入します

Helm を使用して Trident オペレータを導入するには、以下の手順を実行します。

必要なもの

上記の前提条件に加え、 Helm を使用して Trident Operator を導入するには、次のものが必要です。

- Kubernetes 1.17 以降
- Helm バージョン 3

手順

1. Trident の Helm リポジトリを追加：

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

2. 「 helm install 」コマンドを使用して、展開の名前を指定します。次の例を参照してください。

```
helm install <release-name> netapp-trident/trident-operator --version
22.1.0 --namespace <trident-namespace>
```

> Trident の名前空間をまだ作成していない場合は '--create-namespace パラメータを helm install コマンドに追加できますHelm によってネームスペースが自動的に作成されます。

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。

- `--values]` （ または `'-f`）: オーバーライドを含む 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 ' デフォルト値の `debug` を変更するには ' 次のように --set` コマンドを実行します

```
$ helm install <name> netapp-trident/trident-operator --version 22.1.0
--set tridentDebug=true
```

Helm チャートの一部である `values] の .yaml ファイルには、キーのリストとデフォルト値が表示されます。

「 helm list 」は、名前、名前空間、グラフ、ステータスなど、インストールに関する詳細を表示します。 アプリケーションのバージョン、リビジョン番号など。

Trident オペレータを手動で導入

Trident のオペレータを手動で導入するには、以下の手順を実行します。

ステップ 1 ： Kubernetes クラスタを確認する

まず、 Linux ホストにログインして、 _working _ 、 "サポートされる Kubernetes クラスタ" に必要な権限があることを確認します。

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

Kubernetes のバージョンが 1.17 よりも新しいかどうかを確認するには、次のコマンドを実行します。

```
kubectl version
```

Kubernetes クラスタ管理者の権限があるかどうかを確認するには、次のコマンドを実行します。

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

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かどうかを確認するには、次のコマンドを実行します。

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

手順 **2** ：オペレータをダウンロードして設定します

> 21.01 以降、 Trident Operator はクラスタを対象とします。Trident オペレータを使用して Trident をインストールするには、「 TridentOrchestrator 」カスタムリソース定義（ CRD ）を作成し、その他のリソースを定義する必要があります。Astra Trident をインストールする前に、次の手順を実行してオペレータをセットアップする必要があります。

1. の最新バージョンをダウンロードします "Trident インストーラバンドル" _Downloads_section から抽出します

   ```
   wget https://github.com/NetApp/trident/releases/download/v21.04/trident-
   installer-21.04.tar.gz
   tar -xf trident-installer-21.04.tar.gz
   cd trident-installer
   ```

2. 適切な CRD マニフェストを使用して、「 TridentOrchestrator 」 CRD を作成します。次に、後で「 TridentOrchestrator 」カスタムリソースを作成して、オペレータによってインストールをインスタンス化します。

   次のコマンドを実行します。

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

3. 「 TridentOrchestrator 」 CRD が作成されたら、オペレータの展開に必要な次のリソースを作成します。

   - 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
   - 演算子自体

     Trident インストーラには、これらのリソースを定義するマニフェストが含まれています。デフォルトでは ' 演算子は trident' 名前空間に配置されます'trident' 名前空間が存在しない場合は ' 次のマニフェストを使用して名前空間を作成します

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

4. デフォルトの 'trident` 名前空間以外の名前空間に演算子を配備するには ``erviceaccount.yaml 'clusterrolebinding.yaml ' および `operator.yML' マニフェストを更新し 'bundle.yaml を生成する必要があります

   次のコマンドを実行して YAML マニフェストを更新し、「 customizization.yaml 」を使用して「 bundle.yaml 」を生成します。

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

   次のコマンドを実行してリソースを作成し、オペレータを配置します。

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

5. 展開後にオペレータのステータスを確認するには、次の手順を実行します。

```
$ kubectl get deployment -n <operator-namespace>
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator    1/1     1            1           3m

$ kubectl get pods -n <operator-namespace>
NAME                                READY   STATUS             RESTARTS
AGE
trident-operator-54cb664d-lnjxh     1/1     Running            0
3m
```

オペレータによる導入で、クラスタ内のいずれかのワーカーノードで実行されるポッドが正常に作成されます。

> 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**3**：作成 `TridentOrchestrator` **Trident**をインストール

これで、オペレータを使って Astra Trident をインストールする準備ができました。これには 'TridentOrchestrator を作成する必要がありますTrident インストーラには 'TridentOrchestrator を作成するための定義例が付属していますこれは 'trident' 名前空間にインストールされます

```
$ 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:
  Debug:     true
  Namespace: trident
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:21.04
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Enable Node Prep:          false
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:21.04.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v21.04.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

Trident オペレータは 'TridentOrchestrator 仕様の属性を使用して 'Astra Trident のインストール方法をカスタマイズできますを参照してください "Trident の導入をカスタマイズ"。

「 TridentOrchestrator 」のステータスは、インストールが成功したかどうかを示し、インストールされている Trident のバージョンを表示します。

ステータス	説明
インストール中です	オペレータは、この「 TridentOrchestrator 」 CR を使用して Astra Trident をインストールしています。
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	オペレータは 'stra Trident をアンインストールしていますこれは 'pec.uninstall=true だからです
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	オペレータは Astra Trident をインストール、パッチ適用、更新、またはアンインストールできませんでした。オペレータはこの状態からのリカバリを自動的に試みます。この状態が解消されない場合は、トラブルシューティングが必要です。
更新中です	オペレータが既存のインストールを更新しています。
エラー	「 TridentOrchestrator 」は使用されません。別のファイルがすでに存在します。

インストール中に 'TridentOrchestrator のステータスが Installing から Installed に変わります「失敗」ステータスが表示され、オペレータが自身で回復できない場合は、オペレータのログを確認する必要があります。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。

Astra Trident のインストールが完了しているかどうかを確認するには、作成したポッドを確認します。

```
$ kubectl get pod -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-7d466bf5c7-v4cpw        5/5     Running   0           1m
trident-csi-mr6zc                   2/2     Running   0           1m
trident-csi-xrp7w                   2/2     Running   0           1m
trident-csi-zh2jt                   2/2     Running   0           1m
trident-operator-766f7b8658-ldzsv   1/1     Running   0           3m
```

また 'tridentctl を使用して 'Astra Trident のバージョンを確認することもできます

```
$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.04.0        | 21.04.0        |
+----------------+----------------+
```

これで、バックエンドを作成できます。を参照してください "導入後のタスク"。

導入時の問題のトラブルシューティングについては、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。

Trident オペレータの環境をカスタマイズ

Trident オペレータは 'TridentOrchestrator 仕様の属性を使用して 'Astra Trident のインストール方法をカスタマイズできます

属性のリストについては、次の表を参照してください。

パラメータ	説明	デフォルト
「 namespace 」を参照してください	Astra Trident をインストールするネームスペース	デフォルト
「バグ」	Astra Trident のデバッグを有効にします	いいえ
「 IPv6 」	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
'ilenceAutosupport	AutoSupport バンドルをネットアップに自動的に送信しない	いいえ
enableNodePrep	ワーカーノードの依存関係を自動的に管理（ * beta * ）	いいえ
「 autosupportImage 」を参照してください	AutoSupport テレメトリのコンテナイメージ	「 NetApp/trident-autosupport ： 21.04.0 」
「 autosupportProxy 」と入力します	AutoSupport テレメトリを送信するプロキシのアドレス / ポート	"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 」のように入力します	インストールする Astra Trident イメージ	「 NetApp / Trident ： 21.04 」
「 imageRegistry 」と入力します	内部レジストリへのパス。形式は「 <registry fqdn>[:port][/ssubpath]] 」です	"k83.gcr.io/sig-storage (k8s 1.17+) または Qua.io/k8scsi"
「 kubeletDir 」を参照してください	ホスト上の kubelet ディレクトリへのパス	「 /var/lib/kubelet 」
「 wippeout 」	Astra Trident を完全に削除するために削除するリソースのリスト	
「 imagePullSecrets 」	内部レジストリからイメージをプルするシークレット	

パラメータ	説明	デフォルト
controller'PluginNodeSelector	Trident Controller CSI プラグインを実行しているポッドの追加ノードセレクタ。pod.spec.nodeSelector と同じ形式を使用します。	デフォルトはありません。オプションです
controllerPluginTolerations	Trident Controller CSI プラグインを実行しているポッドに対する許容値を上書きします。POD .spec.Toleratations と同じ形式を使用します。	デフォルトはありません。オプションです
「 nodePluginNodeSelector 」	Trident ノード CSI プラグインを実行しているポッドの追加ノードセレクタ。pod.spec.nodeSelector と同じ形式を使用します。	デフォルトはありません。オプションです
「 nodePluginTolerations` 」	Trident Node CSI プラグインを実行しているポッドに対する許容値を上書きします。POD .spec.Toleratations と同じ形式を使用します。	デフォルトはありません。オプションです

「 TridentOrchestrator 」では、どのネームスペース Astra Trident がにインストールされているかを示すために pec.namespace` が指定されています。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、「 TridentOrchestrator 」のステータスが「 Failed 」に変わります。Astra Trident は、ネームスペース間での移行を意図したものではありません。

自動ワーカーノードの前処理は、非本番環境でのみ使用することを目的とした * ベータ機能です。

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"。

「 TridentOrchestrator 」を定義するときに上記の属性を使用して、インストールをカスタマイズできます。次に例を示します。

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

次に、ノードセレクタを使用して Trident を導入する例を示します。

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

「 TridentOrchestrator 」引数で許可される範囲を超えてインストールをカスタマイズする場合は、「 tridentctl 」を使用して必要に応じて変更できるカスタム YAML マニフェストを生成することを検討してください。

tridentctl を使用して導入します

tridentctl を使用して Astra Trident を導入できます。

をまだ理解していない場合は、を参照してください "基本概念"今こそ、そのための絶好の機会です。

展開をカスタマイズするには、を参照してください "こちらをご覧ください"。

必要なもの

Astra Trident を導入するには、次の前提条件を満たしている必要があります。

- サポート対象の Kubernetes クラスタに対するすべての権限が必要です。
- サポートされているネットアップストレージシステムを利用できるようにしておきます。
- すべての Kubernetes ワーカーノードからボリュームをマウントできます。
- 使用する Kubernetes クラスタを管理するために 'kubectl(OpenShift を使用している場合は 'OC) をインストールして構成した Linux ホストがあります
- 「 KUBECONFIG 」環境変数を、 Kubernetes クラスタ構成を指すように設定しておきます。
- を有効にしておきます "Astra Trident に必要な機能ゲート"。
- 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"。

それはすべてですか？最高！それでは始めましょう。

導入環境のカスタマイズについては、を参照してください "こちらをご覧ください"。

ステップ 1 ： Kubernetes クラスタを確認する

まず、 Linux ホストにログインして、 _working _ 、 "サポートされる Kubernetes クラスタ" に必要な権限があることを確認します。

> OpenShift では、以降のすべての例で「 kubectl 」の代わりに「 OC 」を使用します。まず、「 OC login-u SYSTEM ： admin 」または「 OC login-u kube-admin 」を実行して、「 admin 」としてログインする必要があります。

Kubernetes のバージョンを確認するには、次のコマンドを実行します。

```
kubectl version
```

Kubernetes クラスタ管理者の権限があるかどうかを確認するには、次のコマンドを実行します。

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

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かどうかを確認するには、次のコマンドを実行します。

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

Kubernetes サーバのバージョンを確認します。このポートは、 Astra Trident のインストール時に使用します。

手順 2 ：インストーラをダウンロードして展開します

> Trident インストーラは 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 オブジェクトを構成し、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する CSI サイドカーを初期化します。

の最新バージョンをダウンロードできます "Trident インストーラバンドル" _Downloads_section から ' 抽出します

たとえば、最新バージョンが 21.07.1 の場合は、次のようになります。

```
wget https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
tar -xf trident-installer-21.07.1.tar.gz
cd trident-installer
```

手順 3 ： Astra Trident をインストールする

tridentctl install コマンドを実行して 'Astra Trident を目的の名前空間にインストールします

```
$ ./tridentctl install -n trident
....
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-csi-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=21.07.1
INFO Trident installation succeeded.
....
```

インストーラが完了すると、次のように表示されます。Kubernetes クラスタ内のノードの数によっては、ポッドをさらに確認することもできます。

```
$ kubectl get pod -n trident
NAME                            READY   STATUS    RESTARTS   AGE
trident-csi-679648bd45-cv2mx    4/4     Running   0          5m29s
trident-csi-vgc8n               2/2     Running   0          5m29s

$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.07.1        | 21.07.1        |
+----------------+----------------+
```

上記の例のような出力が表示された場合、この手順は完了していますが、 Astra Trident の設定はまだ完了していません。次の手順に進みます。を参照してください "導入後のタスク"。

ただし、インストーラが正常に完了しなかった場合、または * running * trident-csi-< generated id> が表示されない場合は、プラットフォームがインストールされていません。

導入時の問題のトラブルシューティングについては、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。

tridentctl 展開をカスタマイズします

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、 -trident-image' を使用してイメージ名を指定できます。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、 --image_registry' スイッチを使用してリポジトリの場所を指定することをお勧めします。この場合、形式は「 <registry fqdn>[:port] 」です。

Astra Trident が自動的にワーカーノードを構成するには、「 --enable-node-prep 」を使用します。この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"。

ワーカーノードの自動準備は * ベータ機能 * で、非本番環境でのみ使用できます。

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

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

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

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

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

次の手順

Astra Trident の導入が完了したら、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、ポッドで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。そのためには、必要なパラメータを含む「 backend.json 」ファイルを作成します。さまざまなバックエンドタイプのサンプル構成ファイルは 'sample-input' ディレクトリにあります

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。

最もシンプルなストレージクラスは ' インストーラに付属の 'sample -input/storage-class-csl.yamml.templ' ファイルに基づいていますこれは ' *backend_type_* をストレージドライバ名に置き換えます

```
./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 'kubectl' を使用して 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。

例については 'sample-input/pvc-basic-csI.yaml を参照してください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。次に '/usr/share/nginx/html' の下に PV をマウントする nginx ポッドを起動します

```
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 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。Astra Trident の新機能を活用するには、 1 年に 1 回以上アップグレードすることを推奨します。

> 5 つ先のリリースにアップグレードするには、複数の手順でアップグレードする必要があります。

アップグレード先のバージョンを確認します

- YY-1.mm' リリースから 'YY.MM' リリースにアップグレードできますたとえば、 19.07 以降（ 19.07.1 などのドットリリースを含む）から 20.07 への直接アップグレードを実行できます。
- 以前のリリースを使用している場合は、複数の手順からなるアップグレードを実行する必要があります。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 ' 次に示すように ' 複数ステップのアップグレード・プロセスを実行します
  - 最初のアップグレードは 18.07 から 19.07 へ。特定のアップグレード手順については、該当するリリースのドキュメントを参照してください。
  - その後 '19.07 から 20.07 にアップグレードします

> バージョン 19.04 以前のすべてのアップグレードでは、 Astra Trident のメタデータを独自の「etcd」 オブジェクトから CRD オブジェクトに移行する必要があります。アップグレードの仕組みについては、リリースのドキュメント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'SANボリュームに対してhttps://kubernetes.io/docs/tasks/configure-pod-container/security-context/[securityコンテキストを適用するための要件ですhttps://github.com/NetApp/trident/tree/master/trident-installer/sample-input[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"。

どのアップグレードパスを選択すればよいですか？

次のいずれかのパスを使用してアップグレードできます。

- Trident 演算子を使用する。
- tridentctl を使用。

> 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。Astra Trident をアップグレードする場合、アップグレードを実行する前に、以前のスナップショット CRS と CRD （ボリューム Snapshot クラス、ボリューム Snapshot 、ボリューム Snapshot コンテンツ）をすべて削除する必要があります。を参照してください "この blog" アルファスナップショットを beta/GA 仕様に移行する手順を理解する。

Trident のオペレータは、次の条件が満たされている場合にアップグレードできます。

- CSI Trident を実行している（ 19.07 以降）。
- CRD ベースの Trident リリース（ 19.07 以降）があります。
- カスタム YAML を使用して ' カスタマイズされたインストールを実行することは できません

> etcd' ベースの Trident リリース（ 19.04 以前）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。

オペレータを使用しない場合 ' またはオペレータがサポートできないカスタマイズされたインストールを使用している場合は 'tridentctl' を使用してアップグレードできますTrident リリース 19.04 以前では、これがアップグレードに推奨される方法です。

演算子に変更があります

Astra Trident の 21.01 リリースでは、アーキテクチャに関する次のような重要な変更がオペレータに導入されています。

- 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部として、「 ownerReferences 」を使用して複数のリソースを作成し、保持します。クラスタを対象としたリソースで「 ownerReferences 」を維持すると、 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 は、 Astra Trident のインストールと管理に使用されるカスタムリソースとして、 TridentOrchestrator に置き換えられました。さらに、「 TridentOrchestrator 」の仕様に新しいフィールドが追加されました。ユーザは、「 pec.namespace` 」フィールドを使用して Trident ネームスペースを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。例を見てみましょう "こちらをご覧ください"。

詳細については、こちらをご覧ください

- "Trident オペレータを使用してアップグレード"
- "b41f68e89b7b02a9077fb800d31a2fbe"

オペレータにアップグレードしてください

既存の Astra Trident インストールは、オペレータが簡単にアップグレードできます。

必要なもの

オペレータを使用してアップグレードするには、次の条件を満たしている必要があります。

- CSI ベースの Astra Trident がインストールされている必要があります。CSI Trident を実行しているかどうかを確認するには、 Trident ネームスペースのポッドを調べます。"trident-csi-*" の命名パターンに従っている場合は、 CSI Trident を実行しています。
- CRD ベースの Trident をインストールしている必要があります。19.07 以降のすべてのリリースを表します。CSI ベースのインストールを使用している場合は、 CRD ベースのインストールを使用している可能性があります。
- CSI Trident をアンインストールしても、インストールからのメタデータが保持されている場合は、オペレータを使用してアップグレードできます。
- 特定の Kubernetes クラスタ内のすべてのネームスペースに存在する Trident のは、 1 つの Astra だけです。
- を実行する Kubernetes クラスタを使用する必要があります "バージョン 1.17 以降"。
- アルファスナップショットの CRD が存在する場合は、 tridentctl を使用して削除してください。これにより、アルファスナップショット仕様の CRD が削除されます。削除または移行が必要な既存のスナップショットについては、を参照してください "この blog"。

> OpenShift Container Platform で演算子を使用して Trident を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"。

クラスタを対象としたオペレータ環境をアップグレードします

Trident 21.01 以降 * からアップグレードするには、以下の手順に従ってください。

手順

1. 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、 21.01 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 21.01/trident-installer/deploy/bundle.yaml -n trident
```

2. （オプション）インストールパラメータを変更する場合は、 Trident のインストール時に作成した「 TridentOrchestrator 」オブジェクトを編集します。これには、カスタム Trident イメージの変更、コンテナイメージを抽出するためのプライベートイメージレジストリ、デバッグログの有効化、イメージのプルシークレットの指定などの変更が含まれます。
3. 新しいバージョンの Trident 演算子を設定する 'undle.yaml ファイルを使用して Astra Trident をインストールします次のコマンドを実行します。

```
kubectl create -f 21.10.0/trident-installer/deploy/bundle.yaml -n
trident
```

このステップの一環として、 21.10.0 Trident 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

名前空間を対象とした演算子（バージョン 20.07 〜 20.10 ）を使用してインストールされた Astra Trident のインスタンスからアップグレードするには、次の手順に従います。

手順

1. 既存の Trident インストールのステータスを確認これを行うには、 TridentProvisioner の * Status * を確認してください。ステータスは「 Installed 」になります。

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

```
# Download the release required [21.01]
$ mkdir 21.07.1
$ cd 21.07.1
$ wget
https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
$ tar -xf trident-installer-21.07.1.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。この手順を完了するには、名前空間を対象とした演算子を配備するために使用する 'undle.yaml ファイルが必要です。から 'bundle.yaml を取得できます "Trident リポジトリ"。適切なブランチを使用するようにしてください。

Tridentのインストールパラメータに必要な変更を加えます（の値の変更など）`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 "パラメータのリスト"。

```
#Ensure you are in the right directory
$ pwd
$ /root/20.10.1/trident-installer

#Delete the namespace-scoped operator
$ kubectl delete -f deploy/bundle.yaml
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

#Confirm the Trident operator was removed
$ 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
```

この段階で 'trident-operator-xxxxxxxxxxxxxxx Pod が削除されます

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、「 TridentProvisioner 」の仕様を更新します。これらの変更には、コンテナイメージをからプルするためのプライベートイメージレジストリの変更、デバッグログの有効化、イメージプルシークレットの指定などがあります。

```
$  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演算子をインストールします。

> クラスタを対象とした演算子をインストールすると、「 TridentProvisioner 」オブジェクトから「 TridentOrchestrator 」オブジェクトへの移行が開始され、「 TridentProvisioner 」オブジェクトと「 tridentProvisioner`CRD 」が削除され、 Astra Trident が使用中のクラスタを対象とした演算子のバージョンにアップグレードされます。次の例では、 Trident が 21.07.1 にアップグレードされています。

> クラスタを対象とした演算子を使用して Astra Trident をアップグレードすると、「 tridentProvisioner 」が同じ名前の「 tridentOrchestrator 」オブジェクトに移行します。これは、オペレータによって自動的に処理されます。アップグレードの際には、 Astra Trident が以前と同じネームスペースにインストールされる予定です。

```
#Ensure you are in the correct directory
$ pwd
$ /root/21.07.1/trident-installer

#Install the cluster-scoped operator in the **same namespace**
$ 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

#Examine Trident pods in the namespace
$ kubectl get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m41s
trident-csi-xrst8                   2/2     Running   0          1m41s
trident-operator-5574dbbc68-nthjv   1/1     Running   0          1m52s

#Confirm Trident has been updated to the desired version
$ kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v21.07.1
```

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。

手順

1. 最新の Astra Trident リリースをダウンロード
2. 「 helm upgrade 」コマンドを使用します。次の例を参照してください。

```
$ helm upgrade <name> trident-operator-21.07.1.tgz
```

ここで 'trident-operator-21.07.tgz は ' アップグレード先のバージョンを反映しています

3. Helm list を実行して ' グラフとアプリケーション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

> アップグレード中に構成データを渡すには '--set' を使用します

たとえば 'tridentDebug' のデフォルト値を変更するには ' 次のコマンドを実行します

```
$ helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
tridentDebug=true
```

$tridentctl logs を実行すると ' デバッグメッセージを表示できます

> 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する場合は、オプションが upgrade コマンドに含まれていることを確認してください。含まれていない場合は、値がデフォルトにリセットされます。

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

CSI Trident インスタンスが上記の前提条件を満たしている場合は、 Trident オペレータの最新リリースにアップグレードできます。

手順

1. 最新の Astra Trident リリースをダウンロード

```
# Download the release required [21.07.1]
$ mkdir 21.07.1
$ cd 21.07.1
$ wget
https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
$ tar -xf trident-installer-21.07.1.tar.gz
$ cd trident-installer
```

2. マニフェストから「 tridentオーケストラ 」 CRD を作成します。

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

3. オペレータを配備します。

```
#Install the cluster-scoped operator in the **same namespace**
$ 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-csi-79df798bdc-m79dc        6/6     Running   0          150d
trident-csi-xrst8                   2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv   1/1     Running   0          1m30s
```

4. Astra Trident をインストールするための TridentOrchestrator CR を作成します。

```
#Create a tridentOrchestrator to initate a Trident install
$ 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

#Confirm Trident was upgraded to the desired version
$ kubectl describe torc trident | grep Message -A 3
Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v21.07.1
```

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

tridentctl 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

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。

- 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。
- ボリュームスナップショットのベータリリースでは、一連の新しい CRD とスナップショットコントローラが導入されています。どちらも Astra Trident をインストールする前にセットアップする必要があります。

> "この blog" alpha ボリュームの Snapshot をベータ版に移行する手順について説明します。

このタスクについて

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。

> Astra Trident をアップグレードするときは、アップグレードプロセスを中断しないでください。インストーラが実行されていることを確認します。

アップグレード後の次の手順

新しい Trident リリース（ On-Demand Volume Snapshots など）で利用できる豊富な機能を活用するには、「 tridentctl upgrade 」コマンドを使用してボリュームをアップグレードします。

レガシーボリュームがある場合は、それらのボリュームを NFS/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できるように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。

CSI タイプにボリュームをアップグレードする場合は、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。

ボリュームのアップグレードの例

次の例は、ボリュームのアップグレードを実行する方法を示しています。

1. kubectl get pv を実行して PVS を一覧表示します

```
$ 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「 NetApp.io/trident 」プロビジョニングツールを使用して作成された 4 つの PVS があります。

2. kubectl describe pv を実行して PV の詳細を取得します。

```
$ 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 'NetApp.io/trident プロビジョニングツールを使用して作成されたものであり 'NFS タイプのものです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. tridenctl upgrade volume <name-of -trident-volume>` コマンドを実行して、従来の Astra Trident ボリュームを CSI 仕様にアップグレードします。

```
$ ./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. kubectl describe pv を実行して、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>
```

このようにして、 Astra Trident によって作成された NFS/iSCSI タイプのボリュームを、ボリューム単位で CSI タイプにアップグレードできます。

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 'helm uninstall を使用してアンインストールできます

```
#List the Helm release corresponding to the 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。

- 「 **TridentOrchestrator** 」を編集してアンインストールフラグを設定してください：「 TridentOrchestrator 」を編集し、「 PEC.UNINSTALL = TRUE 」を設定できます。「 TridentOrchestrator 」 CR を編集し、次に示すように「 uninstall 」フラグを設定します。

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

  uninstall` フラグが 'true' に設定されている場合、 Trident オペレータは Trident をアンインストールしますが、 TridentOrchestrator 自体は削除しません。Trident を再度インストールする場合は、 TridentOrchestrator をクリーンアップして新しい Trident を作成する必要があります。

- 「 **TridentOrchestrator** 」を削除 Astra Trident のデプロイに使用した「 TridentOrchestrator 」 CR を削除することで、 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。オペレータは「 TridentOrchestrator 」の削除を処理し、 Astra Trident の展開とデストレーションを削除し、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 (それが作成する CRD を含む) 、スレートをクリーンにして効果的に消去するには、「 TridentOrchestrator 」を編集して「 wippeout 」オプションを渡すことができます。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`

次のように 'tridentctl コマンドを実行して '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 および関連オブジェクトは削除されますこれにより ' インストーラを再実行して ' より新しいバージョンに更新することが容易になります

```
./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。

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

ダウングレードするタイミング

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 は現在、ステートを維持するために 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、「 trident PV 」（以前にインストールされた Astra Trident のバージョンで使用）に書き込まれたデータではなく、関連する CRD オブジェクトがあります。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。ダウングレードが必要な場合は、 CRD （ 19.07 以降）を使用して実行されている Astra Trident のバージョンに対してのみ実行してください。これは、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すべての処理が認識されるようにするためです。

ダウングレードしない場合

「 etcd」 を使用して状態を維持する Trident のリリース（ 19.04 以前）にダウングレードしないでください。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では ' ダウングレード・プロセスは異なり 'tridentctl を使用する必要はありません

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** を削除しないでください。
2. クラスタを対象とした演算子を削除します。これを行うには、オペレータを配置するために使用するマニフェストが必要です。から入手できます "Trident GitHub repo"。必要なブランチに切り替えていることを確認してください。
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。 「 tridentオーケストラ 」が削除されていることを確認してください。

```
#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. クラスタを対象とした演算子を削除します。これを行うには、オペレータを配置するために使用するマニフェストが必要です。このファイルは、から入手できます "Trident GitHub repo"。必要なブランチに切り替えていることを確認してください。
3. 「 tridentオーケストラ 」の「 CRD 」を削除します。

```
#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 'helm rollback コマンドを使用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`

tridentctl を使用して Astra Trident をインストールした場合、ダウングレードプロセスで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「 etcd」 のスナップショットを作成する必要があります。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. Trident が tridentctl を使用してインストールされていることを確認します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。tridentctl を使用するか、 Trident ポッドで使用されるイメージを見ることができます。
   c. 「 tridentOrchestrator 」、（または）「 tridentProvisioner` 」、（または）「 trident-operator-xxxxxxxxx-xxxxx 」という名前のポッドが表示されない場合、 Astra Trident * は「 tridentctl 」を使用してインストールされます。
2. 既存の tridentctl バイナリを使用して Astra Trident をアンインストールしますこの場合は、 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

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、ストレージクラスが定義した要件に合わせて、バックエンドからストレージプールを自動的に提供します。お使いのストレージシステムのタイプに基づいたバックエンドの設定の詳細については、こちらを参照してください。

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

Azure NetApp Files バックエンドを設定します

提供されている構成例を使用して、 Azure NetApp Files （ANF ）を Astra Trident インストールのバックエンドとして設定する方法を説明します。

> 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。

必要なもの

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。

- Azure NetApp Files が有効な Azure サブスクリプションのスクリプト ID 。
- 「 tenantID 」、「 clientID 」、「 clientSecret 」を「」から選択します "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。App Registration では、Azure で事前定義されている「 Owner 」または「 Contributor 」の役割を使用する必要があります。

> Azure の組み込みロールの詳細については、を参照してください "Azure に関するドキュメント"。

- 少なくとも 1 つを含む Azure の「場所」 "委任されたサブネット"。Trident 22.01 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。
- Azure NetApp Files を初めて使用する場合や新しい場所で使用する場合は、いくつかの初期設定が必要です。を参照してください "クイックスタートガイド"。

このタスクについて

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）に基づいて、要求された場所で利用可能な容量プールに ANF ボリュームを作成し、要求されたサービスレベルとサブネットに

対応します。

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

バックエンド構成オプション

バックエンド設定オプションについては、次の表を参照してください。

パラメータ	説明	デフォルト
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「 azure-NetApp-files 」
backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + ランダムな文字
' スクリプト ID' 。	Azure サブスクリプションのサブスクリプション ID	
「 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 」を入力します。	アプリケーション登録からのクライアントシークレット	
「サービスレベル」	「標準」、「プレミアム」、「ウルトラ」のいずれかです	"" （ランダム）
「ロケーション」	新しいボリュームを作成する Azure の場所の名前	
「サービスレベル」	「標準」、「プレミアム」、「ウルトラ」のいずれかです	"" （ランダム）
「 resourceGroups 」	検出されたリソースをフィルタリングするためのリソースグループのリスト	"[]" （フィルタなし）
「 netappAccounts 」のように入力します	検出されたリソースをフィルタリングするためのネットアップアカウントのリスト	"[]" （フィルタなし）
「 capacityPools	検出されたリソースをフィルタリングする容量プールのリスト	"[]" （フィルタなし、ランダム）
「 virtualNetwork 」	委任されたサブネットを持つ仮想ネットワークの名前	""
「サブネット」	「 icrosoft.Netapp/volumes` 」に委任されたサブネットの名前	""
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "

パラメータ	説明	デフォルト
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例 : `\{"API":false," メソッド ":true," 検出 ":true}` トラブルシューティングを行って詳細なログダンプが必要な場合を除き、このオプションは使用しないでください。	null

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。Astra Trident は、デバッグが有効なときにバックエンドが作成されたときに、検出した 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かどうかを確認してください。

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場合は、 NFS v4.1 を選択するために、カンマ区切りのマウントオプションリストに「 nfsvers=4` 」を含めることができます。ストレージクラスで設定されたマウントオプションは、バックエンド構成ファイルで設定されたマウントオプションよりも優先されます。

resourceGroups’netappAccounts’capacityPools’virtualNetwork' および 'subnet' の値は ' 短い名前または完全修飾名を使用して指定できます省略形は同じ名前の複数のリソースに一致している可能性があるため、ほとんどの場合は完全修飾名を使用することを推奨します。resourceGroups’netappAccounts’capacityPools' の値は ' 検出されたリソースのセットをこのストレージバックエンドで使用可能な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 >

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、各ボリュームのデフォルトのプロビジョニング方法を制御できます。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
「 exportRule 」	新しいボリュームのエクスポートルール	"0.0.0.0/0 "
「スナップショット方向」	.snapshot ディレクトリの表示を制御します	いいえ

パラメータ	説明	デフォルト
「 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）	""（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）

「 exportRule 」の値は、 CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切ったリストにする必要があります。

> ANF バックエンドに作成されたすべてのボリュームに対して、ストレージプールに含まれるすべてのラベルが、プロビジョニング時にストレージボリュームにコピーされます。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これにより、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを簡単に区別できます。

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。

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 'Ultra 容量プール内の Azure の eastus ロケーションにボリュームを配置します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",
        "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** ：仮想ストレージプールの構成

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。

```
{
    "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"],
    "nfsMountOptions": "vers=3,proto=tcp,timeo=600",
    "labels": {
        "cloud": "azure"
    },
    "location": "eastus",

    "storage": [
        {
            "labels": {
                "performance": "gold"
            },
            "serviceLevel": "Ultra",
            "capacityPools": ["ultra-1", "ultra-2"]
        },
        {
            "labels": {
                "performance": "silver"
            },
            "serviceLevel": "Premium",
            "capacityPools": ["premium-1"]
        },
        {
            "labels": {
                "performance": "bronze"
            },
            "serviceLevel": "Standard",
            "capacityPools": ["standard-1", "standard-2"]
        }
    ]
}
```

以下の「 torageClass 」定義は、上記のストレージプールを参照しています。「 parameters.selector` 」フィールドを使用すると、ボリュームのホストに使用される仮想プールを「 S torageClass 」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。

```
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
```

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。

GCP バックエンドの CVS を設定します

提供されている構成例を使用して、ネットアップ Cloud Volumes Service （ CVS ） for Google Cloud Platform （ GCP ）を Astra Trident インストールのバックエンドとして設定する方法を説明します。

NetApp Cloud Volumes Service for Google Cloud では、サイズが 100GiB 未満の CVS パフォーマンスボリュームや 300GiB 未満の CVS ボリュームはサポートされていません。Trident は、要求されたボリュームが最小サイズより小さい場合、最小サイズのボリュームを自動的に作成します。

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。

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

Astra Trident に、小規模なボリュームがデフォルトでサポートされるようになりました "GCP 上の CVS サービスタイプ"。を使用して作成したバックエンドの場合 `storageClass=software`をクリックすると、ボリュームのプロビジョニングサイズが300GiB以上になります。現在、 CVS ではこの機能が限定的な可用性で提供されており、テクニカルサポートは提供されていません。1TiB 未満のボリュームにアクセスするには、ユーザがサインアップする必要があります "こちらをご覧ください"。 非本番 のワークロードでは 1TiB 未満のボリュームを使用することを推奨します。

デフォルトの CVS サービスタイプ（「 torageClass=software 」）を使用してバックエンドを導入する場合、該当するプロジェクト番号とプロジェクト ID について、 GCP の 1TiB 未満のボリューム機能へのアクセス権をユーザが取得する必要があります。これは Astra Trident で sub-1TiB 個のボリュームをプロビジョニングするために必要です。この条件を指定しない場合、 600 GiB 未満の PVC でボリュームの作成が失敗します。を使用して 1TiB 未満のボリュームへのアクセスを取得します "このフォーム"。

デフォルトの CVS サービスレベル用に Astra Trident で作成されたボリュームは、次のようにプロビジョニングされます。

- 300GiB 未満の PVC があると、 Astra Trident によって 300GiB の CVS ボリュームが作成されます。
- 300GiB から 600GiB の PVC があると、 Astra Trident が要求されたサイズの CVS ボリュームを作成します。
- 600GiB から 1TiB までの PVC の場合、 Astra Trident によって 1TiB の CVS ボリュームが作成されます。
- 1TiB を超える PVC の場合、要求サイズの CVS ボリュームが Astra Trident に作成されます。

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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	"GCP-cvs"
backendName`	カスタム名またはストレージバックエンド	ドライバ名 + "_" + API キーの一部

パラメータ	説明	デフォルト
'torageClass'	ストレージのタイプ「ハードウェア」（パフォーマンス最適化）または「ソフトウェア」（ CVS サービスタイプ）から選択可能	
「 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。	
「 apiRegion 」と入力します	CVS アカウント地域。バックエンドがボリュームをプロビジョニングするリージョンです。	
「 apiKey 」と入力します	「 netappcloudvolumes 」ロールを持つ Google Cloud サービスアカウントの API キー。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 」と入力します	CVS アカウントへの接続にプロキシサーバが必要な場合は、プロキシ URL を指定します。プロキシサーバには、 HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、プロキシサーバで自己署名証明書を使用するために証明書の検証はスキップされます。認証が有効になっているプロキシサーバはサポートされていません。	
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。	"nfsvers=3 "
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合はプロビジョニングが失敗します	"" （デフォルトでは適用されません）
「サービスレベル」	新しいボリュームの CVS サービスレベル。「 Standard 」、「 Premium 」、「 Extreme 」のいずれかです。	標準
「ネットワーク」	CVSボリュームに使用するGCPネットワーク	デフォルト

パラメータ	説明	デフォルト
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例：\\{"API":false," メソッド ":true}`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

共有 VPC ネットワークを使用する場合は、「 ProjectNumber 」と「 hostProjectNumber 」の両方を指定する必要があります。その場合、「 ProjectNumber 」はサービスプロジェクト、「 hostProjectNumber 」はホストプロジェクトです。

「 apiRegion 」は、 Astra Trident が CVS ボリュームを作成する GCP リージョンです。複数リージョンの Kubernetes クラスタを作成する場合、「 apiRegion 」で作成した CVS ボリュームは、複数の GCP 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がかかることに注意してください。

- クロスリージョンアクセスを有効にするには、「 allowedTopologies 」の StorageClass 定義に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
```

- 'torageClass' は、目的のを選択するために使用できるオプションのパラメータです "CVS サービスタイプ"。基本の CVS サービスタイプ (`torageClass=software') または CVS-Performance サービスタイプ (`torageClass=hardware') から選択できます。このサービスタイプは、 Trident がデフォルトで使用します。バックエンド定義でそれぞれの CVS の torageClass を提供する 'apiRegion' を指定していることを確認します

Astra Trident は、 Google Cloud 上の基本 CVS サービスタイプと統合されている ベータ版の機能 で、本番環境のワークロード向けではありません。Trident は、 CVS パフォーマンスサービスタイプでは完全にサポートされている ** で、デフォルトで使用されます。

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。

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、各ボリュームのデフォルトのプロビジョニング方法を制御できます。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
「 exportRule 」	新しいボリュームのエクスポートルール	"0.0.0.0/0 "
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスします	いいえ

パラメータ	説明	デフォルト
「スナップショット予約」	Snapshot 用にリザーブされているボリュームの割合	"" （ CVS のデフォルト値をそのまま使用）
「 size 」	新しいボリュームのサイズ	"100Gi"

「 exportRule 」の値は、 CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切ったリストにする必要があります。

> CVS Google Cloud バックエンドで作成されたすべてのボリュームについて、 Trident は、ストレージプールにあるすべてのラベルを、プロビジョニング時にストレージボリュームにコピーします。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これにより、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを簡単に区別できます。

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。

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
```

```
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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** ：基本 **CVS** サービスタイプの設定

この例は、基本 CVS サービスタイプを使用するバックエンド定義を示しています。このサービスタイプは、汎用ワークロード向けであり、パフォーマンスが低く、ゾーンの可用性も高くなります。


```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "storageClass": "software",
    "apiRegion": "us-east4",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
```

```
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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 **3** ：単一のサービスレベルの設定

この例は、 Google Cloud us-west2 リージョン内のすべての Astra Trident で作成されたストレージに同じ要素を適用するバックエンドファイルを示しています。この例は ' バックエンド構成ファイルでの ProxyURL の使用方法も示しています

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
```

```
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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 **4** ：仮想ストレージプールの構成

この例は、仮想ストレージプールで構成されたバックエンド定義ファイルと、それを参照する「

torageClasses 」を示しています。

以下に示すバックエンド定義ファイルの例では ' 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これにより 'napshotReserve が 5% に設定され 'exportRule' が 0.0.0.0/0 に設定されます仮想ストレージプールは「ストレージ」セクションで定義します。この例では ' 各ストレージ・プールが独自のサービス・レベルを設定し ' 一部のプールがデフォルト値を上書きします

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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"
        }
    ]
}
```

次の StorageClass 定義は、上記のストレージプールを参照してください。parameters.selector` フィールドを使用すると ' ボリュームのホストに使用される仮想プールを各 StorageClass 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。

最初の StorageClass （「 cvs-mextreme -extra-protection 」）は、最初の仮想ストレージプールにマッピングされます。スナップショット予約が 10% の非常に高いパフォーマンスを提供する唯一のプールです。最後の StorageClass （「 cvs-extra-protection 」）は、 10% のスナップショット予約を提供するストレージプールを呼び出します。Trident が、どの仮想ストレージプールを選択するかを決定し、 Snapshot リザーブの要件を確実に満たします。

```
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
```

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。

必要なもの

- 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"。

知っておくべきこと

'olidfire -SAN' ストレージ・ドライバは ' ファイル・モードとブロック・モードの両方をサポートしています「 Filesystem 」 volumeMode の場合、 Astra Trident はボリュームを作成し、ファイルシステムを作成します。ファイルシステムのタイプは StorageClass で指定されます。

ドライバ	プロトコル	ボリュームモード	サポートされているアクセスモード	サポートされるファイルシステム
「 olidfire -san 」	iSCSI	ブロック	RWO 、 ROX 、 RWX	ファイルシステムがありません。raw ブロックデバイスです。
「 olidfire -san 」	iSCSI	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。
「 olidfire -san 」	iSCSI	ファイルシステム	RWO 、 ROX	「 xfs 」、「 ext3 」、「 ext4 」
「 olidfire -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。CSI 以外の Trident で CHAP を使用する場合は 'UseCHAP' オプションを明示的に設定することをお勧めしますそれ以外は、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。

AccessGroups または UseCHAP のどちらも設定されていない場合は ' 次のいずれかの規則が適用されます

- デフォルトの trident' アクセスグループが検出された場合は ' 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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	常に「 solidfire-san-」
backendName`	カスタム名またはストレージバックエンド	「 iSCSI_ 」 + ストレージ（ iSCSI ） IP アドレス SolidFire
「エンドポイント」	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP	
「 VIP 」	ストレージ（ iSCSI ）の IP アドレスとポート	
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット。	「」
「 tenantname 」	使用するテナント名（見つからない場合に作成）	
「 InitiatorIFCace 」	iSCSI トラフィックを特定のホストインターフェイスに制限します	デフォルト
UseCHAP'	CHAP を使用して iSCSI を認証します	正しいです
「アクセスグループ」	使用するアクセスグループ ID のリスト	「 trident 」という名前のアクセスグループの ID を検索します。
「タイプ」	QoS の仕様	
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します	“”（デフォルトでは適用されません）
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true}	null

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

Astra Trident は、ボリュームを作成すると、ストレージプール上のすべてのラベルを、プロビジョニング時にバッキングストレージ LUN にコピーします。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これにより、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを簡単に区別できます。

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 ' ストレージ・クラスを定義して 'iops`storage クラス・パラメータを使用します

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

例**2**：のバックエンドとストレージクラスの設定 `solidfire-san` 仮想ストレージプール用のドライバ

この例は、仮想ストレージプールで設定されたバックエンド定義ファイルと、それらを参照する StorageClasses を示しています。

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 'type' が Silver に設定されます仮想ストレージプールは「ストレージ」セクションで定義します。この例では、一部のストレージプールで独自のタイプが設定されており、一部のプールでは上記で設定したデフォルト値が上書きされます。

```
{
    "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、上記の仮想ストレージプールを参照してください。parameters.selector` フィールドを使用すると ' 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ボリュームには、選択した仮想プール内で定義された要素があります。

最初の StorageClass （ 'olidfire-gold -f4` ）は、最初の仮想ストレージプールにマップされます。ゴールドの

ボリューム・タイプ QoS を備えた唯一のゴールド・パフォーマンスを提供するプールです最後の StorageClass （ 'olidfire-cin`) は、シルバーパフォーマンスを提供するストレージプールをすべて呼び出します。Trident が、どの仮想ストレージ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"

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

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。

- "準備"
- "設定と例"

ユーザ権限

Astra Trident は、 ONTAP 管理者または SVM 管理者のいずれかとして実行されることを想定しています。通常は、「 admin 」クラスタユーザまたは「 vsadmin 」 SVM ユーザを使用するか、同じロールを持つ別の名前のユーザを使用します。Amazon FSX for NetApp ONTAP 環境では、 Astra Trident は、 ONTAP 管理者または SVM 管理者として、クラスタ「 fsxadmin 」ユーザまたは「 vsadmin 」 SVM ユーザ、または同じロールを持つ別の名前のユーザを実行する必要があります。「 fsxadmin 」ユーザは、クラスタ管理ユーザの限定的な代替ユーザです。

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

ONTAP 内では、 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。

準備

ONTAP SAN ドライバを使用して ONTAP バックエンドを設定するための準備方法について説明します。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 'ONTAP-SAN' ドライバを使用する「 -dev 」クラスと 'ONTAP-SAN-エコノミー 'one を使用する「デフォルト」クラスを設定できます

すべての Kubernetes ワーカーノードに適切な iSCSI ツールをインストールしておく必要があります。を参照してください "こちらをご覧ください" 詳細：

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。ONTAP バージョンとの互換性を最大限に高めるために 'admin' または vsadmin などの事前定義されたセキュリティ・ログイン・ロールを使用することを推奨します
- 証明書ベース： 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。

また、既存のバックエンドを更新したり、クレデンシャルベースから証明書ベースに移行したり、その逆に移行したりすることもできます。クレデンシャルと証明書の両方が * 提供されている場合、 Astra Trident は、バックエンド定義からクレデンシャルを削除するように警告を発行しながら、デフォルトで証明書を使用します。

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。「 admin 」や「 vsadmin 」など、事前定義された標準的な役割を使用することをお勧めします。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。

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

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=admin"
```

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

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

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

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

4. ONTAP セキュリティ・ログイン・ロールが 'cert' 認証方式をサポートしていることを確認します

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

5. 生成された証明書を使用して認証をテストONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

```
$ cat cert-backend.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaallllluuuueeee",
"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、別の認証方式を使用したり、クレデンシャルをローテーションしたりすることができます。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 ' tridentctl backend update を実行するために必要なパラメータを含む更新された backend.json ファイルを使用します

```
$ cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "secret",
"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。

igroup を指定します

Astra Trident は、 igroup を使用して、プロビジョニングするボリューム（ LUN ）へのアクセスを制御します。管理者はバックエンドに igroup を指定する方法として、次の 2 つを選択できます。

- Astra Trident では、バックエンドごとに igroup を自動的に作成、管理できます。バックエンド定義に igroupName が含まれていない場合、 Astra Trident は、 SVM 上に trident-<backend-UUID> という名前の igroup を作成します。これにより、各バックエンドに専用の igroup が割り当てられ、 Kubernetes ノードの IQN の自動追加や削除が処理されます。

- また、事前に作成された igroup もバックエンドの定義で提供できます。これは 'igroupName’config パラメータを使用して実行できますAstra Trident が、 Kubernetes ノードの IQN を既存の igroup に追加または削除します。

igroupName が定義されているバックエンドの場合 'igroupName を tridentctl バックエンド・アップデートで削除して 'Astra Trident の自動ハンドル・ igroup を持つことができますすでにワークロードに接続されているボリュームへのアクセスが中断されることはありません。今後作成される igroup Astra Trident を使用して接続を処理します。

> Astra Trident の一意のインスタンスごとに igroup を専用にすることを推奨します。これは、Kubernetes 管理者とストレージ管理者にとって有益です。CSI Trident は、クラスタノード IQN の igroup への追加と削除を自動化し、管理を大幅に簡易化します。Kubernetes 環境（および Astra Trident インストール）全体で同じ SVM を使用する場合、専用の igroup を使用することで、ある Kubernetes クラスタに対する変更が、別の Kubernetes クラスタに関連付けられた igroup に影響しないようにできます。また、 Kubernetes クラスタ内の各ノードに一意の IQN を設定することも重要です。前述のように、 Astra Trident は IQN の追加と削除を自動的に処理します。ホスト間で IQN を再使用すると、ホスト間で誤って認識されて LUN にアクセスできないような、望ましくないシナリオが発生する可能性があります。

Astra Trident が CSI Provisioner として機能するように設定されている場合、 Kubernetes ノード IQN は自動的に igroup に追加 / 削除されます。Kubernetes クラスタにノードを追加すると 'trident-csi ’ DemonSet によって ' 新しく追加されたノードにポッド（ trident-csi-xxxxx ）が導入され ' ボリュームを接続できる新しいノードが登録されますノード IQN もバックエンドの igroup に追加されます。ノードが遮断され、削除され、Kubernetes から削除された場合も、同様の手順で IQN の削除が処理されます。

Astra Trident が CSI Provisioner として実行されない場合は、 Kubernetes クラスタ内のすべてのワーカーノードからの iSCSI IQN を含むように、 igroup を手動で更新する必要があります。Kubernetes クラスタに参加するノードの IQN を igroup に追加する必要があります。同様に、 Kubernetes クラスタから削除されたノードの IQN を igroup から削除する必要があります。

双方向 **CHAP** を使用して接続を認証します

Astra Trident は 'ONTAP-SAN' ドライバと 'ONTAP-SAN-エコノミー ドライバの双方向 CHAP を使用して iSCSI セッションを認証できますこれには ' バックエンド定義で useCHAP オプションを有効にする必要がありますtrue に設定すると 'Astra Trident は SVM のデフォルトイニシエータセキュリティを双方向 CHAP に構成し ' 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": "FaKePaSsWoRd",
    "igroupName": "trident",
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
}
```

「 useCHAP 」パラメータは、 1 回だけ設定できるブール型のオプションです。デフォルトでは false に設定されています。true に設定したあとで、 false に設定することはできません。

「 useCHAP=true' に加えて、「 chapInitiatorSecret 」、「 chapTargetInitiatorSecret 」、「 chapTargetUsername 」、および「 chapUsername 」フィールドもバックエンド定義に含める必要があります。シークレットは 'tridentctl update を実行してバックエンドを作成した後に変更できます

動作の仕組み

「 useCHAP 」を 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 を構成するように 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「 CHAP 」に設定し、 CHAP イニシエータとターゲットのユーザ名とシークレットの設定に進みます。
  - SVM に LUN が含まれている場合、 Trident は SVM で CHAP を有効にしません。これにより、 SVM にすでに存在する LUN へのアクセスが制限され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）。
- バックエンドに与えられた 'igroupName' へのイニシエータの追加を管理する指定されていない場合、デフォルトは「 trident 」になります。

バックエンドが作成されると、 Astra Trident は対応する「 tridentbackend 」 CRD を作成し、 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 証明書を更新するには 'backend.json ファイルの CHAP パラメータを更新しますこれには 'CHAP シー

クレットを更新し 'tridentctl update コマンドを使用してこれらの変更を反映する必要があります

> バックエンドの CHAP シークレットを更新する場合は 'tridentctl' を使用してバックエンドを更新する必要があります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": "FaKePaSsWoRd",
    "igroupName": "trident",
    "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。

設定オプションと例

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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「 ONTAP-NAS 」、「 ONTAP-NAS-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
「管理 LIF 」	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。IPv6 には角かっこを使用します。設定後に更新することはできません	特に指定がないかぎり、 SVM が派生します
「 useCHAP 」	CHAP を使用して ONTAP SAN ドライバ用の iSCSI を認証する [ブーリアン]	いいえ
「 chapInitiatorSecret 」	CHAP イニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	「」
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	「」
「 chapTargetInitiatorSecret 」	CHAP ターゲットイニシエータシークレット。「 useCHAP = TRUE 」の場合は必須	「」
「 chapUsername 」	インバウンドユーザ名。「 useCHAP = TRUE 」の場合は必須	「」
「 chapTargetUsername 」	ターゲットユーザ名。「 useCHAP = TRUE 」の場合は必須	「」
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	「」
「 clientPrivateKey 」	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます	「」
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	「」
「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	「」

パラメータ	説明	デフォルト
「 password 」と入力します	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	「」
'VM'	使用する Storage Virtual Machine	SVM 「管理 LIF 」が指定されている場合に生成されます
「 igroupName 」と入力します	SAN ボリュームで使用する igroup の名前	"trident-<backend-UUID> "
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	Trident
「 AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	“”（デフォルトでは適用されません）
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。	“”（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	100
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true}	null
「 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 *	いいえ

「 useREST 」は **tech preview** として提供されています。これはテスト環境では推奨され、本番ワークロードでは推奨されません。「 true 」に設定すると、 Astra Trident は ONTAP REST API を使用してバックエンドと通信します。この機能を使用するには、 ONTAP 9.9 以降が必要です。また ' 使用する ONTAP ログイン・ロールには ONTAP アプリケーションへのアクセス権が必要ですこれは ' 事前に定義された vsadmin ロールと cluster-admin ロールによって満たされます

ONTAP クラスタと通信するには、認証パラメータを指定する必要があります。これは、セキュリティログインまたはインストールされている証明書のユーザ名 / パスワードです。

NetApp ONTAP バックエンドに Amazon FSX を使用している場合は、「 limitAggregateUsage 」パラメータを指定しないでください。Amazon FSX for NetApp ONTAP が提供する「 fsxadmin 」と「 vsadmin 」の役割には、集計の使用状況を取得したり、 Astra Trident を介して制限したりするために必要なアクセス権限が含まれていません。

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

「 ONTAP -SAN 」ドライバの場合、デフォルトでは SVM のすべてのデータ LIF IP が使用され、 iSCSI マルチパスが使用されます。「 ONTAP -SAN 」ドライバのデータ LIF の IP アドレスを指定すると、マルチパスが無効になり、指定したアドレスだけが使用されます。

> バックエンドを作成するときは、作成後に「 datalif 」と「 toragePrefix 」を変更できないことに注意してください。これら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。

igroupName は、 ONTAP クラスタ上ですでに作成されている igroup に設定できます。指定しない場合、 Trident は trident-<backend-UUID> という名前の igroup を自動的に作成します。事前に定義された igroupName を指定する場合は、各 Kubernetes クラスタで igroup を使用することを推奨します。ただし、 SVM が環境間で共有される場合です。これは、 Astra Trident が IQN の追加や削除を自動的に維持するために必要です。

バックエンドは、作成後に igroup を更新することもできます。

- igroupName は、 Astra Trident の外部の SVM で作成および管理される新しい igroup を指すように更新できます。
- igroupName は省略できます。この場合、 Astra Trident は Trident によって trident-<backend-UUID> igroup が自動的に作成および管理されます。

どちらの場合も、ボリュームの添付ファイルには引き続きアクセスできます。以降のボリューム接続では、更新された igroup が使用されます。この更新によって、バックエンドにあるボリュームへのアクセスが中断されることはありません。

「管理 LIF 」オプションには完全修飾ドメイン名（ FQDN ）を指定できます。

すべての ONTAP ドライバ用の「管理 LIF 」を IPv6 アドレスに設定することもできます。--use-ipv6' フラグを付けて Trident をインストールしてください角かっこで囲まれた「管理 LIF 」 IPv6 アドレスを定義するように注意する必要があります。

> IPv6 アドレスを使用する場合は、 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555] のように、バックエンド定義に含まれている場合は「 anagementlif` 」と「 datalif 」が角かっこ内に定義されていることを確認してください。「 data lif 」を指定しない場合、 Astra Trident は SVM から IPv6 データ LIF を取得します。

SAN ドライバで CHAP を使用できるようにするには ' バックエンド定義で useCHAP パラメータを true に設定しますその後、 Astra Trident が、バックエンドで指定された SVM のデフォルト認証として双方向 CHAP を設定して使用します。を参照してください "こちらをご覧ください" その仕組みについては、を参照してください。

「 ONTAP-SAN-エコノミー 」ドライバの場合は、「 limitVolumeSize 」オプションによって、 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されます。

> Astra Trident は 'ONTAP-SAN' ドライバを使用して作成されたすべてのボリュームの Comments フィールドにプロビジョニングラベルを設定します作成された各ボリュームについて、 FlexVol の [Comments] フィールドに、配置先のストレージプールにあるすべてのラベルが入力されます。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これにより、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを簡単に区別できます。

ボリュームのプロビジョニング用のバックエンド構成オプション

これらのオプションを使用して、構成の特別なセクションで各ボリュームをデフォルトでプロビジョニングする方法を制御できます。例については、以下の設定例を参照してください。

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	正しいです
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	なし
「ナプショットポリシー」	使用する Snapshot ポリシー	なし
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	「 napshotPolicy 」が「 none 」の場合、それ以外の場合は「」
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	ネットアップのボリューム暗号化を有効にします	いいえ
'ecurityStyle'	新しいボリュームのセキュリティ形式	「 UNIX 」
階層ポリシー	「なし」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」

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。

次に、デフォルトが定義されている例を示します。

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "managementLIF": "10.0.0.1",
 "dataLIF": "10.0.0.2",
 "svm": "trident_svm",
 "username": "admin",
 "password": "password",
 "labels": {"k8scluster": "dev2", "backend": "dev2-sanbackend"},
 "storagePrefix": "alternate-trident",
 "igroupName": "custom",
 "debugTraceFlags": {"api":false, "method":true},
 "defaults": {
     "spaceReserve": "volume",
     "qosPolicy": "standard",
     "spaceAllocation": "false",
     "snapshotPolicy": "default",
     "snapshotReserve": "10"
 }
}
```

> 「 'SAN 」ドライバを使用して作成されたすべてのボリュームに対して 'Astra Trident は 'LUN のメタデータに対応するために FlexVol にさらに 10% の容量を追加しますLUN は、ユーザが PVC で要求したサイズとまったく同じサイズでプロビジョニングされます。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。

「スナップショット予約」を定義するバックエンドの場合、 Astra Trident は次のようにボリュームのサイズを計算します。

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

1.1 は、 Astra Trident の 10% の追加料金で、 FlexVol のメタデータに対応します。「 napshotReserve 」=5% 、 PVC 要求 =5GiB の場合、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 コマンドは ' 次の例のような結果を表示する必要があります

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

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

最小限の設定例

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

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

`ontap-san` 証明書ベースの認証を使用するドライバ

これは、バックエンドの最小限の設定例です。「 clientCertificate` 」、「 clientPrivateKey 」、「 trustedCACertifate` （信頼された CA を使用する場合はオプション）」は「 backend.json 」に格納され、それぞれクライアント証明書、秘密鍵、信頼された CA 証明書の Base64 でエンコードされた値を取得します。

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "DefaultSANBackend",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "clientCertificate": "ZXR0ZXJwYXB...ICMgJ3BhcGVyc2",
    "clientPrivateKey": "vciwKIyAgZG...0cnksIGRlc2NyaX",
    "trustedCACertificate": "zcyBbaG...b3Igb3duIGNsYXNz"
}
```

`ontap-san` 双方向**CHAP**を備えたドライバ

これは、バックエンドの最小限の設定例です。この基本的な構成では 'useCHAP' を true' に設定して 'ONTAP-SAN' バックエンドを作成します

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "labels": {"k8scluster": "test-cluster-1", "backend": "testcluster1-
sanbackend"},
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

`ontap-san-economy` ドライバ

```
{
    "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",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret"
}
```

仮想ストレージプールを使用するバックエンドの例

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たとえば 'paceReserve at none'`paceAllocation] at false' と 'encryption' は false です仮想ストレージプールは、ストレージセクションで定義します。

この例では ' 一部のストレージ・プールで独自の 'aceReserve'`paceAllocation] ' および [encryption`] 値が設定されていますまた ' 一部のプールでは ' 上で設定したデフォルト値が上書きされます

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "useCHAP": true,
    "chapInitiatorSecret": "cl9qxIm36DKyawxy",
    "chapTargetInitiatorSecret": "rqxigXgkesIpwxyz",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret",

    "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"
            }
        }
    ]
}
```

次に 'ONTAP-SAN-エコノミー ・ドライバの iSCSI の例を示します

```
{
    "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",
    "igroupName": "trident",
    "username": "vsadmin",
    "password": "secret",

    "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"
            }
        }
    ]
}
```

バックエンドを **StorageClasses** にマッピングします

次の StorageClass 定義は、上記の仮想ストレージプールを参照してください。parameters.selector` フィールドを使用すると ' 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ボリュームには、選択した仮想プール内で定義された要素があります。

- 最初の StorageClass （「 protection-gold 」）は、「 ontap/na-slexgroup 」バックエンドの最初の 2 番目の仮想ストレージプールと「 ontap/san' バックエンドの最初の仮想ストレージプールにマッピングされます。ゴールドレベルの保護を提供している唯一のプールです。
- 2 番目の StorageClass （「 protection-not-gold 」）は、「 ONTAP-NAS-flexgroup 」バックエンドの第 3 の仮想ストレージプールと「 ONTAP-SAN' バックエンドの第 2 の第 3 の仮想ストレージプールにマッピングされます。金色以外の保護レベルを提供する唯一のプールです。
- 3 番目の StorageClass （「 app-mysqldb 」）は、「 ONTAP-NAS' バックエンドの 4 番目の仮想ストレージプールと「 ONTAP-SAN-エコノミー 」バックエンドの 3 番目の仮想ストレージプールにマッピングされます。mysqldb タイプのアプリケーション用のストレージプール設定を提供しているプールは、これらだけです。
- 4 番目の StorageClass （「 protection-silver - creditpoints-20K 」）は、「 ONTAP-NAS-flexgroup 」バックエンドの 3 番目の仮想ストレージプールと「 ONTAP-SAN' バックエンドの 2 番目の仮想ストレージプールにマッピングされます。ゴールドレベルの保護を提供している唯一のプールは、 20000 の利用可能なクレジットポイントです。
- 5 番目の StorageClass （「 creditpoints-5k 」）は、「 ONTAP-NAS-エコノミー 」バックエンドの 2 番目の仮想ストレージプール、「 ONTAP-SAN」 バックエンドの 3 番目の仮想ストレージプールにマッピングされます。5000 ポイントの利用可能な唯一のプールは以下のとおりです。

Trident が、どの仮想ストレージ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: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
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"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

バックエンドに **ONTAP NAS** ドライバを設定します

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。

- "準備"
- "設定と例"

ユーザ権限

Astra Trident は、 ONTAP 管理者または SVM 管理者のいずれかとして実行されることを想定しています。通常は、「 admin 」クラスタユーザまたは「 vsadmin 」 SVM ユーザを使用するか、同じロールを持つ別の名前のユーザを使用します。Amazon FSX for NetApp ONTAP 環境では、 Astra Trident は、 ONTAP 管理者または SVM 管理者として、クラスタ「 fsxadmin 」ユーザまたは「 vsadmin 」 SVM ユーザ、または同じロールを持つ別の名前のユーザを実行する必要があります。「 fsxadmin 」ユーザは、クラスタ管理ユーザの限定的な代替ユーザです。

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

ONTAP 内では、 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。

準備

ONTAP NAS ドライバを使用して ONTAP バックエンドを設定するための準備方法について説明します。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。

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 'ONTAP-NAS' ドライバを使用する Gold クラスと 'ONTAP-NAS-エコノミー 'one を使用する Bronze クラスを構成できます

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：

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。ONTAP バージョンとの互換性を最大限に高めるために 'admin' または vsadmin などの事前定義されたセキュリティ・ログイン・ロールを使用することを推奨します
- 証明書ベース： 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。

また、既存のバックエンドを更新したり、クレデンシャルベースから証明書ベースに移行したり、その逆に移行したりすることもできます。クレデンシャルと証明書の両方が * 提供されている場合、 Astra Trident は、バックエンド定義からクレデンシャルを削除するように警告を発行しながら、デフォルトで証明書を使用します。

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。「 admin 」や「 vsadmin 」など、事前定義された標準的な役割を使用することをお勧めします。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。

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

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

証明書ベースの認証を有効にします

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

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

一般的なワークフローは次の手順で構成されます。

手順

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

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

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

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

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

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

4. ONTAP セキュリティ・ログイン・ロールが 'cert' 認証方式をサポートしていることを確認します

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

5. 生成された証明書を使用して認証をテストONTAP 管理 LIF > と <vserver name> は、管理 LIF の IP アドレスおよび SVM 名に置き換えてください。LIF のサービスポリシーが「 default-data-management 」に設定されていることを確認する必要があります。

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

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

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

7. 前の手順で得た値を使用してバックエンドを作成します。

```
$ cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaallllluuuueeee",
"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、別の認証方式を使用したり、クレデンシャルをローテーションしたりすることができます。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 ' tridentctl backend update を実行するために必要なパラメータを含む更新された backend.json ファイルを使用します

```
$ cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "secret",
"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、ストレージクラスタへのアクセスを、指定した範囲の IP を持つワーカーノードだけに制限することで、きめ細かな管理と自動化をサポートします。

> 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": "FaKePaSsWoRd",
    "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）を含む事前に作成され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 」は「 true 」に設定されています。これは、 Astra Trident が「 vm1 」 SVM のエクスポートポリシーを作成し、「 autoExportCIDRs 」アドレスブロックを使用してルールの追加と削除を処理することを示しています。たとえば、 UUID 403b5326-842-40dB-96d0-d83fb3f4daec と「 autoExportPolicy 」が「 true 」に設定されているバックエンドは、 SVM 上に「 trident-403b5326-842-40dB-96d0-d83f3f4daec 」という名前のエクスポートポリシーを作成します。
- 「 autoExportCI` 」には、アドレスブロックのリストが含まれています。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 '192.168.0.0/24 アドレス空間が提供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、「 autoExportCIDRs 」で提供されているアドレスブロックと照合します。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。

バックエンドの作成後に 'autoExportPolicy' および 'autoExportCIDRs を更新できます自動的に管理されるバックエンドに新しい CIDRs を追加したり、既存の CIDRs を削除したりできます。CIDRs を削除する際は、既存の接続が切断されないように注意してください。バックエンドに対して「 autoExportPolicy 」を無効にし、手動で作成したエクスポートポリシーに戻すこともできます。これには、バックエンド構成で「 exportPolicy 」パラメータを設定する必要があります。

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

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

ノードが Kubernetes クラスタに追加されて Astra Trident コントローラに登録されると、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（バックエンドの「 autoExportCIDRs 」に指定されたアドレス範囲に含まれる場合）。

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 'tridentctl update backend でバックエンドを更新することで 'Astra Trident がエクスポートポリシーを自動的に管理できるようになりますこれにより、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。

設定オプションと例

ONTAP NAS 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
「バージョン」		常に 1
'torageDriverName'	ストレージドライバの名前	「 ONTAP-NAS 」、「 ONTAP-NAS-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
「管理 LIF 」	クラスタ管理 LIF または SVM 管理 LIF の IP アドレス	「 10.0.0.1 」、「 [2001:1234:abcd::fefe] 」
「重複排除	プロトコル LIF の IP アドレス。IPv6 には角かっこを使用します。設定後に更新することはできません	特に指定がないかぎり、 SVM が派生します
「 autoExportPolicy 」を参照してください	エクスポートポリシーの自動作成と更新を有効にする [ブーリアン]	いいえ
「 autoExportCI` 」	「 autoExportPolicy 」が有効な場合に、 Kubernetes のノード IP をフィルタリングするための CIDR のリスト	[0.0.0.0/0] 、 [::/0]
「ラベル」	ボリュームに適用する任意の JSON 形式のラベルのセット	「」
「 clientCertificate 」をクリックします	クライアント証明書の Base64 エンコード値。証明書ベースの認証に使用されます	「」

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	「」
「 trustedCacertifate 」	信頼された CA 証明書の Base64 エンコード値。任意。証明書ベースの認証に使用されます	「」
「ユーザ名」	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます	
「 password 」と入力します	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます	
'VM'	使用する Storage Virtual Machine	SVM 「管理 LIF 」が指定されている場合に生成されます
「 igroupName 」と入力します	SAN ボリュームで使用する igroup の名前	"trident-<backend-UUID> "
'toragePrefix'	SVM で新しいボリュームをプロビジョニングする際に使用するプレフィックスを指定します。設定後に更新することはできません	Trident
「 AggreglimitateUsage 」と入力します	使用率がこの割合を超えている場合は、プロビジョニングが失敗します。* Amazon FSX for ONTAP * には適用されません	“”（デフォルトでは適用されません）
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、プロビジョニングが失敗します。	“”（デフォルトでは適用されません）
'lunsPerFlexvol	FlexVol あたりの最大 LUN 数。有効な範囲は 50 、 200 です	100
「バグトレースフラグ」	トラブルシューティング時に使用するデバッグフラグ。例： {"API" ： false 、 "method" ： true}	null
「 nfsvMountOptions 」のように入力します	NFS マウントオプションをカンマで区切ったリスト	「」
qtreesPerFlexvol`	FlexVol あたりの最大 qtree 数。有効な範囲は [50 、 300] です。	200
「 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 *	いいえ

「 useREST 」は **tech preview** として提供されています。これはテスト環境では推奨され、本番ワークロードでは推奨されません。「 true 」に設定すると、 Astra Trident は ONTAP REST API を使用してバックエンドと通信します。この機能を使用するには、 ONTAP 9.9 以降が必要です。また ' 使用する ONTAP ログイン・ロールには ONTAP アプリケーションへのアクセス権が必要ですこれは ' 事前に定義された vsadmin ロールと cluster-admin ロールによって満たされます

ONTAP クラスタと通信するには、認証パラメータを指定する必要があります。これは、セキュリティログインまたはインストールされている証明書のユーザ名 / パスワードです。

NetApp ONTAP バックエンドに Amazon FSX を使用している場合は、「 limitAggregateUsage 」パラメータを指定しないでください。Amazon FSX for NetApp ONTAP が提供する「 fsxadmin 」と「 vsadmin 」の役割には、集計の使用状況を取得したり、 Astra Trident を介して制限したりするために必要なアクセス権限が含まれていません。

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

バックエンドを作成するときは、作成後に「 dataLIF' 」と「 toragePrefix 」を変更できないことに注意してください。これら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。

「管理 LIF 」オプションには完全修飾ドメイン名（ FQDN ）を指定できます。「 atalif 」オプションに FQDN を指定した場合も、 NFS のマウント処理に FQDN が使用されます。こうすることで、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。

すべての ONTAP ドライバ用の「管理 LIF 」を IPv6 アドレスに設定することもできます。Astra Trident には '--use-ipv6' フラグを付けてインストールしてください角かっこで囲まれた「管理 LIF 」 IPv6 アドレスを定義するように注意する必要があります。

IPv6 アドレスを使用する場合は、 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555] のように、バックエンド定義に含まれている場合は「 anagementlif` 」と「 datalif 」が角かっこ内に定義されていることを確認してください。「 data lif 」を指定しない場合、 Astra Trident は SVM から IPv6 データ LIF を取得します。

CSI Trident では、「 autoExportPolicy 」オプションおよび「 autoExportCIDRs 」オプションを使用して、エクスポートポリシーを自動的に管理できます。これはすべての ONTAP-NAS-* ドライバでサポートされています。

「 ONTAP-NAS-エコノミー 」ドライバの場合、「 limitVolumeSize 」オプションを使用すると、 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されます。「 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。

マウントオプションを指定するには 'nfsMountOptions' パラメータを使用しますKubernetes 永続ボリュームのマウントオプションは通常ストレージクラスで指定されますが、ストレージクラスでマウントオプションが指定されていない場合、 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。ストレージクラスまたは構成ファイルにマウントオプションが指定されていない場合、 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。

Astra Trident は 'ONTAP-NAS' および 'ONTAP-NAS-flexgroup を使用して作成されたすべてのボリュームの Comments フィールドにプロビジョニングラベルを設定します使用するドライバに基づいて ' コメントは FlexVol (`ONTAP-NAS') または FlexGroup (`ONTAP-NAS-flexgroup ') に設定されますTrident が、ストレージプール上にあるすべてのラベルを、プロビジョニング時にストレージボリュームにコピーします。ストレージ管理者は、ストレージプールごとにラベルを定義し、ストレージプール内に作成されたすべてのボリュームをグループ化できます。これにより、バックエンド構成で提供されるカスタマイズ可能な一連のラベルに基づいてボリュームを簡単に区別できます。

ボリュームのプロビジョニング用のバックエンド構成オプション

これらのオプションを使用して、構成の特別なセクションで各ボリュームをデフォルトでプロビジョニングする方法を制御できます。例については、以下の設定例を参照してください。

パラメータ	説明	デフォルト
「平和の配分」	space-allocation for LUN のコマンドを指定します	正しいです
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シン）または「 volume 」（シック）	なし
「ナプショットポリシー」	使用する Snapshot ポリシー	なし
「 QOSPolicy 」	作成したボリュームに割り当てる QoS ポリシーグループ。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します	「」
「 adaptiveQosPolicy 」を参照してください	アダプティブ QoS ポリシーグループ：作成したボリュームに割り当てます。ストレージプール / バックエンドごとに QOSPolicy または adaptiveQosPolicy のいずれかを選択します。経済性に影響する ONTAP - NAS ではサポートされません。	「」
「スナップショット予約」	スナップショット "0" 用に予約されたボリュームの割合	「 napshotPolicy 」が「 none 」の場合、それ以外の場合は「」
'plitOnClone	作成時にクローンを親からスプリットします	いいえ
「暗号化」	ネットアップのボリューム暗号化を有効にします	いいえ
'ecurityStyle'	新しいボリュームのセキュリティ形式	「 UNIX 」
階層ポリシー	「なし」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	777

パラメータ	説明	デフォルト
Snapshot ディレクトリ	「 .snapshot 」ディレクトリの表示を制御します	いいえ
エクスポートポリシー	使用するエクスポートポリシー	デフォルト
securityStyle の追加	新しいボリュームのセキュリティ形式	「 UNIX 」

> 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。

次に、デフォルトが定義されている例を示します。

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

「 ONTAP-NAS' 」と「 ONTAP-NAS-flexgroups' では、 Astra Trident は新しい計算を使用して、 FlexVol がスナップショット予約の割合と PVC で正しくサイズ設定されるようにします。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が「 SnapshotReserve 」であるためです。Trident 21.07 では、ユーザが要求するのは書き込み可能なスペースであり、 Astra Trident は「 napshotReserve 」の値をボリューム全体の割合で定義します。これは「 ONTAP-NAS-エコノミー 」には適用されません。この機能の仕組みについて

は、次の例を参照してください。

計算は次のとおりです。

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

snapshotReserve = 50% 、 PVC 要求 = 5GiB の場合、ボリュームの合計サイズは 2/0.5 = 10GiB であり、使用可能なサイズは 5GiB であり、これが PVC 要求で要求されたサイズです。volume show コマンドは ' 次の例のような結果を表示する必要があります

```
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、「 napshotReserve 」が 50 であった 2GiB PVC の場合、ボリュームは書き込み可能なスペースが 1GiB であると考えられていました。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。

`ontap-nas` 証明書ベースの認証を使用するドライバ

これは、バックエンドの最小限の設定例です。「 clientCertificate` 」、「 clientPrivateKey 」、「 trustedCACertifate` （信頼された CA を使用する場合はオプション）」は「 backend.json 」に格納され、それぞれクライアント証明書、秘密鍵、信頼された CA 証明書の Base64 でエンコードされた値を取得します。

```
{
  "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_"
}
```

`ontap-nas` ドライバと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「 ONTAP-NAS-エコノミー 」と「 ONTAP-NAS-flexgroup 」ドライバで同様に機能します。

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

`ontap-nas-flexgroup` ドライバ

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "labels": {"k8scluster": "test-cluster-east-1b", "backend": "test1-
ontap-cluster"},
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
}
```

`ontap-nas` **IPv6**対応ドライバ

```
{
 "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": "netapp123"
}
```

`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": "secret"
}
```

仮想ストレージプールを使用するバックエンドの例

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たとえば 'paceReserve at none'`paceAllocation] at false' と 'encryption' は false です仮想ストレージプールは、ストレージセクションで定義します。

この例では ' 一部のストレージ・プールで独自の 'aceReserve'`paceAllocation] ' および [encryption`] 値が設定されていますまた ' 一部のプールでは ' 上で設定したデフォルト値が上書きされます

`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": "admin",
    "password": "secret",
    "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":{"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` 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": "secret",

    "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-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": "secret",

    "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":{"department":"legal", "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、上記の仮想ストレージプールを参照してください。parameters.selector` フィール

ドを使用すると ' 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ボリュームには、選択した仮想プール内で定義された要素があります。

- 最初の StorageClass （「 protection-gold 」）は、「 ontap/na-slexgroup 」バックエンドの最初の 2 番目の仮想ストレージプールと「 ontap/san' バックエンドの最初の仮想ストレージプールにマッピングされます。ゴールドレベルの保護を提供している唯一のプールです。
- 2 番目の StorageClass （「 protection-not-gold 」）は、「 ONTAP-NAS-flexgroup 」バックエンドの第 3 の仮想ストレージプールと「 ONTAP-SAN' バックエンドの第 2 の第 3 の仮想ストレージプールにマッピングされます。金色以外の保護レベルを提供する唯一のプールです。
- 3 番目の StorageClass （「 app-mysqldb 」）は、「 ONTAP-NAS' バックエンドの 4 番目の仮想ストレージプールと「 ONTAP-SAN-エコノミー 」バックエンドの 3 番目の仮想ストレージプールにマッピングされます。mysqldb タイプのアプリケーション用のストレージプール設定を提供しているプールは、これらだけです。
- 4 番目の StorageClass （「 protection-silver - creditpoints-20K 」）は、「 ONTAP-NAS-flexgroup 」バックエンドの 3 番目の仮想ストレージプールと「 ONTAP-SAN' バックエンドの 2 番目の仮想ストレージプールにマッピングされます。ゴールドレベルの保護を提供している唯一のプールは、 20000 の利用可能なクレジットポイントです。
- 5 番目の StorageClass （「 creditpoints-5k 」）は、「 ONTAP-NAS-エコノミー 」バックエンドの 2 番目の仮想ストレージプール、「 ONTAP-SAN」 バックエンドの 3 番目の仮想ストレージプールにマッピングされます。5000 ポイントの利用可能な唯一のプールは以下のとおりです。

Trident が、どの仮想ストレージ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: protection-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: netapp.io/trident
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: netapp.io/trident
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
---
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"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: netapp.io/trident
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

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。Amazon FSX for NetApp 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際の簡易性、即応性、セキュリティ、拡張性を活用できます。FSX 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。

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。

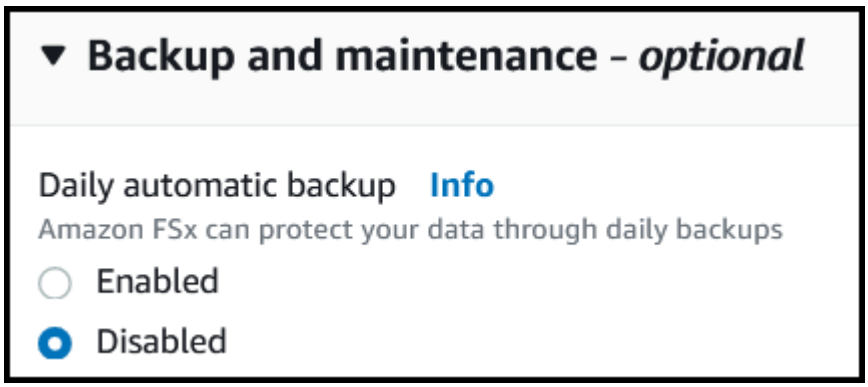
Amazon Elastic Kubernetes Service （ EKS ）で実行されている Astra Trident と Amazon FSX for NetApp ONTAP を使用すると、 ONTAP がサポートするブロックボリュームとファイル永続ボリュームを確実にプロビジョニングできます。

ONTAP ファイルシステム用に **Amazon FSX** を作成します

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。



Astra Trident の詳細をご確認ください

Astra Trident を初めて使用する場合は、以下のリンクを使用して確認してください。

- "よくある質問です"
- "Astra Trident を使用するための要件"
- "Astra Trident を導入"
- "ネットアップ ONTAP 用に ONTAP 、 Cloud Volumes ONTAP 、 Amazon FSX を設定する際のベストプ

ラクティス"

- "Astra Trident を統合"
- "ONTAP SAN バックエンド構成"
- "ONTAP NAS バックエンド構成"

ドライバー機能の詳細をご覧ください "こちらをご覧ください"。

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。

Astra Tridentは、「vsadmin」のSVMユーザとして、または同じロールを持つ別の名前を持つユーザとして実行されることを想定しています。Amazon FSX for NetApp ONTAP には'fsxadmin'ユーザーがいますこれはONTAP のadminクラスタ・ユーザーの限定的な置き換えです「vsadmin」のSVMユーザがより多くのAstra Trident機能にアクセスできるため、「fsxadmin」ユーザをTridentとともに使用することはお勧めしません。

ドライバ

Astra Trident と Amazon FSX for NetApp ONTAP を統合するには、次のドライバを使用します。

- 「 ONTAP-SAN」 ：プロビジョニングされる各 PV は、 NetApp ONTAP ボリューム用の独自の Amazon FSX 内の LUN です。
- 「 ONTAP と SAN の経済性」：プロビジョニングされた各 PV は、 NetApp ONTAP ボリュームの Amazon FSX ごとに構成可能な数の LUN を持つ LUN です。
- 「 ONTAP-NAS' ：プロビジョニングされた各 PV は、 NetApp ONTAP ボリューム用の完全な Amazon FSX です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree であり、 NetApp ONTAP ボリュームの Amazon FSX ごとに設定可能な数の qtree があります。
- 「 ONTAP-NAS-flexgroup 」：プロビジョニングされた各 PV は、 NetApp ONTAP FlexGroup ボリューム用の完全な Amazon FSX です。

認証

Astra Trident には、次の 2 つの認証モードがあります。

- 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。
- 認証情報ベース：ファイルシステムには「 fsxadmin 」ユーザを、 SVM には「 vsadmin 」ユーザを使用できます。

> バックエンドの構成には'fsxadmin'ではなく'vsadminユーザーを使用することを強くお勧めしますAstra Trident は、このユーザ名とパスワードを使用して FSX ファイルシステムと通信します。

認証の詳細については、次のリンクを参照してください。

- "ONTAP NAS"
- "ONTAP SAN"

Amazon FSX for NetApp ONTAP を使用して、 **EKS** に **Astra Trident** を導入して設定する

必要なもの

- 既存の Amazon EKS クラスタまたは 'kubectl' がインストールされた自己管理型 Kubernetes クラスタ
- クラスタのワーカーノードからアクセスできる、 NetApp ONTAP ファイルシステムと Storage Virtual Machine （ SVM ）用の既存の Amazon FSX 。
- 準備されているワーカーノード "NFS か iSCSI か"。

> 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 のその他の要件については、を参照してください "こちらをご覧ください"。

手順

1. ./trident-get-started/Kubernetes -deployment.html のいずれかを使用して Astra Trident を導入します（導入方法 ^ ）。
2. Trident を設定する手順は次のとおりです。
   a. SVM の管理 LIF の DNS 名を収集します。たとえば、 AWS CLI を使用して、次のコマンドを実行した後、「 Endpoints 」 → 「 Manager 」の下にある「 D NSName 」エントリを探します。

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

3. 認証用の証明書を作成してインストールします。「 ONTAP-SAN' バックエンド」を使用している場合は、を参照してください "こちらをご覧ください"。「 ONTAP-NAS' バックエンド」を使用している場合は、を参照してください "こちらをご覧ください"。

> 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。「 fsxadmin 」ユーザ、ファイルシステムの作成時に設定したパスワード、「 aws FSX describe -file-systems 」の管理 DNS 名を使用します。

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。

```
{
  "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。

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

> 'ONTAP-SAN' および 'ONTAP-SAN-エコノミー のドライバには 'atalif' を指定しないでください Astra Trident がマルチパスを使用できるようにします

> 「 limitAggregateUsage 」パラメータは、「 vsadmin 」および「 fsxadmin 」ユーザアカウントでは機能しません。このパラメータを指定すると設定処理は失敗します。

導入後、次の手順を実行してを作成します "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"。

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"

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 カスタムリソース定義（ 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。これを行うには 'kubectl' または Kubernetes ディストリビューションに相当する CLI ツールを使用します

`TridentBackendConfig`

TridentBackendConfig` (`tbc`, `tbconfig`, tbackendconfig`) はフロントエンドであり、 "kubectl" を使って Astra Trident バックエンドを管理するための名前空間 CRD です。Kubernetes とストレージ管理者は、専用のコマンドラインユーティリティ（「 tridentctl 」）を使用せずに、 Kubernetes CLI を使用してバックエンドを直接作成および管理できるようになりました。

「 TridentBackendConfig 」オブジェクトを作成すると、次のようになります。

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

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

> TridentBackend CRS は 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、「 TridentBackendConfig 」オブジェクトを変更します。

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

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

の例を確認することもできます "Trident インストーラ" 目的のストレージプラットフォーム / サービスの設定例を示すディレクトリ。

「 PEC 」は、バックエンド固有の設定パラメータをとります。この例では ' バックエンドは 'ONTAP-SAN' ストレージ・ドライバを使用し ' ここで表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"。

「 PEC 」セクションには、「 credentials 」フィールドと「 eleetionPolicy 」フィールドも含まれています。これらのフィールドは、「 TridentBackendConfig 」 CR に新しく導入されました。

- `credentials` ：このパラメータは必須フィールドで、ストレージシステム / サービスとの認証に使用されるクレデンシャルが含まれています。ユーザが作成した Kubernetes Secret に設定されます。クレデンシャルをプレーンテキストで渡すことはできないため、エラーになります。
- `DeleetionPolicy`: 「 TridentBackendConfig 」が削除されたときに何が起こるかを定義します。次の 2 つの値のいずれかを指定できます。
  - 「削除」：これにより、「 TridentBackendConfig 」 CR とそれに関連付けられたバックエンドの両方

が削除されます。これがデフォルト値です。

- 「管理」：「 TridentBackendConfig 」 CR を削除しても、バックエンド定義は引き続き表示され、「 tridentctl 」で管理できます。削除ポリシーを「 retain 」に設定すると、ユーザは以前のリリース (21.04 より前) にダウングレードし、作成されたバックエンドを保持できます。このフィールドの値は、「 TridentBackendConfig 」が作成された後で更新できます。

> バックエンドの名前は 'PEC.backendName' を使用して設定されます指定しない場合、バックエンドの名前は「 TridentBackendConfig 」オブジェクト（ metadata.name ）の名前に設定されます。'PEC.backendName' を使用してバックエンド名を明示的に設定することをお勧めします

> tridentctl で作成されたバックエンドには 'TridentBackendConfig' オブジェクトが関連付けられていません「 TridentBackendConfig 」 CR を作成することで、「 kubectl 」を使用してこのようなバックエンドを管理できます。同一の構成パラメータ (`PEC.backendName``PEC.storagePrefix''PEC.storageDriverName') を指定するように注意する必要がありますAstra Trident は、新しく作成した「 TridentBackendConfig 」を既存のバックエンドに自動的にバインドします。

手順の概要

kubectl' を使用して新しいバックエンドを作成するには ' 次の手順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. 「 TridentBackendConfig 」オブジェクトを作成します。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。

バックエンドを作成したら、「 kubectl get tbc <tbc-name> -n <trident-namespac>` 」を使用してバックエンドのステータスを確認し、詳細を収集できます。

手順 1 ： Kubernetes Secret を作成します

バックエンドのアクセスクレデンシャルを含むシークレットを作成します。ストレージサービス / プラットフォームごとに異なる固有の機能です。次に例を示します。

```
$ kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: t@Ax@7q(>
```

次の表に、各ストレージプラットフォームの Secret に含める必要があるフィールドをまとめます。

ストレージプラットフォームのシークレットフィールド概要	秘密	**Field** 概要の略
Azure NetApp Files の特長	ClientID	アプリケーション登録からのクライアント ID
Cloud Volumes Service for GCP	private_key_id です	秘密鍵の ID 。CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Cloud Volumes Service for GCP	private_key を使用します	秘密鍵CVS 管理者ロールを持つ GCP サービスアカウントの API キーの一部
Element （ NetApp HCI / SolidFire ）	エンドポイント	テナントのクレデンシャルを使用する SolidFire クラスタの MVIP
ONTAP	ユーザ名	クラスタ / SVM に接続するためのユーザ名。クレデンシャルベースの認証に使用されます
ONTAP	パスワード	クラスタ / SVM に接続するためのパスワード。クレデンシャルベースの認証に使用されます
ONTAP	clientPrivateKey	クライアント秘密鍵の Base64 エンコード値。証明書ベースの認証に使用されます
ONTAP	chapUsername のコマンド	インバウンドユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapInitiatorSecret	CHAP イニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetUsername のコマンド	ターゲットユーザ名。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合
ONTAP	chapTargetInitiatorSecret	CHAP ターゲットイニシエータシークレット。useCHAP = true の場合は必須。「 ONTAP-SAN' 」と「 ONTAP-SAN-エコノミー 」の場合

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

手順2：を作成します `TridentBackendConfig` CR

これで「 TridentBackendConfig 」 CR を作成する準備ができました。この例では 'ONTAP-SAN' ドライバを使用するバックエンドは ' 次に示す TridentBackendConfig オブジェクトを使用して作成されます

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

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

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

これで「 TridentBackendConfig 」 CR が作成され、ステータスを確認できるようになりました。次の例を参照してください。

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

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

フェーズには次のいずれかの値を指定できます。

- 「 bound` ： TridentBackendConfig 」 CR はバックエンドに関連付けられており、バックエンドには「 TridentBackendConfig 」 CR の uid に設定された「 configRef 」が含まれています。
- Unbound ： "" を使用して表現されています「 TridentBackendConfig 」オブジェクトはバックエンドにバインドされません。新しく作成されたすべての TridentBackendConfig ' CRS は、デフォルトでこのフェーズに入ります。フェーズが変更された後、再度 Unbound に戻すことはできません。

- 「削除」：「 TridentBackendConfig 」 CR の「要素ポリシー」は「削除」に設定されています。「 TridentBackendConfig 」 CR が削除されると、「 Deleting 」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、「 TridentBackendConfig 」を削除すると、 Astra Trident はバックエンドと「 TridentBackendConfig 」 CR を削除します。
  - バックエンドに 1 つ以上の PVC が存在する場合は、削除状態になります。次に 'TridentBackendConfig'CR が削除フェーズに入りますバックエンドおよび TridentBackendConfig は、すべての PVC が削除された後にのみ削除されます。
- `lost` ：「 TridentBackendConfig 」 CR に関連付けられているバックエンドが誤って削除されたか、意図的に削除されました。「 TridentBackendConfig 」 CR には削除されたバックエンドへの参照があります。「 TridentBackendConfig 」 CR は、「 $eleetionPolicy 」の値に関係なく削除できます。
- Unknown`: Astra Trident は 'TridentBackendConfig'CR に関連付けられたバックエンドの状態または存在を判断できませんたとえば、 API サーバが応答していない場合や、「 tridentbackends.trident.netapp.io` CRD 」がない場合などです。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、「 TridentBackendConfig 」の YAML ／ JSON ダンプを取得することもできます。

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

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

backendInfo` には 'TridentBackendConfig'CR に応答して作成されたバックエンドの backendName' と backendUUID' が含まれます「 lastOperationStatus 」フィールドは、「 TridentBackendConfig 」 CR の最後の操作のステータスを表します。これは、ユーザーが起動する（例えば、ユーザーが「 PEC 」の何かを変更した）か、 Astra Trident によってトリガーされる（例えば、 Astra Trident の再起動時）ことができます。Success または Failed のいずれかです。「 phase 」は、「 TridentBackendConfig 」 CR とバックエンド間の関係のステータスを表します。上の例では 'phase' に値がバインドされていますこれは 'TridentBackendConfig'CR がバックエンドに関連付けられていることを意味します

イベントログの詳細を取得するには、「 kubectl -n trident describe describe tbc <tbc -cr-name> 」コマンドを実行します。

> tridentctl を使用して ' 関連付けられた TridentBackendConfig' オブジェクトを含むバックエンドを更新または削除することはできません「 tridentctl 」と「 TridentBackendConfig 」の切り替えに関連する手順を理解するには、次の手順に従います。 "こちらを参照してください"。

kubectl を使用してバックエンド管理を実行します

kubectl' を使用してバックエンド管理操作を実行する方法について説明します

バックエンドを削除します

「 TridentBackendConfig 」を削除すると、「ネットワークポリシー」に基づいて、 Astra Trident にバックエンドを削除 / 保持するように指示します。バックエンドを削除するには、「削除ポリシー」が「削除」に設定されていることを確認します。「 TridentBackendConfig 」だけを削除するには、「 $eleetionPolicy 」が「 retain 」に設定されていることを確認します。これにより ' バックエンドがまだ存在していることが保証され 'tridentctl' を使用して管理できます

次のコマンドを実行します。

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

Astra Trident は、 TridentBackendConfig が使用していた Kubernetes シークレットを削除しません。Kubernetes ユーザは、シークレットのクリーンアップを担当します。シークレットを削除するときは注意が必要です。シークレットは、バックエンドで使用されていない場合にのみ削除してください。

既存のバックエンドを表示します

次のコマンドを実行します。

```
$ kubectl get tbc -n trident
```

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

バックエンドを更新します

バックエンドを更新する理由はいくつかあります。

- ストレージシステムのクレデンシャルが変更されている。クレデンシャルを更新するには、「 TridentBackendConfig 」オブジェクトで使用される Kubernetes Secret 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 を介して直接更新できます

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

または、次のコマンドを実行して、既存の「 TridentBackendConfig 」 CR に変更を加えます。

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

バックエンドの更新に失敗した場合、バックエンドは最後の既知の設定のまま残ります。ログを表示して原因を確認するには、「 kubectl get tbc <tbc-name> -o yaml -n trident` または「 kubectl describe tbc <tbc-name> -n trident` 」を実行します。

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

tridentctl を使用してバックエンド管理を実行します

tridentctl を使用してバックエンド管理操作を実行する方法について説明します

バックエンドを作成します

を作成したら "バックエンド構成ファイル"を使用して、次のコマンドを実行します。

```
$ tridentctl create backend -f <backend-file> -n trident
```

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

```
$ tridentctl logs -n trident
```

構成ファイルの問題を特定して修正したら ' 再度 create コマンドを実行します

バックエンドを削除します

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
```

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

バックエンドを使用するストレージクラスを特定します

ここでは ' バックエンド・オブジェクトの tridentctl 出力と同じ JSON を使用して回答で実行できる質問の例を示しますこれには 'jq' ユーティリティが使用されますこのユーティリティをインストールする必要があります

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

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

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。「 TridentBackendConfig 」が導入されたことで、管理者はバックエンドを 2 つの方法で管理できるようになりました。これには、次のような質問があります。

- tridentctl を使用して作成したバックエンドは 'TridentBackendConfig' で管理できますか
- 「 TridentBackendConfig 」を使用して作成したバックエンドは、「 tridentctl 」を使用して管理できますか。

管理 `tridentctl` を使用してバックエンドを `TridentBackendConfig`

このセクションでは 'tridentBackendConfig' オブジェクトを作成して Kubernetes インターフェイスから直接 'tridentctl を使用して作成されたバックエンドの管理に必要な手順について説明します

これは、次のシナリオに該当します。

- 「 tridentBackendConfig 」を持たない既存のバックエンドは、「 tridentctl 」で作成されています。
- 「 tridentctl 」で作成された新しいバックエンドと、その他の「 TridentBackendConfig 」オブジェクトが存在します。

どちらの場合も、 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。

- tridentctl を使用して ' バックエンドを使用して作成したバックエンドを管理します
- tridentctl を使用して作成されたバックエンドを新しい TridentBackendConfig オブジェクトにバインドしますこれは ' バックエンドが tridentctl ではなく 'kubectl' を使用して管理されることを意味します

「 kubectl 」を使用して既存のバックエンドを管理するには、既存のバックエンドにバインドする「 TridentBackendConfig 」を作成する必要があります。その仕組みの概要を以下に示します。

1. Kubernetes Secret を作成します。シークレットには、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. 「 TridentBackendConfig 」オブジェクトを作成します。ストレージクラスタ / サービスの詳細を指定し、前の手順で作成したシークレットを参照します。同一の構成パラメータ (`PEC.backendName'`PEC.storagePrefix'`PEC.storageDriverName') を指定するように注意する必要があります'PEC.backendName' は ' 既存のバックエンドの名前に設定する必要があります

手順 **0** ：バックエンドを特定します

既存のバックエンドにバインドする「 TridentBackendConfig 」を作成するには、バックエンドの設定を取得する必要があります。この例では、バックエンドが次の JSON 定義を使用して作成されているとします。

```
$ tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+----------+
|          NAME        | STORAGE DRIVER |                  UUID
| STATE   | VOLUMES |
```

```
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+

$ cat ontap-nas-backend.json

{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.10.10.1",
    "dataLIF": "10.10.10.2",
    "backendName": "ontap-nas-backend",
    "svm": "trident_svm",
    "username": "cluster-admin",
    "password": "admin-password",

    "defaults": {
        "spaceReserve": "none",
        "encryption": "false"
    },
    "labels":{"store":"nas_store"},
    "region": "us_east_1",
    "storage": [
        {
            "labels":{"app":"msoffice", "cost":"100"},
            "zone":"us_east_1a",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "true",
                "unixPermissions": "0755"
            }
        },
        {
            "labels":{"app":"mysqldb", "cost":"25"},
            "zone":"us_east_1d",
            "defaults": {
                "spaceReserve": "volume",
                "encryption": "false",
                "unixPermissions": "0775"
            }
        }
    ]
}
```

手順 **1** ： **Kubernetes Secret** を作成します

次の例に示すように、バックエンドのクレデンシャルを含むシークレットを作成します。

```
$ cat tbc-ontap-nas-backend-secret.yaml

apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  passWord: admin-password

$ kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

手順**2**：を作成します `TridentBackendConfig` **CR**

次の手順では ' （この例のように）事前に存在する 'ONTAP-NAS-backend' に自動的にバインドされる 'TridentBackendConfig’CR を作成します次の要件が満たされていることを確認します。

- 「 'PEC.backendName' 」に同じバックエンド名が定義されています。
- 設定パラメータは元のバックエンドと同じです。
- 仮想ストレージプール（存在する場合）は、元のバックエンドと同じ順序で設定する必要があります。
- クレデンシャルは、プレーンテキストではなく、 Kubernetes Secret を通じて提供されます。

この場合、「 TridentBackendConfig 」は次のようになります。

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

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

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

「 TridentBackendConfig 」が作成された後、そのフェーズは「バインド」されている必要があります。また、既存のバックエンドと同じバックエンド名と UUID が反映されている必要があります。

```
$ kubectl -n trident get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
$ tridentctl get backend -n trident
+----------------------+----------------
+--------------------------------------+--------+---------+
|          NAME        | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+----------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+----------------------+----------------
+--------------------------------------+--------+---------+
```

これで ' バックエンドは 'tbc-ontap/nas-backend`TridentBackendConfig' オブジェクトを使用して完全に管理されます

管理 `TridentBackendConfig` を使用してバックエンドを `tridentctl`

tridentBackendConfig を使用して作成されたバックエンドを一覧表示するには 'tridentctl を使用しますまた、管理者は、「 TridentBackendConfig 」を削除し、「 pec.deletionPolicy` 」が「 re 」に設定されていることを確認することで、「 tridentctl 」を使用してこのようなバックエンドを完全に管理することもできます。

手順 **0** ：バックエンドを特定します

たとえば ' 次のバックエンドが TridentBackendConfig を使用して作成されたとします

```
$ kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

$ tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

出力からは、「 TridentBackendConfig 」が正常に作成され、バックエンドにバインドされていることがわかります（バックエンドの UUID を確認してください）。

手順**1**：確認します `deletionPolicy` がに設定されます `retain`

「ネットワークポリシー」の値を見てみましょう。これは「山」に設定する必要があります。これにより 'TridentBackendConfig'CR が削除されても ' バックエンドの定義は引き続き表示され 'tridentctl' で管理できます

```
$ kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
$ kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
$ kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

「削除ポリシー」が「再取得」に設定されていない限り、次の手順に進まないでください。

手順**2**：を削除します `TridentBackendConfig` **CR**

最後の手順は、「 TridentBackendConfig 」 CR を削除することです。「削除ポリシー」が「取得」に設定されていることを確認したら、削除を続行できます。

```
$ kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

$ tridentctl get backend ontap-san-backend -n trident
+--------------------+----------------
+--------------------------------------+--------+---------+
|       NAME         | STORAGE DRIVER |                  UUID
| STATE   | VOLUMES |
+--------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+--------------------+----------------
+--------------------------------------+--------+---------+
```

TridentBackendConfig オブジェクトを削除すると、 Astra Trident はバックエンド自体を削除せずに削除します。

ストレージクラスを管理する

ストレージクラスの作成、ストレージクラスの削除、および既存のストレージクラスの表示に関する情報を検索します。

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る必要があります。

ストレージクラスを削除する

Kubernetes からストレージクラスを削除するには、次のコマンドを実行します。

```
kubectl delete storageclass <storage-class>
```

「 <storage-class> 」は、ご使用のストレージクラスに置き換えてください。

このストレージクラス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 は ' 作成したボリュームにブランクの fsType を適用しますiSCSI バックエンドの場合は 'StorageClass に parameters.fsType を適用することをお勧めしますesixting StorageClasses を削除して 'parameters.fsType' を指定して作成し直す必要があります

既存のストレージクラスを表示します

- 既存の Kubernetes ストレージクラスを表示するには、次のコマンドを実行します。

```
kubectl get storageclass
```

- Kubernetes ストレージクラスの詳細を表示するには、次のコマンドを実行します。

```
kubectl get storageclass <storage-class> -o json
```

- 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。

- ストレージクラスの定義でアノテーションの「 torageclass.Kubernetes .io/is-default-class 」を true に設定して、デフォルトのストレージクラスを定義します。仕様に応じて、それ以外の値やアノテーションがない場合は false と解釈されます。

- 次のコマンドを使用して、既存のストレージクラスをデフォルトのストレージクラスとして設定できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 同様に、次のコマンドを使用して、デフォルトのストレージクラスアノテーションを削除できます。

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

また、このアノテーションが含まれている Trident インストーラバンドルにも例があります。

> クラスタには、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

これは 'tridentctl が 'Astra Trident バックエンド・オブジェクトに出力する JSON を使用して回答が実行できる質問の一例ですこれには 'jq' ユーティリティが使用されますこのユーティリティを最初にインストールする必要があります

```
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

Trident がボリュームを管理するための各種機能をご紹介します。

- "CSI トポロジを使用します"
- "スナップショットを操作します"
- "ボリュームを展開します"
- "ボリュームをインポート"

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"。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。

- 'VolumeBindingMode' が Immediate に設定されていると 'Astra Trident は ' トポロジーを認識せずにボリュームを作成しますボリュームバインディングと動的プロビジョニングは、 PVC が作成されるときに処理されます。これはデフォルトの「 VolumeBindingMode 」であり、トポロジ制約を適用しないクラスタに適しています。永続ボリュームは、要求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。
- VolumeBindingMode を「 WaitForFirstConsumer 」に設定すると、 PVC の永続ボリュームの作成とバインドは、 PVC を使用するポッドがスケジュールされて作成されるまで遅延されます。これにより、トポロジの要件に応じたスケジュールの制約を満たすようにボリュームが作成されます。

「 WaitForFirstConsumer 」バインディングモードでは、トポロジラベルは必要ありません。これは CSI トポロジ機能とは無関係に使用できます。

必要なもの

CSI トポロジを使用するには、次のものが必要です。

- 1.17 以降を実行する Kubernetes クラスタ。

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

- クラスタ内のノードには ' トポロジー認識を導入するラベルが必要です（ topology .Kubernetes .io/region および topology .Kubernetes .io/zone ）このラベル * 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、サポートする必要があるゾーンおよびリージョンのリストを表すオプションの「 upportedTopologies 」ブロックを伝送できます。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。

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "xxxxxxxxxxxx",
 "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"}
]
}
```

「 upportedTopologies 」は、バックエンドごとにリージョンとゾーンのリストを提供するために使用されます。これらのリージョンとゾーンは、 StorageClass で指定できる許容値のリストを表します。バックエンドで提供されるリージョンとゾーンのサブセットを含む 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。

また ' ストレージ・プールごとに 'upportedTopologies を定義することもできます次の例を参照してください。

```
{"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "nas-backend-us-central1",
"managementLIF": "172.16.238.5",
"svm": "nfs_svm",
"username": "admin",
"password": "Netapp123",
"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、「 re Gion 」および「 zone 」ラベルはストレージプールの場所を表しています。「 topology.efix/region' 」と「 topology.uns.io/zone 」は、ストレージプールの消費元を決定します。

手順 **2** ：トポロジを認識するストレージクラスを定義する

クラスタ内のノードに提供されるトポロジラベルに基づいて、トポロジ情報を含めるように StorageClasses を定義できます。これにより、作成された PVC 要求の候補となるストレージプール、および Trident によってプロビジョニングされたボリュームを使用できるノードのサブセットが決まります。

次の例を参照してください。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
- matchLabelExpressions:
- key: topology.kubernetes.io/zone
  values:
  - us-east1-a
  - us-east1-b
- key: topology.kubernetes.io/region
  values:
  - us-east1
parameters:
  fsType: "ext4"
```

上記の StorageClass 定義では、「 volumeBindingMode 」が「 WaitForFirstConsumer 」に設定されます。この StorageClass で要求された PVC は、ポッドで参照されるまで処理されません。また 'allowedTopology' は ' 使用するゾーンと領域を提供しますNetApp-SAN-us-east1StorageClass は、上で定義した「 -backend-us -east1` バックエンド」に PVC を作成します。

ステップ **3** ： **PVC** を作成して使用する

StorageClass を作成してバックエンドにマッピングすると、 PVC を作成できるようになりました。

以下の例「 PEC 」を参照してください。

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

このマニフェストを使用して PVC を作成すると、次のような結果になります。

```
$ kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
$ kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
$ kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

Trident でボリュームを作成して PVC にバインドするには、ポッド内の PVC を使用します。次の例を参照してください。

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

この podSpec は 'us-east1` 領域に存在するノード上のポッドをスケジュールするよう Kubernetes に指示し 'us-east1-a' または `us-east1-b' ゾーン内に存在する任意のノードから選択します

次の出力を参照してください。

```
$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
$ kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

バックエンドを更新して追加 `supportedTopologies`

既存のバックエンドは 'tridentctl backend update を使用して 'upportedTopologies のリストを含むように更新できますこれは、すでにプロビジョニングされているボリュームには影響せず、以降の PVC にのみ使用されます。

詳細については、こちらをご覧ください

- "コンテナのリソースを管理"
- "ノードセレクタ"
- "アフィニティと非アフィニティ"
- "塗料および耐性"

スナップショットを操作します

Astra Trident の 20.01 リリースから、 Kubernetes レイヤで PVS のスナップショットを作成できるようになりました。この Snapshot を使用して、 Astra Trident で作成されたボリュームのポイントインタイムコピーを管理し、追加のボリューム（クローン）の作成をスケジュールできます。ボリューム・スナップショットは 'ONTAP-NAS'`ONTAP-SAN’ONTAP-SAN’ONTAP-エコノミー "solidfire-san-'`solidfire-SAN’GCP - cvs` によってサポートされています 「 azure-NetApp-files 」ドライバを使用します。

> この機能は Kubernetes 1.17 （ベータ版）から提供され、 1.20 から GA になります。ベータ版から GA 版への移行に伴う変更点については、を参照してください "リリースのブログ"。GA への卒業とともに 'v1' API バージョンが導入され 'v1beta` スナップショットと下位互換性があります

必要なもの

- ボリューム Snapshot を作成するには、外部の Snapshot コントローラとカスタムリソース定義（ CRD ）を作成する必要があります。使用されている Kubernetes Orchestrator （例： Kubeadm 、 GKE 、 OpenShift ）の役割を担っています。

次のように、外部スナップショットコントローラとスナップショット作成 SSD を作成できます。

1. ボリューム Snapshot の作成：

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 目的のネームスペースにスナップショットコントローラを作成します。以下の YAML マニフェストを編集して名前空間を変更します。

> GKE 環境でオンデマンドボリュームスナップショットを設定する場合は、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。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

> CSI Snapshotter は、を提供します "webhook を検証しています" ユーザーが既存の v1beta1 スナップショットを検証し、有効なリソースオブジェクトであることを確認できるようにするため。検証中の webhook は、無効なスナップショットオブジェクトに自動的にラベルを付け、今後無効なオブジェクトが作成されないようにします。検証する webhook は Kubernetes Orchestrator によって導入されます。検証するウェブフックを手動で配備する手順を参照してください "こちらをご覧ください"。無効なスナップショットマニフェストの例を探します "こちらをご覧ください"。

以下に、スナップショットの操作に必要な構成要素と、スナップショットの作成方法および使用方法の例を示します。

手順**1**：を設定します `VolumeSnapshotClass`

ボリューム Snapshot を作成する前に、をセットアップします "d7ca7162c394dee752c35d07a92823da"。

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

「川」はアストラトライデントの CSI ドライバを指しています。「保持ポリシー」には「削除」または「保持」を指定できます。「 Retain 」に設定すると、「 VolumeSnapshot 」オブジェクトが削除されても、ストレージ・クラスタ上の基盤となる物理スナップショットは保持されます。

手順 **2** ：既存の **PVC** のスナップショットを作成します

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

スナップショットは 'PVC1' という名前の PVC 用に作成されており ' スナップショットの名前は PVC1-snap' に設定されています

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

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

これにより 'VolumeSnapshot' オブジェクトが作成されましたVolumeSnapshot は PVC に似ており、実際のスナップショットを表す「 VolumeSnapshotContent 」オブジェクトに関連付けられています。

「 PVC1-SNAP 」ボリューム Snapshot の「 VolumeSnapshotContent 」オブジェクトを指定することができます。

```
$ 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
.
.
```

「スナップショットコンテンツ名」は、このスナップショットを提供する VolumeSnapshotContent オブジェクトを識別します。' 使用準備完了 ' パラメータは ' スナップショットを使用して新しい PVC を作成できることを示します

手順 **3** ：ボリューム **Snapshot** から **PVC** を作成します

スナップショットを使用して PVC を作成する例は、次のとおりです。

```
$ 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 」は、「 PVC1-SNAP 」という名前のボリューム Snapshot をデータのソースとして使用して PVC を作成する必要があることを示します。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。

> 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。Astra 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。

詳細については、こちらをご覧ください

- "ボリューム Snapshot"
- "d7ca7162c394dee752c35d07a92823da"

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 ボリュームの拡張は 'ONTAP-SAN"ONTAP-SAN-エコノミー "olidfire-SAN' ドライバによってサポートされており 'Kubernetes 1.16 以降が必要です

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 'allowVolumeExpansion フィールドを true に設定します
- PVC 定義を編集し 'PEC.resources.requests.storage を更新して ' 新たに必要とされるサイズを反映します これは ' 元のサイズより大きくなければなりません
- 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 StorageClass の場合は 'allowVolumeExpansion パラメータを含めるように編集します

手順 **2** ：作成した **StorageClass** を使用して **PVC** を作成します

```
$ 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、ポッドが作成され、「 1-pvc 」が使用されます。

```
$  kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-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:    centos-pod
```

ステップ **4** ： **PV** を展開します

1Gi から 2Gi に作成された PV のサイズを変更するには、 PVC の定義を編集し、「 PEC.resources.request.storage 」を 2Gi に更新します。

```
$ kubectl edit pvc san-pvc
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 ...
```

手順 **5** ：拡張を検証する

PVC 、 PV 、 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 'ONTAP-NAS''ONTAP-NAS-Bエコノミー 'ONTAP-NAS-flexp'GCP-cvs'Azure-NetApp-files' バックエンドでプロビジョニングされた NFS PVS のボリューム拡張をサポートしています

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

NFS PV のサイズを変更するには ' まず 'allowVolumeExpansion フィールドを true に設定してボリュームを拡張できるようにストレージ・クラスを構成する必要があります

```
$ cat storageclass-ontapnas.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

このオプションを指定せずにすでにストレージ・クラスを作成している場合は 'kubectl Edit storageclass を使用して既存のストレージ・クラスを編集するだけで ' ボリュームの拡張が可能になります

手順 **2** ：作成した **StorageClass** を使用して **PVC** を作成します

```
$ cat pvc-ontapnas.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

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** を展開します

新しく作成した 20MiB PV のサイズを 1GiB に変更するには、 PVC を編集し、「 pec.resources.request.storage 」を 1GB に設定します。

```
$ kubectl edit pvc ontapnas20mb
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
...
```

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 を使用して、既存のストレージボリュームを Kubernetes PV としてインポートできます。

ボリュームインポートをサポートするドライバ

次の表は、ボリュームのインポートをサポートするドライバと、それらのアップグレードが導入されたリリースを示しています。

ドライバ	リリース。
「 ONTAP - NAS 」	19.04
「 ONTAP-NAS-flexgroup 」	19.04
「 olidfire -san 」	19.04
「 azure-NetApp-files 」と入力します	19.04

ドライバ	リリース。
「 gcp-cvs 」	19.04
「 ontap - san 」	19.04

ボリュームをインポートする理由

Trident にボリュームをインポートするユースケースはいくつかあります。

- アプリケーションのコンテナ化と既存のデータセットの再利用
- エフェメラルアプリケーション用のデータセットのクローンを使用する
- 障害が発生した Kubernetes クラスタの再構築
- 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する

インポートはどのように機能しますか。

Persistent Volume Claim （ PVC ；永続ボリューム要求）ファイルは、ボリュームインポートプロセスで PVC を作成するために使用されます。少なくとも、次の例に示すように、 PVC ファイルには name 、 namespace 、 accessModes 、および storageClassName フィールドが含まれている必要があります。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class
```

tridentctl クライアントは ' 既存のストレージ・ボリュームをインポートするために使用されますTrident は、ボリュームのメタデータを保持し、 PVC と PV を作成することで、ボリュームをインポートします。

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

ストレージボリュームをインポートするには、ボリュームが含まれている 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 Volume 、 CVS ボリュームパスなど）を指定します。ストレージボリュームは、読み取り / 書き込みアクセスを許可し、指定された Astra Trident バックエンドからアクセスできる必要があります。引数 -f 文字列は必須であり、 YAML または JSON PVC ファイルへのパスを指定します。

Astra Trident がインポートボリューム要求を受信すると、既存のボリュームサイズが決定され、 PVC で設定されます。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。再生ポリシーは、最初に PV 内の「そのまま」に設定されます。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「削除」の場合、 PV を削除するとストレージボリュームは削除されます。

--no-manage 引数を指定してボリュームをインポートすると、 Trident はオブジェクトのライフサイクルについて PVC または PV に対する追加の操作を実行しません。Trident は '--no-managed' オブジェクトの PV イベントと PVC イベントを無視するため 'PV を削除してもストレージ・ボリュームは削除されませんボリュームのクローンやサイズ変更など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。

Trident 19.07 以降では、 PVS の添付ファイルを処理し、ボリュームをインポートの一環としてマウントします。以前のバージョンの Astra Trident を使用しているインポートの場合、データパスに処理は存在しないため、ボリュームをマウントできるかどうかがボリュームインポートで検証されません。ボリュームのインポートに誤りがあった場合（ StorageClass が正しくない場合など）は、 PV の再利用ポリシーを「リカバリ」に変更し、 PVC と PV を削除してから、 volume import コマンドを再試行してリカバリできます。

`ontap-nas` および `ontap-nas-flexgroup` インポート

「 ontap/nas 」ドライバで作成される各ボリュームは、 ONTAP クラスタ上の FlexVol です。「 ontap/nas 」ドライバを使用して FlexVol をインポートする方法は同じです。ONTAP クラスタにすでに存在する FlexVol は 'ONTAP-NAS’PVC としてインポートできます同様に、 FlexGroup ボリュームは「 ONTAP-NAS-flexgroup 」 PVC としてインポートできます。

> Trident がインポートする ONTAP のタイプは RW である必要があります。DP 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。 Trident にボリュームを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。

> 「 ONTAP - NAS 」ドライバは、 qtree のインポートおよび管理を行うことができません。「 ONTAP-NAS' 」および「 ONTAP-NAS-flexgroup 」ドライバでは、ボリューム名の重複が許可されていません。

たとえば、「 ONTAP_NAS' 」という名前のバックエンドに「管理されたボリューム」という名前のボリュームをインポートするには、次のコマンドを使用します。

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

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

Trident が管理しない 'unmanaged_volume' （ ONTAP_NAS バックエンド上）という名前のボリュームをインポートするには ' 次のコマンドを使用します

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

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

--no-manage 引数を使用する場合、 Trident はボリュームの名前を変更したり、ボリュームがマウントされたかどうかを検証したりしません。ボリュームが手動でマウントされていない場合、ボリュームインポート処理は失敗します。

> UnixPermissions カスタムのボリュームをインポートするという既存のバグが修正されました。PVC 定義またはバックエンド構成に unixPermissions を指定し、必要に応じて Astra Trident にボリュームをインポートするように指示できます。

`ontap-san` インポート

Astra Trident は、 1 つの LUN を含む ONTAP SAN FlexVol をインポートすることもできます。これは 'ONTAP-SAN' ドライバと一致しています FlexVol は 'PVC ごとに FlexVol 内の 1 つの LUN に対して を作成しますtridentctl import コマンドは ' 他の場合と同じ方法で使用できます

- 「 ontap - san 」バックエンドの名前を含めます。

- インポートする必要がある FlexVol の名前を指定します。この FlexVol には、インポートが必要な LUN が 1 つしか含まれていないことに注意してください。
- 「 -f 」フラグとともに使用する必要がある PVC 定義のパスを指定します。
- PVC を管理するか、管理対象外にするかを選択します。デフォルトでは、 Trident によって PVC が管理され、バックエンドの FlexVol と LUN の名前が変更されます。アンマネージボリュームとしてインポートするには、「 --no-manage 」フラグを渡します。

> 管理対象外の「 ONTAP -SAN 」ボリュームをインポートする場合は、 FlexVol 内の LUN が「 lun0 」になっていて、必要なイニシエータを持つ igroup にマッピングされていることを確認する必要があります。Trident が管理対象のインポートに対して自動的に処理します。

次に、 Astra Trident が FlexVol をインポートし、 PVC 定義に関連付けます。Astra Trident は、 FlexVol の名前を「 pvc-<uuid> 」形式に変更し、 FlexVol 内の LUN を「 lun0 」に変更します。

> 既存の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場合は、最初にボリュームをクローニングしてからインポートを実行します。

例

「 ONTAP_SAN_DEFAULT' バックエンドにある「 ONTAP-SAN-managed 」 FlexVol をインポートするには、「 tridentctl import 」コマンドを次のように実行します。

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

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

> ONTAP ボリュームのタイプが RW であることが Astra Trident でインポートされる必要があります。DP 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。

`element` インポート

Trident を使用して、 NetApp Element ソフトウェア / NetApp HCI ボリュームを Kubernetes クラスタにインポートできます。必要なのは 'tridentctl import コマンドの引数として 'Astra Trident バックエンドの名前とボリュームおよび PVC ファイルの一意の名前です

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

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

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。ボリューム名が重複している場合、 Trident のボリュームインポートプロセスはエラーを返します。回避策として、ボリュームをクローニングし、一意のボリューム名を指定します。次に、クローンボリュームをインポートします。

`gcp-cvs` インポート

GCP の NetApp Cloud Volumes Service から作成されたボリュームをインポートするには、名前ではなくボリュームパスでボリュームを特定します。

"gcpcvs_YEppr" という名前のバックエンド上の "gcpcvss_cvs" ボリュームを "adimenthy-jolly -sw" のボリュームパスでインポートするには、次のコマンドを使用します。

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

ボリュームパスは、 / のあとのボリュームのエクスポートパスの部分です。たとえば、エクスポートパスが「 10.0.0.1:/adwiswify-jolly -swift 」の場合、ボリュームパスは「 adwiswy-jolly -swift 」です。

azure-netapp-files インポート

ボリューム・パスが importvol1' の 'azurenetappfiles_40517' というバックエンドにある azure-netapp-files' ボリュームをインポートするには ' 次のコマンドを実行します

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

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

ANF ボリュームのボリュームパスは、 / のあとのマウントパスにあります。たとえば ' マウント・パスが 10.0.0.2::/importvol1 の場合 ' ボリューム・パスは importvol1 になります

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。いずれかのバックエンドに「 ONTAP-NAS' 」、「 ONTAP-NAS-Bエコノミー 」、または「 ONTAP-NAS-flexgroup 」ドライバを使用している場合は、ワーカーノードに NFS ツールが必要です。それ以外の場合は iSCSI ツールが必要です。

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

NFS ツールまたは iSCSI ツールをインストールしたあとは、必ずワーカーノードをリブートしてください。リブートしないと、ボリュームをコンテナに接続できないことがあります。

NFS ボリューム

プロトコル	オペレーティングシステム	コマンド
NFS	RHEL/CentOS	'sudo yum install -y nfs-utils'
NFS	Ubuntu / Debian	'UDO apt-get install-y nfs-common

ブート時に NFS サービスが開始されていることを確認してください。

iSCSI ボリューム

iSCSI ボリュームを使用するときは、次の点に注意してください。

- 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 または CentOS バージョン 8.2 以降で「 olidfire-san 」ドライバを使用している場合は、 CHAP 認証アルゴリズムが /etc/iscsi/iscsid.conf で MD5 に設定されていることを確認してください。

```
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、インラインスペース再生を実行するために、 StorageClass で「 discard 」 mountOption を指定してください。を参照してください "RedHat のマニュアル"。

プロトコル	オペレーティングシステム	コマンド
iSCSI	RHEL/CentOS	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 」のように入力します 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ node.session.scan\).*/\1 = manual/ '/etc/iscsi/iscsid.conf 4. マルチパスを有効化： 'UDO mpathconf—enable --with _multipathd y — find _multipaths n 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 iscsid 」と「 multipathd 」が実行されていることを確認します。 'sudo systemctl enable — 現在 iscsid multipathd 6. 'iSCSI' を有効にして開始します 'sudo systemctl enable — 現在の iSCSI

プロトコル	オペレーティングシステム	コマンド
iSCSI	Ubuntu / Debian	1. 次のシステムパッケージをインストールします。 'UDO 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。 `d pkg -l open-iscsi` 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ 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_friendy_names yes find _multipaths no ｝ EOF sudo systemctl enable — 今では multipath-tools.service sudo service multipath-tools restart' ⓘ 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されていることを確認します。 'sudo systemctl status multipath-tools `sudo systemctl enable` －現在の `open-iscsi.service` 'udo systemctl status open-iscsi'

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

ベータ版の自動ワーカーノードの準備の詳細については、を参照してください "こちらをご覧ください"。

ワーカーノードの自動準備

Astra Trident は、 Kubernetes クラスタ内のノードに必要な「 nfs 」ツールと「 iscsi 」ツールを自動的にインストールできます。これは * ベータ版の機能 * であり、本番環境クラスタ向けの機能ではありません。現在、この機能は、 CentOS 、 RHEL 、および Ubuntu * を実行するノードで使用できます。

この機能のために 'Astra Trident には 'tridentctl を使用して導入されたインストール用の --enable-node-prep という新しいインストール・フラグが含まれていますTrident 演算子を使用する展開の場合は、ブール値オプションの「 enableNodePrep 」を使用します。

--enable-node-prep のインストール・オプションは 'Astra Trident に 'NFS および iSCSI のパッケージやサービスが 'Worker ノードにマウントされているときに確実にインストールされ ' 実行されるように指示しますこの機能は * ベータ版で、本番環境での使用が認められていない * 開発 / テスト環境で使用することを目的としています。

--enable-node-prep フラグが 'tridentctl' とともに導入された Astra Trident のインストールに含まれている場合は ' 次のようになります

1. インストールの一環として、 Astra Trident が実行するノードを登録します。
2. Persistent Volume Claim （ PVC ；永続的ボリューム要求）が行われると、 Astra Trident は管理対象のバックエンドの 1 つから PV を作成します。
3. ポッド内の PVC を使用するには、ポッドが稼働するノードに Astra Trident がボリュームをマウントする必要があります。Trident が、必要な NFS / iSCSI クライアントユーティリティをインストールし、必要なサービスがアクティブになっていることを確認します。これは、ボリュームがマウントされる前に実行します。

ワーカーノードの準備は、最初にボリュームをマウントしようとしたときに 1 回だけ実行されます。Astra Trident の外部では 'NFS' および iSCSI ユーティリティに変更が加えられていない限り ' 後続のすべてのボリューム・マウントは正常に実行されます

このようにして、 Astra Trident は、 Kubernetes クラスタ内のすべてのノードに、ボリュームのマウントと接続に必要なユーティリティを確実に提供します。NFS ボリュームの場合は、エクスポートポリシーでボリュームのマウントも許可する必要があります。Trident では、バックエンドごとにエクスポートポリシーを自動的に管理できます。また、エクスポートポリシーをアウトオブバンドで管理することもできます。

Astra Trident を監視

Astra Trident は、 Astra Trident のパフォーマンスを監視するために使用できる一連の Prometheus 指標エンドポイントを提供します。

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 のメトリックは '/metrics エンドポイントのターゲットポート 8001' に公開されていますこれらの指標は、 Trident のインストール時にデフォルトで * 有効になります。

必要なもの

- 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 のメトリックを使用するには、「 trident-csi 」サービスを監視し、「 m etrics 」ポートを監視する Prometheus ServiceMonitor を作成する必要があります。ServiceMonitor のサンプルは次のようになります。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
      release: prom-operator
  spec:
    jobLabel: trident
    selector:
      matchLabels:
        app: controller.csi.trident.netapp.io
    namespaceSelector:
      matchNames:
      - trident
    endpoints:
    - port: metrics
      interval: 15s
```

この ServiceMonitor 定義は 'trident-csi サービスから返されたメトリックを取得し ' 特にサービスの 'm etrics エンドポイントを探しますその結果、 Prometheus は Astra Trident の指標を理解するように設定されました。

kubelet は、 Astra Trident から直接利用できるメトリックに加えて、独自のメトリック・エンドポイントを通じて多くの「 kubeet_volume_* 」メトリックを公開しています。Kubelet では、接続されているボリュームに関する情報、およびポッドと、それが処理するその他の内部処理を確認できます。を参照してください "こちらをご覧ください"。

ステップ 3 ： PrompQL を使用して Trident 指標を照会する

PrompQL は、時系列データまたは表データを返す式を作成するのに適しています。

次に、 PrompQL クエリーのいくつかを示します。

Trident の健常性情報を取得

- **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 が Prometheus 指標と基本バックエンド情報をネットアップに送信しないようにするには、Astra Trident のインストール時に「 --silence -autosupport 」フラグを渡します。
- Astra Trident は 'tridentctl send AutoSupport を介してコンテナ・ログをオンデマンドでネットアップ・サポートに送信することもできます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https://www.netapp.com/company/legal/privacy-policy/["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。
- ログ保持期間は '--since ' フラグで指定できますたとえば 'tridentctl は AutoSupport --since =1h` を送信しますこの情報は、 Astra Trident と一緒にインストールされた 'trident-autosupport' コンテナを介して収集および送信されます。コンテナイメージは、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。それにはが付いていま

す "EULA" これは Trident コンテナイメージ自体には該当しません。ネットアップのデータセキュリティと信頼に対する取り組みの詳細を確認できます "こちらをご覧ください"。

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": null,
        "disableDelete": false,
        "serialNumbers": [
          "nwkvzfanek_SN"
        ],
        "limitVolumeSize": ""
      },
      "state": "online",
      "online": true
    }
  ]
}
```

- AutoSupport メッセージは、ネットアップの AutoSupport エンドポイントに送信されます。プライベートレジストリを使用してコンテナイメージを格納している場合は '--image_registry' フラグを使用できます
- インストール YAML ファイルを生成してプロキシ URL を設定することもできます。これは 'tridentctl install --generate-custom-yaml を使用して YAML ファイルを作成し 'trident-deployment.yaml の trident-autosupport コンテナに --proxy-url 引数を追加することによって実行できます

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 '--generate-custom-yaml' フラグを使用してカスタム YAML を生成し ' これらを編集して 'trident-main' コンテナに対して --metrics フラグが呼び出されないようにします

Trident 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。

- 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/CentOS	'sudo yum install -y nfs-utils'
NFS	Ubuntu / Debian	'UDO 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 / CentOS 7.	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 」のように入力します 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ node.session.scan\).*/\1 = manual/ '/etc/iscsi/iscsid.conf 4. マルチパスを有効化： 'UDO mpathconf—enable --with _multipathd y — find _multipaths n 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 iscsid 」と「 multipathd 」が実行されていることを確認します。 'sudo systemctl enable — 現在 iscsid multipathd 6. 'iSCSI' を有効にして開始します 'sudo systemctl enable — 現在の 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。 'UDO 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。 `d pkg -l open-iscsi` 3. スキャンを手動に設定： 'sudo sed-i'/ ^ \ （ 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_friendy_names yes find _multipaths no ｝ EOF sudo systemctl enable — 今では multipath-tools.service sudo service multipath-tools restart' ⓘ 「 /etc/multipath.conf 」に「 find _ multipaths no 」が「 defVaults 」に含まれていることを確認します。 5. 「 open-iSCSI」 および「マルチパスツール」が有効で実行されていることを確認します。 'sudo systemctl status multipath-tools `sudo systemctl enable` －現在の `open-iscsi.service` 'udo systemctl status open-iscsi'

Astra Trident を導入

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 'config.json ですが ' ファイル名に config オプションを指定することで ' 選択した任意の名前を使用できます構成ファイルは ' ホスト・システムの /etc/netappdvp ディレクトリに格納されている必要があります
   - 「 log-level 」：ログレベルを指定します（「 debug` 」、「 info 」、「 warn 」、「 error 」、「 fatal 」）。デフォルトは「 info 」です。

- ebug: デバッグログを有効にするかどうかを指定します。デフォルトは false です。true の場合、ログレベルを上書きします。

  i. 構成ファイルの場所を作成します。

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

  ii. 構成ファイルを作成します

  ```
  cat << EOF > /etc/netappdvp/config.json
  {
      "version": 1,
      "storageDriverName": "ontap-nas",
      "managementLIF": "10.0.0.1",
      "dataLIF": "10.0.0.2",
      "svm": "svm_nfs",
      "username": "vsadmin",
      "password": "secret",
      "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

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:21.07 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/v21.04.0/trident-
   installer-21.07.0.tar.gz
   tar zxf trident-installer-21.07.0.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": "secret",
    "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用のサンプルユニットファイルは、 Git repo の「 contrib / trident.service.example`」にあります。このファイルを CentOS / RHEL で使用するには、次の手順を実行します。

1. ファイルを正しい場所にコピーします。

   複数のインスタンスを実行している場合は、ユニットファイルに一意の名前を使用してください。

   ```
   cp contrib/trident.service.example
   /usr/lib/systemd/system/trident.service
   ```

2. ファイルを編集し、概要（２行目）を変更してドライバ名と構成ファイルのパス（９行目）を環境に合わせます。
3. 変更を取り込むためにシステムをリロードします。

   ```
   systemctl daemon-reload
   ```

4. サービスを有効にします。

   この名前は '/usr/lib/systemd/system' ディレクトリ内のファイルの名前によって異なります

   ```
   systemctl enable trident
   ```

5. サービスを開始します。

   ```
   systemctl start trident
   ```

6. ステータスを確認します。

   ```
   systemctl status trident
   ```

> ユニット・ファイルを変更するときは ' 変更を認識するために 'systemctl daemon-reload コマンドを実行します

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。アップグレード処理中に、プラグインで指示された「 Occker volume 」コマンドが正常に実行されず、プラグインが再度実行されるまでアプリケーションはボリュームをマウントできません。ほとんどの場合、これは秒の問題です。

アップグレード

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。NetApp/ndvp-plugin イメージから NetApp/trident-plugin` イメージに直接アップグレードする必要があります。

4. プラグインを有効にします。

```
docker plugin enable netapp:latest
```

5. プラグインが有効になっていることを確認します。

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. ボリュームが表示され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、必要な Astra Trident バージョンをインストールします。これには、「 d Occker plugin install NetApp/trident-plugin:20.10 --alias netapp --grant-all -permissions config=config.json 」という追加設定パラメータを渡します

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

必要に応じて Astra Trident ドライバ名を指定した標準の「 DOcker volume 」コマンドを使用すると、ボリュームの作成、クローニング、削除を簡単に行うことができます。

ボリュームを作成します

- デフォルトの名前を使用して、ドライバでボリュームを作成します。

```
docker volume create -d netapp --name firstVolume
```

- 特定の 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
```

> 「 olidfire -san 」ドライバを使用する場合、上記の例ではボリュームを削除してパージします。

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

「 ONTAP-NAS` 」、「 ONTAP-SAN」 、「 solidfire-san-」 、「 GCP-cvs ストレージドライバ」を使用する場合、 Astra Trident はボリュームのクローンを作成できます。「 ONTAP-NAS-flexgroup 」または「 ONTAP-NAS-エコノミー 」ドライバを使用する場合、クローニングはサポートされません。既存のボリュームから新しいボリュームを作成すると、新しい 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

```
[me@host ~]$ 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"
                }
            ]
        }
    }
]

[me@host ~]$ docker volume create -d ontap-nas --name clonedVolume -o
from=firstVolume
clonedVolume

[me@host ~]$ docker volume rm clonedVolume
[me@host ~]$ docker volume create -d ontap-nas --name volFromSnap -o
from=firstVolume -o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

[me@host ~]$ docker volume rm volFromSnap
```

外部で作成されたボリュームにアクセス

Trident * を使用して、外部で作成されたブロックデバイス（またはそれらのクローン）には、パーティションがなく、ファイルシステムが Astra Trident でサポートされている場合（例えば、 "ext4 ’でフォーマットされた "/dev/sdc1" は Astra Trident 経由でアクセスできません）のみ、コンテナからアクセスできます。

ドライバ固有のボリュームオプション

ストレージドライバにはそれぞれ異なるオプションがあり、ボリュームの作成時に指定することで結果をカスタマイズできます。構成済みのストレージシステムに適用されるオプションについては、以下を参照してください。

ボリューム作成処理では、これらのオプションを簡単に使用できます。CLI の操作中に '-o' 演算子を使用して

' オプションと値を指定しますこれらは、 JSON 構成ファイルの同等の値よりも優先されます。

ONTAP ボリュームのオプション

NFS と iSCSI のどちらの場合も、 volume create オプションには次のオプションがあります。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 1GiB です。
「平和のための準備」を参照してください	ボリュームをシンプロビジョニングまたはシックプロビジョニングします。デフォルトはシンです。有効な値は 'none(シン・プロビジョニング) と 'volume(シック・プロビジョニング) です
「ナプショットポリシー」	Snapshot ポリシーが目的の値に設定されます。デフォルトは「 none 」です。つまり、ボリュームに対してスナップショットが自動的に作成されることはありません。ストレージ管理者によって変更されていない限り、「 default 」という名前のポリシーがすべての ONTAP システムに存在し、 6 個の時間単位 Snapshot 、 2 個の日単位 Snapshot 、および 2 個の週単位 Snapshot を作成して保持します。スナップショットに保存されているデータは ' ボリューム内の任意のディレクトリ内の .snapshot ディレクトリに移動してリカバリできます
「スナップショット予約」	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 Snapshot ポリシーを選択した場合は ONTAP によって snapshotReserve が選択されます（通常は 5% ）。 Snapshot ポリシーがない場合は 0% が選択されます。構成ファイルのすべての ONTAP バックエンドに対して snapshotReserve のデフォルト値を設定できます。また、この値は、 ONTAP-NAS-エコノミー を除くすべての ONTAP バックエンドでボリューム作成オプションとして使用できます。
'plitOnClone	ボリュームをクローニングすると、そのクローンが原因 ONTAP によって親から即座にスプリットされます。デフォルトは「 false 」です。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。

オプション	説明
「暗号化」	これにより、新しいボリュームで NetApp Volume Encryption （ NVE ）がデフォルトで「 false 」に設定されます。このオプションを使用するには、クラスタで NVE のライセンスが設定され、有効になっている必要があります。 NetApp Aggregate Encryption （ NAE ）は、現在 Trident ではサポートされていません。
階層ポリシー	ボリュームに使用する階層化ポリシーを設定します。これにより、アクセス頻度の低いコールドデータをクラウド階層に移動するかどうかが決まります。

以下は、 NFS * のみ * 用の追加オプションです。

オプション	説明
「 unixPermissions 」	これにより、ボリューム自体の権限セットを制御できます。デフォルトでは ' アクセス権は '--rwxr-xr-x' または数値表記 0755 に設定され 'root' は所有者になりますテキスト形式または数値形式のどちらかを使用できます。
「スナップショット方向」	これを「 true 」に設定すると、ボリュームにアクセスしているクライアントから「 .snapshot 」ディレクトリを参照できるようになります。デフォルト値は 'false' ですつまり '.snapshot ディレクトリの表示はデフォルトで無効になっています公式の MySQL イメージなどの一部のイメージは '.snapshot ディレクトリが表示されているときに ' 期待どおりに機能しません
「 exportPolicy 」と入力します	ボリュームで使用するエクスポートポリシーを設定します。デフォルトは「デフォルト」です。
'ecurityStyle'	ボリュームへのアクセスに使用するセキュリティ形式を設定します。デフォルトは「 unix 」です。有効な値は「 unix 」と「 immimixed 」です。

以下の追加オプションは、 iSCSI * のみ * 用です。

オプション	説明
「 filesystemtype 」です	iSCSI ボリュームのフォーマットに使用するファイルシステムを設定します。デフォルトは「 ext4 」です。有効な値は「 ext3 」、「 ext4 」、「 xfs 」です。

オプション	説明
「平和の配分」	これを false に設定すると、 LUN のスペース割り当て機能がオフになります。デフォルト値は「 true 」です。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 はホストに通知します。また、このオプションで ONTAP 、ホストでデータが削除された時点での自動スペース再生も有効になります。

例

以下の例を参照してください。

- 10GiB ボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- Snapshot を使用して 100GiB のボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- setuid ビットが有効になっているボリュームを作成します。

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

最小ボリュームサイズは 20MiB です。

スナップショット予約が指定されておらず、スナップショットポリシーが「 none 」の場合、 Trident は 0% のスナップショット予約を使用します。

- Snapshot ポリシーがなく、 Snapshot リザーブがないボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- Snapshot ポリシーがなく、カスタムの Snapshot リザーブが 10% のボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- Snapshot ポリシーを使用し、カスタムの Snapshot リザーブを 10% に設定してボリュームを作成します。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- Snapshot ポリシーを設定してボリュームを作成し、 ONTAP のデフォルトの Snapshot リザーブ（通常は 5% ）を受け入れます。

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element ソフトウェアのボリュームオプション

Element ソフトウェアのオプションでは、ボリュームに関連付けられているサービス品質（ QoS ）ポリシーのサイズと QoS を指定できます。ボリュームが作成されると '-o type=service_level' という命名法を使用して ' ボリュームに関連付けられた QoS ポリシーが指定されます

Element ドライバを使用して QoS サービスレベルを定義する最初の手順は、少なくとも 1 つのタイプを作成し、構成ファイル内の名前に関連付けられた最小 IOPS 、最大 IOPS 、バースト IOPS を指定することです。

Element ソフトウェアのその他のボリューム作成オプションは次のとおりです。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルト値は 1GiB または設定エントリ … 「 defaults 」： ｛ 「 size 」：「 5G 」｝ 。
「ブロックサイズ」	512 または 4096 のいずれかを使用します。デフォルトは 512 または config エントリ DefaultBlockSize です。

例

QoS 定義を含む次のサンプル構成ファイルを参照してください。

```
{
    "...": "..."
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

上記の構成では、 Bronze 、 Silver 、 Gold の 3 つのポリシー定義を使用します。これらの名前は任意です。

- 10GiB の Gold ボリュームを作成します。

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 100GiB Bronze ボリュームを作成します。

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

CVS on GCP ボリュームのオプション

GCP 上の CVS ドライバのボリューム作成オプションには、次のものがあります。

オプション	説明
「 size 」	ボリュームのサイズ。 CVS パフォーマンスボリュームの場合はデフォルトで 100GiB 、 CVS ボリュームの場合は 300GiB になります。
「サービスレベル」	ボリュームの CVS サービスレベル。デフォルトは「 Standard 」です。有効な値は、 standard 、 premium 、 extreme です。
「スナップショット予約」	これにより、 Snapshot リザーブの割合が希望する値に設定されます。デフォルト値は no で、 CVS によって Snapshot リザーブが選択されます（通常は 0% ）。

例

- 2TiB のボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=2T
```

- 5TiB の Premium ボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=5T -o
serviceLevel=premium
```

CVS パフォーマンスボリュームの場合は最小ボリュームサイズが 100GiB 、 CVS ボリュームの場合は 300GiB です。

Azure NetApp Files ボリュームのオプション

Azure NetApp Files ドライバの volume create オプションには、次のものがあります。

オプション	説明
「 size 」	ボリュームのサイズ。デフォルトは 100GB です。

例

- 200GiB ボリュームを作成します。

```
docker volume create -d netapp --name demo -o size=200G
```

最小ボリュームサイズは 100GB です。

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。

手順

1. 推奨されるマネージプラグインメソッド (`d Occker plugin` コマンドを使用) を使用して Astra Trident を実行している場合は、次のように表示します。

```
# docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
# journalctl -u docker | grep 4fb97d2b956b
```

   標準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. ホストでバイナリ自体を実行している場合、ログはホストの「 /var/log/netappdvp 」ディレクトリにあります。デバッグ・ロギングを有効にするには ' プラグインの実行時に -debug を指定します

一般的なトラブルシューティングのヒント

- 新しいユーザーが実行する最も一般的な問題は、プラグインの初期化を妨げる構成ミスです。この場合、プラグインをインストールまたは有効にしようとすると、次のようなメッセージが表示されることがあります。

「デーモンからのエラー応答 : ダイヤル UNIM/run/docx/plugins/<id>/NetApp/smock: connect: no such file or directory`

これは、プラグインの起動に失敗したことを意味します。幸い、このプラグインには、発生する可能性の高い問題のほとんどを診断するのに役立つ包括的なログ機能が組み込まれています。

- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl ステータス rpcbind またはそれに相当する処理を実行することで確認できます

複数の **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の鍵は、コンテナ化されたプラグインでは「 --alias 」オプション、ホストで Trident をインスタンス化する場合は「 --volume-driver 」オプションを使用して、それぞれ異なる名前を指定することです。

Docker Managed Plugin （バージョン **1.13 / 17.03** 以降）の手順

1. エイリアスと構成ファイルを指定して、最初のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. 別のエイリアスと構成ファイルを指定して、 2 番目のインスタンスを起動します。

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. ドライバ名としてエイリアスを指定するボリュームを作成します。

   たとえば、 gold ボリュームの場合：

```
docker volume create -d gold --name ntapGold
```

   たとえば、 Silver ボリュームの場合：

```
docker volume create -d silver --name ntapSilver
```

従来の（バージョン **1.12** 以前）の場合の手順

1. カスタムドライバ ID を使用して NFS 設定でプラグインを起動します。

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. カスタムドライバ ID を使用して、 iSCSI 構成でプラグインを起動します。

```
sudo trident --volume-driver=netapp-san --config=/path/to/config
-iscsi.json
```

3. ドライバインスタンスごとに Docker ボリュームをプロビジョニングします。

   たとえば、 NFS の場合：

```
docker volume create -d netapp-nas --name my_nfs_vol
```

   たとえば、 iSCSI の場合：

```
docker volume create -d netapp-san --name my_iscsi_vol
```

ストレージ構成オプション

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
「バージョン」	構成ファイルのバージョン番号	1.
'torageDriverName'	ストレージドライバの名前	「 ontap -na`,`ontap -san`,`ontap -nas-エコノミー 」 ,「 ontap -nas -flexgroup 」 ,「 solidfire-san-', 」 「 azure-NetApp-files 」または「 gcp-cvs 」です
'toragePrefix'	ボリューム名のオプションのプレフィックス。デフォルト：「 netappdvp_」 。	ステージング _

オプション	説明	例
「 limitVolumeSize 」と入力します	ボリュームサイズに関するオプションの制限。デフォルト：「」（適用されていない）	10G

Element バックエンドには 'toragePrefix' （デフォルトを含む）を使用しないでくださいデフォルトでは 'solidfire-san' ドライバはこの設定を無視し ' 接頭辞を使用しませんDocker ボリュームマッピングには特定の tenantID を使用するか、 Docker バージョン、ドライバ情報、名前の munging が使用されている可能性がある場合には Docker から取得した属性データを使用することを推奨します。

作成するすべてのボリュームでデフォルトのオプションを指定しなくても済むようになっています。「 size 」オプションは、すべてのコントローラタイプで使用できます。デフォルトのボリュームサイズの設定方法の例については、 ONTAP の設定に関するセクションを参照してください。

オプション	説明	例
「 size 」	新しいボリュームのオプションのデフォルトサイズ。デフォルト：「 1G 」	10G

ONTAP の設定

ONTAP を使用する場合は、上記のグローバル構成値に加えて、次のトップレベルオプションを使用できます。

オプション	説明	例
「管理 LIF 」	ONTAP 管理 LIF の IP アドレス。Fully Qualified Domain Name （ FQDN ；完全修飾ドメイン名）を指定できます。	10.0.0.1
「重複排除	プロトコル LIF の IP アドレス。指定しない場合に生成されます。「 ontap/na`drivers * only * 」では、FQDN を指定できます。この場合、 FQDN が NFS マウント操作に使用されます。「 ONTAP -SAN 」ドライバの場合、デフォルトでは SVM のすべてのデータ LIF IP が使用され、 iSCSI マルチパスが使用されます。「 SAN-SAN 」ドライバの「 datalif 」に IP アドレスを指定すると、ドライバはマルチパスを無効にし、指定されたアドレスだけを使用するように強制されます。	10.0.0.2

オプション	説明	例
'VM'	使用する Storage Virtual Machine（管理 LIF がクラスタ LIF である場合は必須）	SVM_NFS の場合
「ユーザ名」	ストレージデバイスに接続するユーザ名	vsadmin
「 password 」と入力します	ストレージ・デバイスに接続するためのパスワード	秘密
「集約」	プロビジョニング用のアグリゲート（オプション。設定する場合は SVM に割り当てる必要があります）。「 ONTAP-NAS-flexgroup 」ドライバの場合、このオプションは無視されます。SVM に割り当てられたすべてのアグリゲートを使用して FlexGroup ボリュームがプロビジョニングされます。	aggr1
「 AggreglimitateUsage 」と入力します	オプション。使用率がこの割合を超えている場合は、プロビジョニングを失敗させます	75%
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御。デフォルトは「 -o nfsvers=3 」です。* 「 ONTAP-NAS' 」および「 ONTAP-NAS-エコノミー 」ドライバ専用です。 "ここでは、 NFS ホストの設定情報を参照してください"。	-o nfsvers=4
「 igroupName 」と入力します	プラグインで使用される igroup 。デフォルトは「 netappdvp 」です。* 「 ONTAP-SAN'd river 」のみ利用可能です。	myigroup と入力します
「 limitVolumeSize 」と入力します	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。* 「 ONTAP-NAS-エコノミー 」ドライバの場合、このオプションにより、作成する FlexVol のサイズも制限されます。 *	300g

オプション	説明	例
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。*「 ONTAP-NAS-エコノミー」ドライバの場合、このオプションを使用すると、 FlexVol あたりの最大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
「平和のための準備」を参照してください	スペースリザベーションモード：「 none 」（シンプロビジョニング）または「 volume 」（シック）	なし
「ナプショットポリシー」	使用する Snapshot ポリシー。デフォルトは「 none 」です。	なし
「スナップショット予約」	Snapshot リザーブの割合。ONTAP のデフォルトを受け入れる場合は、デフォルトで「」になります	10.
'plitOnClone	作成時にクローンを親からスプリットします。デフォルトは「 false 」です。	いいえ
「暗号化」	NetApp Volume Encryption を有効にします。デフォルトは「 false 」です。	正しいです
「 unixPermissions 」	プロビジョニングされた NFS ボリューム用の NAS オプション。デフォルトは「 777 」	777
「スナップショット方向」	「 .snapshot 」ディレクトリにアクセスするための NAS オプション。デフォルトは「 false 」です。	正しいです
「 exportPolicy 」と入力します	NFS エクスポートポリシーで使用する NAS オプション。デフォルトは「 default 」	デフォルト

オプション	説明	例
'ecurityStyle'	プロビジョニングされた NFS ボリュームにアクセスするための NAS オプション（デフォルトは「 UNIX 」）	混在
「 filesystemtype 」です	SAN オプション：ファイルシステムタイプを選択します。デフォルトは「 ext4 」です。	XFS
階層ポリシー	使用する階層化ポリシー。デフォルトは「 none 」です。 ONTAP 9.5 より前の SVM-DR 構成では「 snapshot-only 」です	なし

スケーリングオプション

「 ONTAP-NAS' 」ドライバと「 ONTAP-SAN' 」ドライバは、各 Docker ボリューム用の ONTAP FlexVol を作成します。ONTAP では、クラスタノードあたり最大 1 、 000 個の FlexVol がサポートされます。クラスタの最大 FlexVol 数は 12 、 000 です。Docker ボリューム要件がこの制限に適合する場合、「 ONTAP - NAS 」ドライバは FlexVol が提供する Docker ボリューム単位のスナップショットやクローン作成などの追加機能により、 NAS 解決策の方が望ましいとされます。

FlexVol の制限で対応できる容量よりも多くの Docker ボリュームが必要な場合は、「 ONTAP - NAS - エコノミー」または「 ONTAP - SAN - エコノミー」ドライバを選択します。

「 ONTAP - NAS - エコノミー」ドライバは、自動的に管理される FlexVol プール内の ONTAP qtree として Docker ボリュームを作成します。qtree の拡張性は、クラスタノードあたり最大 10 、 000 、クラスタあたり最大 2 、 40 、 000 で、一部の機能を犠牲にすることで大幅に向上しています。「 ONTAP - NAS - エコノミー」ドライバは、 Docker ボリューム単位のスナップショットまたはクローン作成をサポートしていません。

> Swarm は複数のノード間でのボリューム作成のオーケストレーションを行わないため 'ONTAP-NAS-エコノミー のドライバは現在 Docker Swarm ではサポートされていません

「 ONTAP と SAN の経済性」のドライバは、自動的に管理される FlexVol の共有プール内で、 ONTAP LUN として Docker ボリュームを作成します。この方法により、各 FlexVol が 1 つの LUN に制限されることはなく、 SAN ワークロードのスケーラビリティが向上します。ストレージアレイに応じて、 ONTAP はクラスタあたり最大 16384 個の LUN をサポートします。このドライバは、ボリュームが下位の LUN であるため、 Docker ボリューム単位の Snapshot とクローニングをサポートします。

「 ONTAP-NAS-flexgroup 」ドライバを選択して、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 つのボリュームに並列処理を増やすことができます。FlexGroup のユースケースとしては、 AI / ML / DL 、ビッグデータと分析、ソフトウェアのビルド、ストリーミング、ファイルリポジトリなどが考えられます。Trident 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 』にある、ボリュームへの移行に関するセクション"。

同じ環境で高度な機能と大規模な拡張性を実現するために 'ONTAP-NAS' を使用して Docker Volume Plugin の複数のインスタンスを実行し ' もう 1 つは「 ONTAP-NAS-エコノミー 」を使用して実行できます

ONTAP 構成ファイルの例

*NFS は 'ONTAP-NAS' ドライバの例です

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "defaults": {
      "size": "10G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

*NFS は 'ONTAP-NAS-flexgroup ドライバの例です

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "defaults": {
      "size": "100G",
      "spaceReserve": "none",
      "exportPolicy": "default"
    }
}
```

NFS は 'ONTAP-NAS-エコノミー 'driver の例です

```
{
    "version": 1,
    "storageDriverName": "ontap-nas-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.2",
    "svm": "svm_nfs",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1"
}
```

*iSCSI は 'ONTAP-SAN' ドライバの例です

```
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "igroupName": "myigroup"
}
```

*NFS は 'ONTAP-SAN-エコノミー ドライバーの例です

```
{
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "dataLIF": "10.0.0.3",
    "svm": "svm_iscsi_eco",
    "username": "vsadmin",
    "password": "secret",
    "aggregate": "aggr1",
    "igroupName": "myigroup"
}
```

Element ソフトウェアの設定

Element ソフトウェア（ NetApp HCI / SolidFire ）を使用する場合は、グローバルな設定値のほかに、以下のオプションも使用できます。

オプション	説明	例
「エンドポイント」	https://<login>:<password>@<mvip>/json-rpc/<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
「 VIP 」	iSCSI の IP アドレスとポート	10.0.0.7 ： 3260
「 tenantname 」	使用する SolidFire テナント（見つからない場合に作成）	Docker です
「 InitiatorIFCace 」	iSCSI トラフィックをデフォルト以外のインターフェイスに制限する場合は、インターフェイスを指定します	デフォルト
「タイプ」	QoS の仕様	以下の例を参照してください
「 LegacyNamePrefix 」のように入力します	アップグレードされた Trident インストールのプレフィックス。1.3.2 より前のバージョンの Trident を使用していて、既存のボリューム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 - 」

「 olidfire -san 」ドライバは Docker Swarm をサポートしていません。

Element ソフトウェア構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
    "SVIP": "10.0.0.7:3260",
    "TenantName": "docker",
    "InitiatorIFace": "default",
    "Types": [
        {
            "Type": "Bronze",
            "Qos": {
                "minIOPS": 1000,
                "maxIOPS": 2000,
                "burstIOPS": 4000
            }
        },
        {
            "Type": "Silver",
            "Qos": {
                "minIOPS": 4000,
                "maxIOPS": 6000,
                "burstIOPS": 8000
            }
        },
        {
            "Type": "Gold",
            "Qos": {
                "minIOPS": 6000,
                "maxIOPS": 8000,
                "burstIOPS": 10000
            }
        }
    ]
}
```

GCP 上の Cloud Volumes Service （ CVS ）構成

Trident に、デフォルトの CVS サービスタイプが on に設定された小規模なボリュームがサポートされるようになりました "GCP"。「 torageClass=software 」で作成されたバックエンドの場合、ボリュームのプロビジョニングサイズは最小 300GiB になります。* 非本番環境のワークロード用に 1TiB 未満のボリュームを使用することを推奨 * 。現在、 CVS ではこの機能が限定的な可用性で提供されており、テクニカルサポートは提供されていません。

> 1TiB 未満のボリュームにアクセスするには、サインアップします "こちらをご覧ください"。

デフォルトの CVS サービスタイプ「 torageClass=software 」を使用してバックエンドを導入する場合は、該当するプロジェクト番号とプロジェクト ID の GCP の sub1TiB ボリューム機能へのアクセス権を取得する必要があります。これは Trident で sub-1TiB ボリュームのプロビジョニングに必要です。そうでない場合、ボリュームの作成に失敗します。 PVC が 600 GiB 未満の場合。を使用して 1TiB 未満のボリュームへのアクセスを取得します "このフォーム"。

デフォルトの CVS サービスレベル用に Trident で作成されたボリュームは次のようにプロビジョニングされます。

- 300GiB 未満の PVC があると、 Trident によって 300GiB の CVS ボリュームが作成されます。
- 300GiB から 600GiB の PVC があると、 Trident が要求されたサイズの CVS ボリュームを作成します。
- 600GiB から 1TiB までの PVC の場合、 Trident によって 1TiB の CVS ボリュームが作成されます。
- 1TiB を超える PVC の場合、 Trident は要求サイズの CVS ボリュームを作成します。

GCP で CVS を使用する場合は、グローバル構成の値に加えて、次のオプションも使用できます。

オプション	説明	例
「 apiRegion 」と入力します	CVS アカウントリージョン（必須）。は、このバックエンドがボリュームをプロビジョニングする GCP リージョンです。	「 us-west2 」
「 ProjectNumber 」	GCP プロジェクト番号（必須）。GCP Web ポータルのホーム画面にあります。	“ 123456789012 ”
「 hostProjectNumber 」	GCP 共有 VPC ホストプロジェクト番号（共有 VPC を使用する場合は必須）	「 098765432109 」
「 apiKey 」と入力します	CVS admin ロールを持つ GCP サービスアカウントの API キー（必須）。は、 GCP サービスアカウントの秘密鍵ファイルの JSON 形式のコンテンツです（バックエンド構成ファイルにそのままコピーされます）。サービスアカウントには netappcloudvolumes .admin ロールが必要です。	（秘密鍵ファイルの内容）
「 ecretKey 」	CVS アカウントのシークレットキー（必須）。CVS Web ポータルの「アカウント設定」 > 「 API アクセス」で確認できます。	デフォルト

オプション	説明	例
「 ProxyURL 」と入力します	CVS アカウントへの接続にプロキシサーバが必要な場合は、プロキシ URL を指定します。プロキシサーバには、 HTTP プロキシまたは HTTPS プロキシを使用できます。HTTPS プロキシの場合、証明書の検証は省略され、プロキシサーバで自己署名証明書が使用されるようになります。* 認証が有効になっているプロキシサーバーはサポートされていません * 。	「 http://proxy-server-hostname/”
「 nfsvMountOptions 」のように入力します	NFS マウントオプション。デフォルトは「 -o nfsvers=3 」です。	「 nfsvers=3 、 proto=tcp 、 timeo=600 」
「サービスレベル」	パフォーマンスレベル（標準、プレミアム、エクストリーム）、デフォルトは「標準」	Premium サービス
「ネットワーク」	CVS ボリュームに使用される GCP ネットワーク。デフォルトは「 default 」です。	デフォルト

共有 VPC ネットワークを使用する場合は、「 ProjectNumber 」と「 hostProjectNumber 」の両方を指定する必要があります。その場合、「 ProjectNumber 」はサービスプロジェクト、「 hostProjectNumber 」はホストプロジェクトです。

NetApp Cloud Volumes Service for GCP では、サイズが 100GiB 未満の CVS パフォーマンスボリュームや 300GiB 未満の CVS ボリュームはサポートされていません。アプリケーションの導入を容易にするために、ボリュームサイズが小さすぎる場合は、 Trident によって最小サイズのボリュームが自動的に作成されます。

GCP で CVS を使用している場合は、これらのデフォルトのボリュームオプション設定を使用できます。

オプション	説明	例
「 exportRule 」	NFS アクセスリスト（アドレスおよび CIDR サブネット）。デフォルトは「 0.0.0.0/0 」です。	「 10.0.1.0/24,10.0.2.100 」
「スナップショット方向」	「 .snapshot 」ディレクトリの表示を制御します	いいえ
「スナップショット予約」	スナップショット予約の割合。デフォルトでは、 CVS のデフォルト値である 0 を使用します	10.

オプション	説明	例
「 size 」	ボリュームサイズ、デフォルトは「 100GiB 」	「 10T 」

GCP 上の CVS 構成ファイルの例

```
{
    "version": 1,
    "storageDriverName": "gcp-cvs",
    "projectNumber": "012345678901",
    "apiRegion": "us-west2",
    "apiKey": {
        "type": "service_account",
        "project_id": "my-gcp-project",
        "private_key_id": "1234567890123456789012345678901234567890",
        "private_key": "-----BEGIN PRIVATE KEY-----
\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZ
srrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisI
sAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSa
PIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZN
chRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlz
ZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl
/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kw
s8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY
9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHc
zZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHi
sIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOgu
SaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyA
ZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGz
lzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3
bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4
Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5o
jY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nzn
HczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrt
HisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbO
guSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKe
yAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRA
GzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\nznHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4j
K3bl/qp8B4Kws8zX5ojY9m\nXsYg6gyxy4zq7OlwWgLwGa==\n-----END PRIVATE
KEY-----\n",
        "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/"
}
```

Azure NetApp Files 構成

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。

- Azure NetApp Files が有効な Azure サブスクリプションのスクリプト ID
- 「 tenantID 」、「 clientID 」、「 clientSecret 」を「」から選択します "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
- Azure ロケーションに少なくとも 1 つ以上が含まれている "委任されたサブネット"

初めて Azure NetApp Files を使用している場合や、新しい場所で を使用している場合は、そのような初期設定が必要になります "クイックスタートガイド" ご案内します。

Astra Trident 21.04.0 以前では、手動 QoS 容量プールはサポートされていません。

オプション	説明	デフォルト
「バージョン」	常に 1	
'torageDriverName'	「 azure-NetApp-files 」	
backendName`	ストレージバックエンドのカスタム名	ドライバ名 + "_" + ランダムな文字
' スクリプト ID' 。	Azure サブスクリプションのサブスクリプション ID	
「 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 」を入力します。	アプリケーション登録からのクライアントシークレット	

オプション	説明	デフォルト
「サービスレベル」	「 Standard 」、「 Premium 」、「 Ultra 」のいずれか	「」（ランダム）
「ロケーション」	新しいボリュームを作成する Azure の場所の名前をに指定します	「」（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	「」（ランダム）
「サブネット」	「 icrosoft.Netapp/volumes` 」に委任されたサブネットの名前	「」（ランダム）
「 nfsvMountOptions 」のように入力します	NFS マウントオプションのきめ細かな制御	「 -o nfsvers=3 」
「 limitVolumeSize 」と入力します	要求されたボリュームサイズがこの値を超えている場合、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）

Azure NetApp Files サービスでは、サイズが 100GB 未満のボリュームはサポートされません。Trident では、アプリケーションの導入を容易にするために、より小さいボリュームが要求された場合に、 100GB のボリュームが自動的に作成されます。

これらのオプションを使用して、構成の特別なセクションで各ボリュームをデフォルトでプロビジョニングする方法を制御できます。

オプション	説明	デフォルト
「 exportRule 」	新しいボリュームのエクスポートルール。CIDR 表記の IPv4 アドレスまたは IPv4 サブネットの任意の組み合わせをカンマで区切って指定する必要があります。	「 0.0.0.0/0 」
「スナップショット方向」	「 .snapshot 」ディレクトリ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 」

Azure NetApp Files 構成の例

- 例 1 ： azure-NetApp-files* のバックエンドの最小構成

これは、バックエンドの絶対的な最小構成です。この構成では、 Trident がお客様のネットアップアカウン

ト、容量プール、および ANF に委譲されたサブネットをすべて検出し、新しいボリュームをいずれかの場所にランダムに配置します。

この構成は、 ANF の利用を開始して問題を解決するのに役立ちます。 しかし実際には、プロビジョニングするボリュームの範囲を追加して、必要な特性を確実に持ち、それを使用しているコンピューティングに近いネットワーク上で終了するようにします。詳細については、以降の例を参照してください。

```
{
    "version": 1,
    "storageDriverName": "azure-netapp-files",
    "subscriptionID": "9f87c765-4774-fake-ae98-a721add45451",
    "tenantID": "68e4f836-edc1-fake-bff9-b2d865ee56cf",
    "clientID": "dd043f63-bf8e-fake-8076-8de91e5713aa",
    "clientSecret": "SECRET"
}
```

- 例 2 ： Azure NetApp Files の単一の場所と特定のサービスレベル *

このバックエンド構成では、 Azure の「 eastus 」ロケーションにボリュームを「 Premium 」容量プールに配置します。Trident は、 ANF に委任されているすべてのサブネットを自動的に検出し、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": "Premium"
}
```

- 例 3 ： azure-NetApp-files* の高度な設定

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": "Premium",
    "virtualNetwork": "my-virtual-network",
    "subnet": "my-subnet",
    "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
    "limitVolumeSize": "500Gi",
    "defaults": {
        "exportRule": "10.0.0.0/24,10.0.1.0/24,10.0.2.100",
        "size": "200Gi"
    }
}
```

- 例 4 ： azure-NetApp-files* を使用する仮想ストレージプール

このバックエンド構成では、複数のが定義され "ストレージのプール" 1 つのファイルに格納できます。これは、異なるサービスレベルをサポートする複数の容量プールがあり、それらを表すストレージクラスを Kubernetes で作成する場合に便利です。

仮想ストレージプールの機能の表面に、ラベルが貼られています。

```
{
    "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",
    "nfsMountOptions": "nfsvers=3,proto=tcp,timeo=600",
    "labels": {
        "cloud": "azure"
    },
    "location": "eastus",

    "storage": [
        {
            "labels": {
                "performance": "gold"
            },
            "serviceLevel": "Ultra"
        },
        {
            "labels": {
                "performance": "silver"
            },
            "serviceLevel": "Premium"
        },
        {
            "labels": {
                "performance": "bronze"
            },
            "serviceLevel": "Standard",
        }
    ]
}
```

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. 追加の 'config' パラメータを指定して ' プラグインを再インストールします

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

ボリューム名は 2 文字以上にする必要があります。

これは Docker クライアントの制限事項です。クライアントは、 1 文字の名前を Windows パスと解釈します。 "バグ 25773 を参照"。

Docker Swarm には、 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。ONTAP の仕組みと 'ONTAP-NAS' および 'ONTAP-SAN' ドライバの機能の仕組みにより ' これらの制限の範囲内で動作できるのはこれらのドライバだけです

その他のドライバには、競合状態などの問題があります。このような問題が発生すると、ボリュームを同じ名前で異なる ID にする機能が Element に備わっているため、「勝者」を明確にせずに 1 回の要求で大量のボリュームを作成できるようになります。

ネットアップは Docker チームにフィードバックを提供しましたが、今後の変更の兆候はありません。

FlexGroup をプロビジョニングする場合、プロビジョニングする FlexGroup と共通のアグリゲートが 2 つ目の FlexGroup に 1 つ以上あると、 ONTAP は 2 つ目の FlexGroup をプロビジョニングしません。

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。

tridentctl get コマンドを使用して 'Astra Trident の構成に関する詳細情報を取得します

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名は csi.trident.netapp.io` です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 はもはや「 etcd 」を必要としない。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、クラスタへのアクセスが「 kubectl 」に設定されている任意のマシンからのリモートインストール機能がサポートされています。kubectl アクセスが確認されたら (たとえば ' リモート・マシンから kubectl get nodes コマンドを起動して確認します) ' 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、「 setup/backend.json 」ファイルに一意の「 StoragePrefix 」パラメータを指定します。これは、両方のインスタンスで同じ FlexVol を使用しないためです。

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？

tridentctl logs-a を実行して ' サポートバンドルを作成できますバンドルでキャプチャされたログに加えて、kubelet ログをキャプチャして、 Kubernetes 側のマウントの問題を診断します。kubelet ログの取得手順は、Kubernetes のインストール方法によって異なります。

新しい機能のリクエストを発行する必要がある場合は、どうすればよいですか。

に問題を作成します "Trident Github の利用" 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、問題、ご要望がございましたら、弊社までお問い合わせください "Slack" チームまたは 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 tridentctl update backend myBackend-f </path/to_new_backend.json >-n trident` で更新します。この例の「 m yBackend` 」をバックエンド名に置き換え、「 /path/to _new_backend.json 」

を正しい「 backend.json 」ファイルへのパスに置き換えます。

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。

バックエンド定義ファイルには両方を指定することを推奨します。必須の管理 LIF は 1 つだけです。

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。バックエンド構成では 'useCHAP =true' を設定する必要があります

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。「 backend.json 」ファイルに「 managementlif 」として設定します。 <IP アドレス >:<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、「 managementlif 」：「 192.0.2.1 ： 1000」 を設定します。

管理 LIF とデータ LIF に IPv6 アドレスを使用できますか。

はい。Astra Trident 20.01 は、 ONTAP バックエンドの管理 LIF パラメータとデータ LIF パラメータに対して IPv6 アドレスを定義できます。アドレスが IPv6 のセマンティクスに準拠していること、および管理 LIF が角かっこで囲まれて定義されていることを確認する必要があります（例：[ec0d:6504:a9c1:ae67:53d1:4bdf:ab32:e233]`` ）。また、 IPv6 で動作するためには、 Astra Trident が「 --use-ipv6' 」フラグを使用してインストールされていることを確認する必要があります。

バックエンドの管理 LIF を更新できますか。

はい。 tridentctl update backend コマンドを使用してバックエンド管理 LIF を更新できます。

バックエンドのデータ LIF を更新できるか。

いいえ、バックエンドのデータ LIF を更新できません。

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切なプールを自動的に選択できない場合は 'toragePools' パラメータと additionalStoragePools' パラメータを使用して ' 特定のプールセットを選択します

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 'Astra Trident がプロビジョニングに使用する一連のプールをフィルタリングし ' 一致するプールをすべて削除します

同じ種類のバックエンドが複数ある場合、 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、「 torageClass 」および「 PersistentVolumeClaim 」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。たとえば、複数の ONTAP-NAS ドライババックエンドがある場合、 Astra Trident は「 S torageClass 」と「 PersistentVolumeClaim 」のパラメータを組み合わせて照合し、「 S torageClass 」と「 PersistentVolumeClaim 」に記載された要件を提供できるバックエンドと照合します。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-エコノミー 」ドライバは、同一の FlexVol （ 50 〜 300 の範囲で設定可能）で最大 200 個の qtree を作成し、クラスタ・ノードあたり 100,000 個の qtree を作成し、クラスタあたり 240 万個を作成します。エコノミー・ドライバーがサービスを提供する新しい「 PersistentVolumeClaim 」を入力すると、ドライバーは新しい qtree にサービスを提供できる FlexVol がすでに存在するかどうかを確認します。qtree を提供できる FlexVol が存在しない場合は、新しい FlexVol が作成されます。

ONTAP NAS でプロビジョニングされたボリュームに UNIX アクセス権を設定するにはどうすればよいですか。

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、バックエンド定義ファイルに設定されている「 exportPolicy 」パラメータを使用します。

ONTAP を使用して 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。

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。

ONTAP の QoS を実装するには、「 torageClasses 」を使用します。

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「 torageClass 」を設定する必要があります。両方が設定されている場合

は、「 torageClass 」が優先されます。ONTAP で次の項目を設定します。

1. 'S torageClass' で 'provisioningType' 属性を thick に設定します
2. バックエンド定義ファイルで 'backend spaceReserve パラメータを volume に設定して ' シックボリュームを有効にします

誤って **PVC** を削除した場合でも、使用中のボリュームが削除されないようにするにはどうすればよいですか。

Kubernetes では、バージョン 1.10 以降、 PVC 保護が自動的に有効になります。

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 'VolumeSnapshotDataSource フィーチャーゲートが有効になっていることを確認します

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のサポートは 'ONTAP-NAS'`ONTAP-NAS-flexgroup 'ONTAP-SAN’ONTAP-SANエコノミー "solidfire-san-SAN'"solidfire-san-'""solidfire-san-'"solidfire-san-'""" で利用できます「 gcp-cvs` 」と「 azure-NetApp-files 」バックエンドドライバ。

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 'ONTAP-NAS' 'ONTAP-SAN' および 'ONTAP-NAS-flexgroup ドライバで利用できますFlexVol レベルでは「 ONTAP-SAN-エコノミー 」ドライバに「スナップショットポリシー」を指定することもできます。

これは「 ONTAP-NAS-エコノミー 」ドライバでも利用できますが、 FlexVol レベルの細分性ではなく、 qtree レベルの細分性で利用できます。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、バックエンドパラメータオプション「 napshotPolicy 」を、 ONTAP バックエンドで定義されている目的のスナップショットポリシーに設定します。ストレージコントローラで作成された 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。バックエンド定義ファイルで「スナップショット予約」属性を設定することで、 Astra Trident を介してスナップショットコピーを保存するためのディスク領域の特定の割合を予約できます。バックエンド定義ファイルで「 napshotPolicy 」と「 napshotReserve 」を設定した場合、バックエンドファイルに記載されている「 napshotReserve 」の割合に従ってスナップショット予約の割合が設定されます。「スナップショット予約」の割合の数値が指定されていない場合、 ONTAP はデフォルトでスナップショット予約の割合を 5 に設定します。「スナップショット予約」オプションが「なし」に設定されている場合、スナップショット予約の割合は 0 に設定されます。

ボリュームの Snapshot ディレクトリに直接アクセスしてファイルをコピーできますか。

はい。バックエンド定義ファイルで「 snapmirror directionDir 」パラメータを設定することで、 Trident によってプロビジョニングされたボリューム上のスナップショットディレクトリにアクセスできます。

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。

コンテナユーザー（ Astra Trident 開発者を含む）の活発なパブリックコミュニティが、「 containers 」チャネルにも存在します "ネットアップの Slack ワーク"。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 のヘルプを参照するには 'tridentctl logs-a -n trident` を使用してサポートバンドルを作成し 'NetApp Support < 困ったときは > に送信してください

> 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場合（たとえば、 Trident ポッドが 2 つ未満の「 ContainerCreating 」フェーズで停止した場合）、「 kubectl-n trident 」を実行して、展開が trident 、「 kubectl-n trident 」が pod trident- を記述します **-**** 追加のインサイトを提供できます。kubelet ログの取得 (journalctl -xeu kubelet など) も役立ちます。
- Trident ログに十分な情報がない場合は、インストールオプションに基づいてインストールパラメータに「 -d 」フラグを渡して、 Trident のデバッグモードを有効にしてみてください。

  次に './tridentctl logs -n trident` を使用して debug が設定され ' ログ内で 'level=debug msg' を検索していることを確認します

  オペレータとともにインストールされます

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

  すべての Trident ポッドが再起動されます。これには数秒かかることがあります。これを確認するには 'kubectl get pod -n trident' の出力の 'age' 列を確認します

  Astra Trident 20.07 と 20.10 では、「 Torc 」の代わりに「 tprov 」を使用します。

  Helm とともにインストールされます

  ```
  $ helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
  tridentDebug=true`
  ```

  tridentctl を使用してインストールされます

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

- バックエンド定義に「 debugTraceFlags 」を含めると、バックエンドごとにデバッグログを取得することもできます。たとえば、 Trident ログで API 呼び出しとメソッドの逆数を取得するには、「

`debugTraceFlags: {"API":true,"method" :true,}` を指定します。既存のバックエンドには 'tridentctl backend update で構成された 'ebugTraceFlags' を設定できます

- RedHat CoreOS を使用する場合は 'iscsid がワーカー・ノードで有効になっており ' デフォルトで起動されていることを確認しますこの設定には、 OpenShift MachineConfig を使用するか、イグニッションテンプレートを変更します。
- Trident をで使用する際によく発生する問題です "Azure NetApp Files の特長" テナントとクライアントのシークレットが、必要な権限がないアプリケーションの登録から取得された場合です。Trident の要件の詳細については、を参照してください "Azure NetApp Files の特長" 設定
- コンテナへの PV のマウントに問題がある場合は 'rpcbind' がインストールされていて実行されていることを確認してくださいホスト OS に必要なパッケージ・マネージャを使用して 'rpcbind' が実行されているかどうかを確認しますrpcbind サービスのステータスは 'systemctl status rpcbind' またはそれに相当する処理を実行することで確認できます
- Trident バックエンドが、以前に作業したことがあるにもかかわらず「 failed 」状態であると報告した場合は、バックエンドに関連付けられている SVM/admin クレデンシャルの変更が原因である可能性があります。「 tridentctl update backend 」または Trident ポッドのバウンスを使用してバックエンド情報を更新すると、この問題は修正されます。
- 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。次に 'tridentctl コマンドを使用して ' アルファスナップショットの CRD を削除できます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。
- Docker をコンテナランタイムとして Trident をインストールするときに権限の問題が発生した場合は、「 --in cluster=false」 フラグを付けて Trident のインストールを試みてください。これはインストーラポッドを使用せず、「 trident-installer 」ユーザのために発生する許可の問題を回避します。
- 実行に失敗した後のクリーンアップには 'uninstall パラメータ <Uninstalling Trident > を使用しますデフォルトでは、スクリプトは Trident によって作成された CRD を削除しないため、実行中の導入環境でも安全にアンインストールしてインストールできます。
- Trident の以前のバージョンにダウングレードする場合は、まず tridenctl uninstall コマンドを実行して Trident を削除します。必要なをダウンロードします "Trident のバージョン" tridentctl install コマンドを使用してインストールします新しい 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、ステートの維持に CRD 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 、およびボリュームスナップショット）には、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 に書き込まれたデータではなく、「関連 CRD オブジェクト」が含まれています。* 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、「 tridentctl identviate 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「保留中」段階で停止した場合、「 kubectl 」を実行して PVC を記述すると、 Trident がこの PVC の PV のプロビジョニングに失敗した理由を追加情報に提供できます。

オペレータを使用して失敗した **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 を導入する場合 'TridentOrchestrator のステータスは 'Installing から Installed に変わります'Failed' ステータスが表示され ' オペレータがそれ自体で回復できない場合は ' 次のコマンドを実行してオペレータのログを確認する必要があります

```
tridentctl logs -l trident-operator
```

trident-operator コンテナのログの末尾には、問題のある場所を示すことができます。たとえば、このような問題の 1 つは、エアーギャップ環境のアップストリームレジストリから必要なコンテナイメージをプルできないことです。

Trident のインストールが失敗した理由を理解するには、「 TridentOrchestrator 」のステータスを確認する必要があります。

```
$ kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Enable Node Prep:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                        Message
  ----     ------  ----               ----                        -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

このエラーは、 Trident のインストールに使用された「 TridentOrchestrator 」がすでに存在することを示します。各 Kubernetes クラスタは Trident のインスタンスを 1 つしか保持できないため、オペレータは任意の時点で作成可能なアクティブな TridentOrchestrator が 1 つだけ存在することを確認します。

また、 Trident ポッドのステータスを確認することで、適切でないものがあるかどうかを確認できます。

```
$ kubectl get pods -n trident

NAME                                  READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                     1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw          4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                     1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                     1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv     1/1     Running            0
8m17s
```

1 つ以上のコンテナイメージがフェッチされなかったため、ポッドが完全に初期化できないことがわかります。

この問題に対処するには、「 TridentOrchestrator 」 CR を編集する必要があります。また、「 TridentOrchestrator 」を削除して、変更された正確な定義を持つ新しいものを作成することもできます。

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特定するために、インストーラをもう一度「 -d`` 」引数を使用して実行すると、デバッグモードが有効になり、問題の内容を理解するのに役立ちます。

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

問題を解決した後 ' 次のようにインストールをクリーンアップし 'tridentctl install コマンドを再度実行できます

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```

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

ネットアップのポートフォリオに含まれる各ストレージプラットフォームは、コンテナ化されたアプリケーションや含まれないアプリケーションにメリットをもたらす独自の機能を備えています。Trident は ONTAP や Element と連携1 つのプラットフォームが他のプラットフォームよりもすべてのアプリケーションとシナリオに適しているわけではありませんが、プラットフォームを選択する際には、アプリケーションのニーズとデバイスを管理するチームを考慮する必要があります。

使用するプロトコルに対応したホストオペレーティングシステムのベースラインベストプラクティスに従う必要があります。必要に応じて、アプリケーションのベストプラクティスを適用する際に、バックエンド、ストレージクラス、 PVC の設定を利用して、特定のアプリケーションのストレージを最適化することもできます。

ONTAP と Cloud Volumes ONTAP のベストプラクティス

Trident 向けに ONTAP と Cloud Volumes ONTAP を設定するためのベストプラクティスをご確認ください。

次に示す推奨事項は、 Trident によって動的にプロビジョニングされたボリュームを消費するコンテナ化されたワークロード用に ONTAP を設定する際のガイドラインです。それぞれの要件を考慮し、環境内で適切かどうかを評価する必要があります。

Trident 専用の SVM を使用

Storage Virtual Machine （ SVM ）を使用すると、 ONTAP システムのテナントを分離し、管理者が分離できます。SVM をアプリケーション専用にしておくと、権限の委譲が可能になり、リソース消費を制限するためのベストプラクティスを適用できます。

SVM の管理には、いくつかのオプションを使用できます。

- バックエンド構成でクラスタ管理インターフェイスを適切なクレデンシャルとともに指定し、 SVM 名を指定します。
- ONTAP System Manager または CLI を使用して、 SVM 専用の管理インターフェイスを作成します。
- NFS データインターフェイスで管理ロールを共有します。

いずれの場合も、インターフェイスは DNS にあり、 Trident の設定時には DNS 名を使用する必要があります。これにより、ネットワーク ID を保持しなくても SVM-DR などの一部の DR シナリオが簡単になります。

専用の管理 LIF または共有の管理 LIF を SVM に使用する方法は推奨されませんが、ネットワークセキュリティポリシーを選択した方法と一致させる必要があります。最大の柔軟性を確保するには、どのような場合でも DNS 経由で管理 LIF にアクセスできるようにします "SVM-DR" Trident と組み合わせて使用できます。

最大ボリューム数を制限します

ONTAP ストレージシステムの最大ボリューム数は、ソフトウェアのバージョンとハードウェアプラットフォームによって異なります。を参照してください "NetApp Hardware Universe の略" 具体的な制限については、使用しているプラットフォームと ONTAP のバージョンに対応しています。ボリューム数を使い果たした場合、 Trident のプロビジョニング処理だけでなく、すべてのストレージ要求に対してプロビジョニング処理が失敗します。

Trident の「 ONTAP - NAS 」と「 ONTAP - SAN 」ドライバは、作成された Kubernetes 永続ボリューム（ PV ）ごとに FlexVol をプロビジョニングします。「 ONTAP-NAS-エコノミー 」ドライバは、 PVS 200 個につき約 1 つの FlexVol を作成します（ 50 〜 300 の範囲で構成可能）。「 ONTAP-SAN-エコノミー 」ドライバは、 PVS 100 個につき約 1 つの FlexVol を作成します（ 50 〜 200 の範囲で設定可能）。Trident がストレージシステム上の使用可能なボリュームをすべて消費しないようにするには、 SVM に制限を設定する必要があります。コマンドラインから実行できます。

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

「 mAX-VOLUMES 」の値は、環境に固有のいくつかの条件によって異なります。

- ONTAP クラスタ内の既存のボリュームの数
- 他のアプリケーション用に Trident 外部でプロビジョニングするボリュームの数
- Kubernetes アプリケーションで消費されると予想される永続ボリュームの数

「 mAX-volumes 」の値は、 ONTAP クラスタ内のすべてのノードでプロビジョニングされたボリュームの合計数であり、個々の ONTAP ノードではプロビジョニングされません。その結果、 ONTAP クラスタノードの Trident でプロビジョニングされたボリュームの数が、別のノードよりもはるかに多い、または少ない場合があります。

たとえば、 2 ノードの ONTAP クラスタでは、最大 2 、 000 個の FlexVol をホストできます。最大ボリュー

ム数を 1250 に設定していると、非常に妥当な結果が得られます。ただし、のみの場合 "アグリゲート" あるノードから SVM に割り当てられている場合や、あるノードから割り当てられたアグリゲートをプロビジョニングできない場合（容量など）は、他のノードが Trident でプロビジョニングされたすべてのボリュームのターゲットになります。これは、「 mAX-VOLUMES 」の値に達する前にそのノードのボリューム数の上限に達する可能性があることを意味し、 Trident とそのノードを使用する他のボリューム処理の両方に影響します。* クラスタ内の各ノードのアグリゲートを、 Trident が使用する SVM に同じ番号で確実に割り当てることで、この状況を回避できます。 *

Trident で作成できるボリュームの最大サイズを制限

Trident で作成できるボリュームの最大サイズを設定するには、「 backend.json 」の定義で「 limitVolumeSize 」パラメータを使用します。

ストレージアレイでボリュームサイズを制御するだけでなく、 Kubernetes の機能も利用する必要があります。

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。バックエンド構成で「 useCHAP 」パラメータを使用すると、 Trident は CHAP を使用して ONTAP バックエンドの iSCSI 接続を認証します。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。

- バックエンド構成の「金庫」ブロックに「 QOSPolicy 」キーを設定する必要があります。次のバックエンド設定例を参照してください。

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "0.0.0.0",
  "dataLIF": "0.0.0.0",
  "svm": "svm0",
  "username": "user",
  "password": "pass",
  "defaults": {
    "qosPolicy": "standard-pg"
  },
  "storage": [
    {
      "labels": {"performance": "extreme"},
      "defaults": {
        "adaptiveQosPolicy": "extremely-adaptive-pg"
      }
    },
    {
      "labels": {"performance": "premium"},
      "defaults": {
        "qosPolicy": "premium-pg"
      }
    }
  ]
}
```

- ボリュームごとにポリシーグループを適用して、各ボリュームがポリシーグループの指定に従ってスループット全体を取得するようにします。共有ポリシーグループはサポートされません。

QoS ポリシーグループの詳細については、を参照してください "ONTAP 9.8 QoS コマンド"。

ストレージリソースへのアクセスを **Kubernetes** クラスタメンバーに制限する

Trident によって作成される NFS ボリュームと iSCSI LUN へのアクセスを制限することは、 Kubernetes 環境のセキュリティ体制に欠かせない要素です。これにより、 Kubernetes クラスタに属していないホストがボリュームにアクセスしたり、データが予期せず変更されたりすることを防止できます。

ネームスペースは Kubernetes のリソースの論理的な境界であることを理解することが重要です。ただし、同じネームスペース内のリソースは共有可能であることが前提です。重要なのは、ネームスペース間に機能がないことです。つまり、 PVS はグローバルオブジェクトですが、 PVC にバインドされている場合は、同じネームスペース内のポッドからのみアクセス可能です。* 適切な場合は、名前空間を使用して分離することが重要です。 *

Kubernetes 環境でデータセキュリティを使用する場合、ほとんどの組織で最も懸念されるのは、コンテナ内のプロセスがホストにマウントされたストレージにアクセスできることですが、コンテナ用ではないためです。 "ネームスペース" この種の妥協を防ぐように設計されています。ただし、特権コンテナという例外が 1 つあります。

権限付きコンテナは、通常よりもホストレベルの権限で実行されるコンテナです。デフォルトでは拒否されないため、を使用してこの機能を無効にしてください "ポッドセキュリティポリシー"。

Kubernetes と外部ホストの両方からアクセスが必要なボリュームでは、 Trident ではなく管理者が導入した PV で、ストレージを従来の方法で管理する必要があります。これにより、 Kubernetes と外部ホストの両方が切断され、ボリュームを使用していない場合にのみ、ストレージボリュームが破棄されます。また、カスタムエクスポートポリシーを適用して、 Kubernetes クラスタノードおよび Kubernetes クラスタの外部にあるターゲットサーバからのアクセスを可能にすることもできます。

専用のインフラノード（ OpenShift など）や、ユーザアプリケーションにスケジュールできない他のノードを導入する場合は、別々のエクスポートポリシーを使用してストレージリソースへのアクセスをさらに制限する必要があります。これには、これらのインフラ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。

- 「 vserver export-policy create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーを作成します。
- 「 vserver export-policy rule create 」 ONTAP CLI コマンドを使用して、エクスポートポリシーにルールを追加します。

これらのコマンドを実行すると、データにアクセスできる Kubernetes ノードを制限できます。

無効にします `showmount` アプリケーション**SVM**用

「 SVM 」機能を使用すると、 NFS クライアントが SVM に照会して使用可能な NFS エクスポートのリストを表示できます。Kubernetes クラスタに導入されたポッドは、データ LIF に対する「 howmount-e 」コマンドを問題に送信し、アクセス権がないマウントも含め、使用可能なマウントのリストを受信できます。これだけではセキュリティ上の妥協ではありませんが、権限のないユーザが NFS エクスポートに接続するのを阻止する可能性のある不要な情報が提供されます。

SVM レベルの ONTAP CLI コマンドを使用して、 SVM の howmount を無効にする必要があります。

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire のベストプラクティス

Trident に SolidFire ストレージを設定するためのベストプラクティスをご確認ください。

SolidFire アカウントを作成します

各 SolidFire アカウントは固有のボリューム所有者で、 Challenge Handshake Authentication Protocol （ CHAP ；チャレンジハンドシェイク認証プロトコル）クレデンシャルのセットを受け取ります。アカウントに割り当てられたボリュームには、アカウント名とその CHAP クレデンシャルを使用してアクセスするか、ボリュームアクセスグループを通じてアクセスできます。アカウントには最大 2 、 000 個のボリュームを関連付けることができますが、 1 つのボリュームが属することのできるアカウントは 1 つだけです。

QoS ポリシーを作成する

標準的なサービス品質設定を作成して保存し、複数のボリュームに適用する場合は、 SolidFire のサービス品質（ QoS ）ポリシーを使用します。

QoS パラメータはボリューム単位で設定できます。QoS を定義する 3 つの設定可能なパラメータである Min IOPS 、 Max IOPS 、 Burst IOPS を設定することで、各ボリュームのパフォーマンスが保証されます。

4KB のブロックサイズの最小 IOPS 、最大 IOPS 、バースト IOPS の値を次に示します。

IOPS パラメータ	定義（ **Definition** ）	最小価値	デフォルト値	最大値（ **4KB** ）
最小 IOPS	ボリュームに対して保証されたレベルのパフォーマンス。	50	50	15000
最大 IOPS	パフォーマンスはこの制限を超えません。	50	15000	200,000
バースト IOPS	短時間のバースト時に許容される最大 IOPS 。	50	15000	200,000

Max IOPS と Burst IOPS は最大 200 、 000 に設定できますが、実際のボリュームの最大パフォーマンスは、クラスタの使用量とノードごとのパフォーマンスによって制限されます。

ブロックサイズと帯域幅は、 IOPS に直接影響します。ブロックサイズが大きくなると、システムはそのブロックサイズを処理するために必要なレベルまで帯域幅を増やします。帯域幅が増えると、システムが処理可能な IOPS は減少します。を参照してください "SolidFire のサービス品質" QoS およびパフォーマンスの詳細については、を参照してください。

SolidFire 認証

Element では、認証方法として CHAP とボリュームアクセスグループ（ VAG ）の 2 つがサポートされています。CHAP は CHAP プロトコルを使用して、バックエンドへのホストの認証を行います。ボリュームアクセスグループは、プロビジョニングするボリュームへのアクセスを制御します。CHAP はシンプルで拡張性に制限がないため、認証に使用することを推奨します。

Trident と強化された CSI プロビジョニングツールは、 CHAP 認証の使用をサポートします。VAG は、従来の CSI 以外の動作モードでのみ使用する必要があります。

CHAP 認証（イニシエータが対象のボリュームユーザであることの確認）は、アカウントベースのアクセス制御でのみサポートされます。認証に CHAP を使用している場合は、単方向 CHAP と双方向 CHAP の 2 つのオプションがあります。単方向 CHAP は、 SolidFire アカウント名とイニシエータシークレットを使用してボリュームアクセスを認証します。双方向の CHAP オプションを使用すると、ボリュームがアカウント名とイニシエータシークレットを使用してホストを認証し、ホストがアカウント名とターゲットシークレットを使用してボリュームを認証するため、ボリュームを最も安全に認証できます。

ただし、 CHAP を有効にできず VAG が必要な場合は、アクセスグループを作成し、ホストのイニシエータとボリュームをアクセスグループに追加します。アクセスグループに追加した各 IQN は、 CHAP 認証の有無に関係なく、グループ内の各ボリュームにアクセスできます。iSCSI イニシエータが CHAP 認証を使用するように設定されている場合は、アカウントベースのアクセス制御が使用されます。iSCSI イニシエータが CHAP 認証を使用するように設定されていない場合は、ボリュームアクセスグループのアクセス制御が使用されます。

詳細情報の入手方法

ベストプラクティスのドキュメントの一部を以下に示します。を検索します "NetApp ライブラリ" 最新バージョンの場合。

- ONTAP *
- "『 NFS Best Practice and Implementation Guide 』を参照してください"
- "『 SAN アドミニストレーションガイド』" （ iSCSI の場合）
- "RHEL 向けの iSCSI のクイック構成"
- Element ソフトウェア *
- "SolidFire for Linux を設定しています"
- NetApp HCI *
- "NetApp HCI 導入の前提条件"
- "NetApp Deployment Engine にアクセスします"
- アプリケーションのベストプラクティス情報 *
- "ONTAP での MySQL に関するベストプラクティスです"
- "SolidFire での MySQL に関するベストプラクティスです"
- "NetApp SolidFire および Cassandra"
- "SolidFire での Oracle のベストプラクティス"
- "SolidFire での PostgreSQL のベストプラクティスです"

すべてのアプリケーションに具体的なガイドラインがあるわけではありません。そのためには、ネットアップのチームと協力し、を使用することが重要です "NetApp ライブラリ" 最新のドキュメントを検索できます。

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、仮想ストレージプールの設計、永続的ボリューム要求（ PVC ）は、 Astra Trident を使用したストレージプロビジョニング、ボリューム運用、 OpenShift サービスの導入に影響を及ぼします。

ドライバの選択と展開

ONTAP のバックエンドドライバを選択します

ONTAP システムでは、 4 種類のバックエンドドライバを使用できます。これらのドライバの違いは、使用するプロトコルと、ストレージシステムでのボリュームのプロビジョニング方法です。そのため、どのドライバを展開するかを慎重に検討してください。

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。

ONTAP バックエンドの 5 つのドライバを次に示します。

- [ONTAP-NAS] ：プロビジョニングされた各 PV は、フル ONTAP FlexVol です。
- 「 ONTAP-NAS-エコノミー 」：プロビジョニングされた各 PV は qtree で、 FlexVol あたりの qtree 数を設定できます（デフォルトは 200 ）。
- 「 ONTAP-NAS-flexgroup 」：フル ONTAP FlexGroup としてプロビジョニングされた各 PV と、 SVM に割り当てられたすべてのアグリゲートが使用されます。
- 「 ONTAP - SAN 」：プロビジョニングされた各 PV は、固有の FlexVol 内の LUN です。
- 「 ONTAP-SAN-エコノミー 」：各 PV がプロビジョニングされた LUN で、 FlexVol あたりの LUN 数を設定できます（デフォルトは 100 ）。

3 つの NAS ドライバの間で選択すると、アプリケーションで使用できる機能にいくつかの影響があります。

次の表では、 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。	○脚注：1[]	はい。	○脚注：1[]
「 ONTAP - NAS - エコノミー」	○脚注：3[]	○脚注：3[]	○脚注：5[]	はい。	○脚注：3[]	はい。	○脚注：3[]
「 ONTAP-NAS-flexgroup 」	○脚注：1[]	いいえ	○脚注：5[]	はい。	○脚注：1[]	はい。	○脚注：1[]

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。	○脚注： 4[]	はい。	○脚注： 1[]	はい。	○脚注： 1[]
「 ONTAP - SAN - エコノミー」	はい。	はい。	○脚注： 4[]	はい。	○脚注： 3[]	○脚注： 1[]	○脚注： 3[]

> 上記の表の脚注： 1 [ ]:Astra Trident 脚注で管理されていません。 2[ ]:Astra Trident で管理されていますが、 PV レベルの細かい脚注ではできません。説明： 4[] ： Raw ブロックボリュームの脚注： 5 [ ]Trident でサポートされています。 CSI でサポートされています

PV に細分化されていない機能は FlexVol 全体に適用され、 PVS （共有 FlexVol 内の qtree または LUN ）にはすべて共通のスケジュールが適用されます。

上の表に示すように 'ONTAP-NAS' と「 ONTAP-NAS-エコノミー 」の機能の多くは同じですしかし 'ONTAP-NAS-エコノミー のドライバは ' スケジュールを PV 単位で制御する機能を制限するため ' これは特に災害復旧やバックアップ計画に影響を与える可能性がありますONTAP ストレージ上で PVC クローン機能を活用したい開発チームの場合 ' これは 'ONTAP-NAS' ONTAP -SAN' または ONTAP -SAN' のいずれかのドライバを使用する場合にのみ可能です

> 「 olidfire -san 」ドライバは PVC のクローン作成にも対応しています。

Cloud Volumes ONTAP のバックエンドドライバを選択します

Cloud Volumes ONTAP は、ファイル共有や NAS および SAN プロトコル（ NFS 、 SMB / CIFS 、 iSCSI ）を提供するブロックレベルストレージなど、さまざまなユースケースでデータ制御とエンタープライズクラスのストレージ機能を提供します。Cloud Volume ONTAP の互換性のあるドライバは、「 ONTAP-NAS' 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-SAN' 」、「 ONTAP-SAN-エコノミー 」です。Cloud Volume ONTAP for Azure と Cloud Volume ONTAP for GCP に該当します。

Amazon FSX for ONTAP のバックエンドドライバを選択します

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、「 ONTAP-NAS' 」、「 ONTAP-NAS-エコノミー 」、「 ONTAP-NAS-flexgroup 」、「 ONTAP-SAN' 」、「 ONTAP-SAN-エコノミー 」です。

NetApp HCI / SolidFire のバックエンドドライバを選択します

NetApp HCI / SolidFire プラットフォームで使用される「 olidfire -SAN」 ドライバは、管理者が QoS 制限に基づいて Trident の Element バックエンドを構成するのに役立ちます。Trident によってプロビジョニングされるボリュームに特定の QoS 制限を設定するためにバックエンドを設計する場合は、バックエンドファイルの「 type 」パラメータを使用します。管理者は 'limitVolumeSize' パラメータを使用して ' ストレージ上に作成できるボリューム・サイズを制限することもできます現在、ボリュームのサイズ変更やボリュームのレプリケーションなどの Element ストレージ機能は、 'olidfire-san' ドライバではサポートされていません。これらの処理は、 Element ソフトウェアの Web UI から手動で実行する必要があります。

SolidFire ドライバ	Snapshot	クローン	マルチアタッチ	CHAP	QoS	サイズ変更	レプリケーション
「 olidfire -san 」	はい。	はい。	○脚注： 2[]	はい。	はい。	はい。	○脚注： 1[]

> 脚注：はい脚注： 1[] ： Astra Trident で管理されていません。 * 注： 2[] ：未フォーマットのブロックボリュームでサポートされています

Azure NetApp Files のバックエンドドライバを選択します

Astra Trident は、「 azure-NetApp-files 」ドライバを使用してを管理します "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 」と入力します	はい。	はい。	はい。	はい。	はい。	○脚注： 1[]

> 脚注：はい脚注： 1[] ： Astra Trident で管理されていません

GCP で **Cloud Volumes Service** のバックエンドドライバを選択します

Astra Trident は、 GCP バックエンドの Cloud Volumes Service とリンクするために、「 gcp-cvs` ドライバ」を使用します。Trident で GCP バックエンドを構成するには、バックエンドファイルで「 ProjectNumber 」、「 apiRegion 」、「 apiKey 」を指定する必要があります。プロジェクト番号は GCP Web ポータルで確認できますが、 GCP で Cloud Volume の API アクセスを設定する際に作成したサービスアカウントの秘密鍵ファイルから API キーを取得する必要があります。Astra Trident なら、 CVS ボリュームを 2 つのうちの 1 つで作成できます "サービスタイプ"：

1. * CVS * ：基本 CVS サービスのタイプ。パフォーマンスレベルが限定的か中程度かに関係なく、高ゾーンの可用性を実現します。
2. * CVS - パフォーマンス * ：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な、パフォーマンスに最適化されたサービスタイプ。独自の 3 つのサービス・レベル（「標準」、「プレミアム」、「エクストリーム」）から選択できます。現在、プロビジョニングする CVS パフォーマンスボリュームの最小サイズは 100GiB で、 CVS ボリュームは 300GiB 以上である必要があります。今後の CVS リリースでは、この制限が解除される可能性があります。

> デフォルトの CVS サービスタイプ [`torageClass=software`] を使用してバックエンドを導入する場合、ユーザ * は該当するプロジェクト番号とプロジェクト ID について GCP の 1TiB サブボリューム機能へのアクセス * を取得する必要があります。これは Trident で sub-1TiB ボリュームのプロビジョニングに必要です。そうでない場合、ボリュームの作成に失敗します。PVC が 600 GiB 未満の場合。使用 "このフォーム" 1TiB 未満のボリュームへのアクセス権を取得するため。

CVS for GCP 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。	○脚注： 1[]

> 脚注：はい脚注： 1[] ： Astra Trident で管理されていません

「 gcp-cvs 」ドライバは仮想ストレージプールを使用します。仮想ストレージプールはバックエンドを抽象化し、 Astra Trident がボリュームの配置を決定できるようにします。管理者が backend.json ファイルに仮想ストレージプールを定義します。ストレージクラスは、ラベルを使用する仮想ストレージプールを識別します。

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に対応したストレージクラスの設計

フィルタリングは、特定のストレージクラスオブジェクト内で使用でき、そのストレージクラスで使用するストレージプールまたはプールのセットを決定します。ストレージクラスでは'`toragePools’additionalStoragePools’excludeStoragePools" の 3 セットのフィルタを設定できます

'toragePools' パラメータは ' 指定した属性に一致するプールのセットにストレージを制限するのに役立ちます「 additionalStoragePools 」パラメータは、 Astra Trident がプロビジョニングに使用する一連のプールと、属性と「 toragePools 」パラメータで選択した一連のプールを拡張するために使用されます。どちらか一方のパラメータを単独で使用することも、両方を使用して、適切なストレージプールセットが選択されていることを確認することもできます。

excludeStoragePools' パラメータを使用して ' 属性に一致するプールの一覧を除外します

QoS ポリシーをエミュレートするストレージクラスの設計

ストレージクラスを設計して Quality of Service ポリシーをエミュレートする場合は ' 「メディア」属性を「 hdd 」または「 sd 」として ' ストレージクラスを作成しますストレージクラスで言及されている「メディア」属性に基づいて、 Trident は「 hdd 」アグリゲートまたは「 sd 」アグリゲートにメディア属性と一致させる適切なバックエンドを選択し、ボリュームのプロビジョニングを特定のアグリゲートに誘導します。したがって、「メディア」属性が「 SD 」に設定されているストレージクラス Premium を作成して、プレミアム QoS ポリシーに分類できます。メディア属性を「 hdd 」に設定し、標準の QoS ポリシーとして分類できる、別のストレージクラス標準を作成できます。また、ストレージクラスの「 IOPS 」属性を使用して、 QoS ポリシーとして定義できる Element アプライアンスにプロビジョニングをリダイレクトすることもできます。

特定の機能に基づいてバックエンドを利用するストレージクラスの設計

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。

仮想ストレージプールのストレージクラス設計

Virtual Storage Pool 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対して仮想ストレージプールを定義できます。

仮想ストレージ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経由で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する場合、 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。

Virtual Storage Pool Design の略

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。この問題は、仮想ストレージプールの導入に伴って、軽減されています。仮想ストレージプールは、バックエンドと Kubernetes ストレージクラスの間に抽象化されたレベルです。管理者は、 Kubernetes ストレージクラスを介してパラメータとラベルを定義でき、セレクタとしてバックエンドに依存しない方法で参照できます。Virtual Storage Pools は、サポート対象の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。

> 仮想ストレージ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ことを推奨します。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 をエミュレートするための仮想ストレージプールを設計します

サービスクラスをエミュレートするための仮想ストレージプールを設計することができます。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。「高度」のアスペクトを適切なパフォーマンスレベルに設定し、各ラベルの下にその他の必要な側面を追加します。では、別の仮想ストレージプールにマッピングする別の Kubernetes ストレージクラスを作成します。parameters.selector` フィールドを使用すると ' 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

特定の側面を割り当てるための仮想プールを設計します

特定の側面を持つ複数の仮想ストレージプールは、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。次に ' 異なる仮想ストレージ・プールにマップされる parameters.selector` フィールドを使用して ' 異なる Kubernetes ストレージ・クラスを作成しますバックエンドでプロビジョニングされるボリュームには、選択した仮想ストレージ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一部は、 PVC の作成時に Astra Trident のプロビジョニング決定プロセスに影響を与える可能性があります。

アクセスモード

PVC 経由でストレージを要求する場合、必須フィールドの 1 つ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、ボリュームを作成する必要のある Snapshot として「ソース」を指定します。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。後で拡張できるようにするには ' ボリュームに関連づけられたストレージ・クラスで 'allowVolumeExpansion を true に設定します永続ボリュームのサイズを変更する必要がある場合は、Persistent Volume Claim の「 PEC.resources.request.storage 」注釈を必要なボリュームサイズに編集します。Trident が、ストレージクラスタ上のボリュームのサイズ変更を自動的に処理します。

既存のボリュームを **Kubernetes** にインポートする

Volume Import では、既存のストレージボリュームを Kubernetes 環境にインポートできます。これは現在、「 ONTAP-NAS」 、「 ONTAP-NAS-flexgroup 」、「 solidfire-san-」 、「 azure-netapp-files 」、「 gcp-cvs` ドライバ」でサポートされています。この機能は、既存のアプリケーションを Kubernetes に移植する場合や、ディザスタリカバリシナリオで使用する場合に便利です。

ONTAP ドライバと 'olidfire-san`drivers を使用する場合は 'tridentctl import volume <backend-name><volume-name><f/path/pvc.yaml コマンドを使用して 'Astra Trident で管理する既存のボリュームを Kubernetes にインポートしますimport volume コマンドで使用した PVC YAML または JSON ファイルは、 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。

「 azure-NetApp-file' 」または「 gcp-cvs` ドライバが使用されている場合は、「 tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml 」コマンドを使用して、 Astra Trident で管理される Kubernetes にボリュームをインポートします。これにより、ボリューム参照が一意になります。

上記のコマンドを実行すると、 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。ここでは、 OpenShift の初期インストール時にロギングを導入し、 OpenShift のインストール後にロギングを導入するという、 2 つのインストール方法について説明します。

> Red Hat OpenShift バージョン 3.9 以降、データ破損に関する懸念があるため、記録サービスに NFS を使用しないことを公式のドキュメントで推奨しています。これは、 Red Hat 製品のテストに基づいています。ONTAP 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、インストーラが失敗しないように、 Ansible 変数「 OpenShift 」の「 OpenShift 」 enable_unsupported _configurations 」を「 true 」に設定する必要があります。

はじめに

ロギングサービスは、必要に応じて、両方のアプリケーションに導入することも、 OpenShift クラスタ自体のコア動作に導入することもできます。オペレーション・ログを配置する場合 ' 変数 OpenShift の logging_use_ops を true として指定すると ' サービスの 2 つのインスタンスが作成されます操作のロギングインスタンスを制御する変数には「 ops 」が含まれ、アプリケーションのインスタンスには含まれません。

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 」ロギング・ストレージ・タイプ	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift 」ロギング・ストレージ・ホスト	NFS ホストのホスト名または IP アドレス。仮想マシンのデータ LIF に設定してください。

変数（ **Variable** ）	詳細
「 OpenShift 」ロギング・ストレージ・NFS_DIRECT'	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift _logging 」としてジャンクションされている場合、この変数にそのパスを使用します。
「 OpenShift 」ロギング・ストレージ・ボリューム名	作成する PV の名前（「 pv_ose_logs 」など）。
「 OpenShift 」ロギング・ストレージ・ボリューム・サイズ	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
'OpenShift の logging_es_vpc_dynamic	動的にプロビジョニングされたボリュームを使用する場合は true に設定します。
「 OpenShift logging _es_vpc_storage_class_name 」	PVC で使用されるストレージクラスの名前。
「 OpenShift logging _es_vpc_size 」を参照してください	PVC で要求されたボリュームのサイズ。
「 OpenShift logging _es_vpc_prefix 」を参照してください	ロギングサービスで使用される PVC のプレフィックス。
'OpenShift の logging_es_ops_pvc_dynamic	動的にプロビジョニングされたボリュームを ops ロギングインスタンスに使用するには、「 true 」に設定します。
「 OpenShift logging _es_ops_pvc_storage_class_name 」を参照してください	処理ロギングインスタンスのストレージクラスの名前。
'OpenShift logging _es_ops_pvc_size	処理インスタンスのボリューム要求のサイズ。
「 OpenShift logging _es_ops_pvc_prefix 」を参照してください	ops インスタンス PVC のプレフィックス。

ロギングスタックを導入します

初期の OpenShift インストールプロセスの一部としてロギングを導入する場合、標準の導入プロセスに従うだけで済みます。Ansible は、必要なサービスと OpenShift オブジェクトを構成および導入して、 Ansible が完了したらすぐにサービスを利用できるようにします。

ただし、最初のインストール後に導入する場合は、コンポーネントプレイブックを Ansible で使用する必要があります。このプロセスは、 OpenShift のバージョンが異なるためわずかに変更される場合があるので、必ず読んで従うようにしてください "Red Hat OpenShift Container Platform 3.11 のドキュメント" 使用しているバージョンに対応した

指標サービス

この指標サービスは、 OpenShift クラスタのステータス、リソース利用率、可用性に関する重要な情報を管理者に提供します。ポッドの自動スケール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

クアプリケーションのためにメトリックサービスから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 」	インストーラがログサービス用の NFS PV を作成するように 'NFS' に設定します
「 OpenShift _ metrics _storage_host 」というようになります	NFS ホストのホスト名または IP アドレス。これは SVM のデータ LIF に設定されている必要があります。
「 OpenShift _ metrics _storage_nfs_directory 」というエラーが表示されます	NFS エクスポートのマウントパス。たとえば、ボリュームが「 /OpenShift メトリック」としてジャンクションされている場合は、この変数にそのパスを使用します。
「 OpenShift _ metrics _storage_volume_name 」という形式で指定します	作成する PV の名前（「 pv_ose_metrics 」など）。
「 OpenShift _ metrics _storage_volume_size 」というようになります	NFS エクスポートのサイズ（例： 100Gi ）

OpenShift クラスタがすでに実行中で、そのため Trident を導入して設定した場合、インストーラは動的プロビジョニングを使用してボリュームを作成できます。次の変数を設定する必要があります。

変数（ **Variable** ）	詳細
「 OpenShift _ metrics _ cassandra_vpc_prefix 」という形式で指定します	メトリック PVC に使用するプレフィックス。
「 OpenShift _ metrics _ cassandra_vp_size' 」のようになります	要求するボリュームのサイズ。
「 OpenShift _ metrics _ cassandra_storage_type 」のようになります	指標に使用するストレージのタイプ。適切なストレージクラスを使用して PVC を作成するには、 Ansible に対してこれを dynamic に設定する必要があります。
「 OpenShift _ metrics _cassanda_pvc_storage_class_name 」という形式で指定します	使用するストレージクラスの名前。

指標サービスを導入する

ホスト / インベントリファイルに適切な Ansible 変数を定義して、 Ansible でサービスを導入します。OpenShift インストール時に導入する場合は、 PV が自動的に作成されて使用されます。コンポーネントプレイブックを使用して導入する場合、 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

NetApp のストレージ・プラットフォームが提供するデータ保護およびリカバリ機能のオプションについて説明します。Astra Trident では、こうした機能の一部を活用できるボリュームをプロビジョニングできます。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。

をバックアップします `etcd` クラスタデータ

Astra Trident は、メタデータを Kubernetes クラスタの「 etcd」 データベースに格納します。災害発生時に Kubernetes クラスタをリカバリするには 'etcd' クラスタ・データを定期的にバックアップすることが重要です

手順

1. etcdctl snapshot save コマンドを使用すると 'etcd' クラスタのポイント・イン・タイム・スナップショットを取得できます

   ```
   sudo docker run --rm -v /backup:/backup \
     --network host \
     -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \
     --env ETCDCTL_API=3 \
     k8s.gcr.io/etcd-amd64:3.2.18 \
     etcdctl --endpoints=https://127.0.0.1:2379 \
     --cacert=/etc/kubernetes/pki/etcd/ca.crt \
     --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
     --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
     snapshot save /backup/etcd-snapshot.db
   ```

   このコマンドは 'etcd' コンテナをスピンアップして etcd' スナップショットを作成し '/backup' ディレクトリに保存します

2. 災害が発生した場合は、 etcd Snapshot を使用して Kubernetes クラスタをスピンアップできます。etcdctl snapshot restore コマンドを使用して '/var/lib/etcdd フォルダに作成された特定のスナップショットをリストアします復元後、「 /var/lib/etcd` 」フォルダに「 member-」 フォルダが格納されているかどうかを確認します。次に 'etcdctl snapshot restore コマンドの例を示します

   ```
   # etcdctl snapshot restore '/backup/etcd-snapshot-latest.db' ; mv
   /default.etcd/member/ /var/lib/etcd/
   ```

3. Kubernetes クラスタを初期化する前に、必要な証明書をすべてコピーしておきます。
4. 「 --ignore-preflight -errors=DirAvailable --var-lib-etcd` 」フラグを付けてクラスタを作成します。

5. クラスタが起動したら、 kube-system ポッドが起動していることを確認します。
6. 「 kubectl get CRD 」コマンドを使用して、 Trident によって作成されたカスタムリソースが存在するかどうかを確認し、 Trident オブジェクトを取得して、すべてのデータが利用可能であることを確認します。

ONTAP スナップショットを使用して日付をリカバリします

Snapshot は、アプリケーションデータのポイントインタイムリカバリオプションを提供することで重要な役割を果たします。ただし、スナップショットは単独ではバックアップされず、ストレージシステムの障害やその他の災害に対する保護は行われません。しかし、ほとんどのシナリオで、データをすばやく簡単にリカバリできる便利な方法です。ONTAP Snapshot テクノロジを使用してボリュームのバックアップを作成する方法とリストアする方法について説明します。

- スナップショット・ポリシーがバックエンドで定義されていない場合 ' デフォルトでは 'none' ポリシーが使用されますそのため、 ONTAP では自動 Snapshot は作成されません。ただし、ストレージ管理者は、 ONTAP 管理インターフェイスから手動で Snapshot を作成したり、 Snapshot ポリシーを変更したりできます。これは Trident の動作には影響しません。
- デフォルトでは、 snapshot ディレクトリは表示されません。これにより 'ONTAP-NAS' および ONTAP -NAS-エコノミー のドライバを使用してプロビジョニングされたボリュームの互換性を最大限に高めることができます「 ONTAP - NAS 」および「 ONTAP - NAS - エコノミー」ドライバを使用する場合は、「 .snapshot 」ディレクトリを有効にして、アプリケーションがスナップショットから直接データをリカバリできるようにします。
- volume snapshot restore ONTAP CLI コマンドを使用して ' ボリュームを以前のスナップショットに記録された状態にリストアしますSnapshot コピーをリストアすると、既存のボリューム構成は上書きされます。Snapshot コピーの作成後にボリューム内のデータに加えた変更はすべて失われます。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

ONTAP を使用してデータをレプリケート

データのレプリケートは、ストレージアレイの障害によるデータ損失から保護する上で重要な役割を果たします。

> ONTAP レプリケーションテクノロジの詳細については、を参照してください "ONTAP のドキュメント"。

SnapMirror Storage Virtual Machine （ SVM ）レプリケーション

を使用できます "SnapMirror" 設定とそのボリュームを含む SVM 全体をレプリケートすること。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。

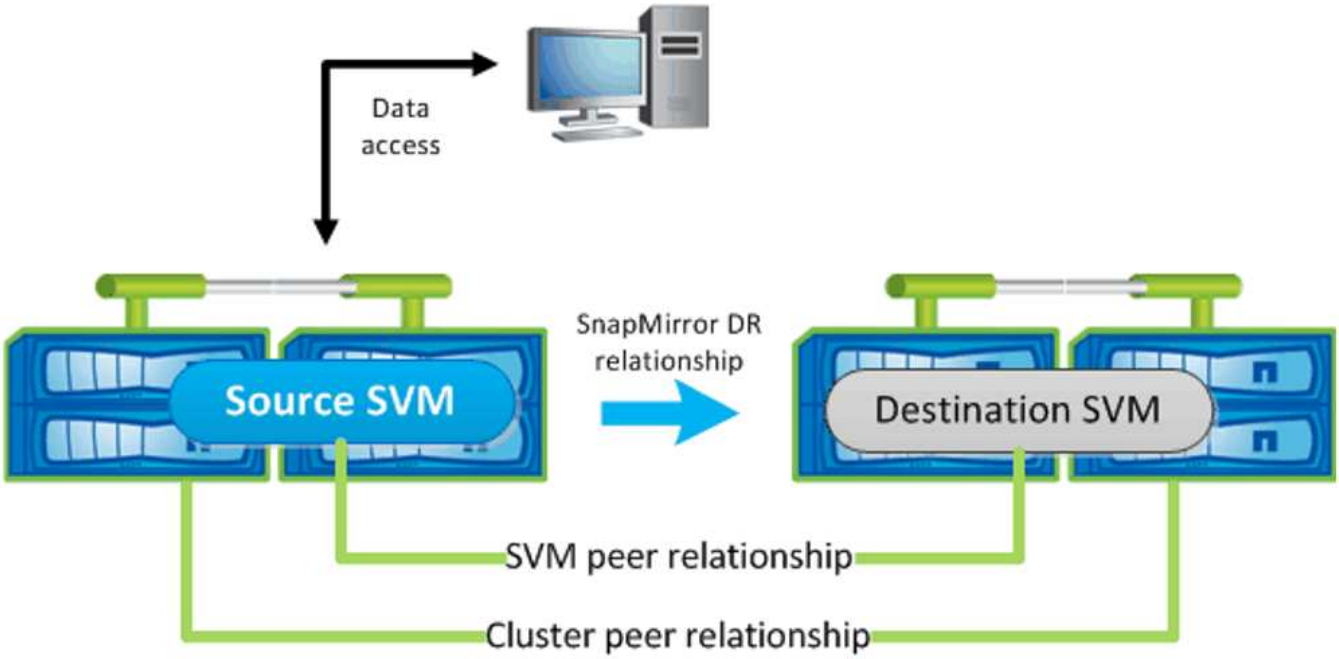
Astra Trident は、レプリケーション関係自体を構成できないため、ストレージ管理者は ONTAP の SnapMirror SVM レプリケーション機能を使用して、ボリュームをディザスタリカバリ（ DR ）デスティネーションに自動的にレプリケートできます。

SnapMirror SVM レプリケーション機能を使用する場合や、現在この機能を使用している場合は、次の点を考慮してください。

- SVM-DR が有効になっている SVM ごとに別個のバックエンドを作成する必要があります。
- レプリケートされたバックエンドを必要な場合を除き選択しないようにストレージクラスを設定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関係の保護をプロビジョニングする必要がないボリュームがある場合、この問題を回避することが重要です。
- アプリケーション管理者は、データのレプリケーションに伴う追加のコストと複雑さを理解し、リカバリプランを決定してから、データレプリケーションを利用する必要があります。
- SnapMirror デスティネーション SVM をアクティブ化する前に、スケジュールされたすべての SnapMirror 転送を停止し、実行中のすべての SnapMirror 転送を中止して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 を起動します。
- Astra Trident では、 SVM の障害は自動では検出されない。したがって、失敗した場合、管理者は tridentctl backend update コマンドを実行して、新しいバックエンドへの Trident のフェイルオーバーをトリガーする必要があります。

SVM のセットアップ手順の概要を次に示します。

- ソースクラスタとデスティネーションクラスタ間にピア関係を設定します。
- subtype dp-destination オプションを使用して、デスティネーション SVM を作成します。
- レプリケーションジョブスケジュールを作成して、必要な間隔でレプリケーションが実行されるようにします。
- 「 -identity-preserve true 」オプションを使用してデスティネーション SVM からソース SVM への SnapMirror レプリケーションを作成し、ソース SVM 構成とソース SVM インターフェイスをデスティネーションにコピーします。デスティネーション SVM から、 SnapMirror SVM レプリケーション関係を初期化します。



Trident のディザスタリカバリワークフロー

Astra Trident 19.07 以降では、 Kubernetes の SSD を使用して独自の状態を保存、管理しています。Kubernetes クラスタのメタデータの保存には 'etcd' が使用されますここでは、 Kubernetes の etcd' デー

タファイルと証明書が NetApp FlexVol に格納されているものとします。この FlexVol は SVM にあり、 SVM の SnapMirror SVM-DR 関係はセカンダリサイトのデスティネーション SVM と一緒にあります。

災害発生時に Astra Trident を使用して、単一のマスター Kubernetes クラスタをリカバリする手順を次に示します。

1. ソース SVM で障害が発生した場合は、 SnapMirror デスティネーション SVM をアクティブ化します。そのためには、スケジュールされた SnapMirror 転送を停止し、実行中の SnapMirror 転送を中止して、レプリケーション関係を解除し、ソース SVM を停止して、デスティネーション SVM を起動します。
2. デスティネーション SVM から、 Kubernetes の「 etcd」 データファイルおよび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されるホストにマウントします。
3. Kubernetes クラスタに関連する必要な証明書をすべて、「 /etc/Kubernetes /pki 」の下にコピーし、「 /var/lib/etcd」 の下に「 etcd`m ember-」 ファイルをコピーします。
4. --ignore-preflight -errors=DirAvailable --var-lib-etcd` フラグを指定して kubeadm init コマンドを使用して Kubernetes クラスタを作成しますKubernetes ノードに使用するホスト名は、ソースの Kubernetes クラスタと同じであることが必要です。
5. 「 kubectl get CRD 」コマンドを実行して、すべての Trident カスタムリソースが起動しているかどうかを確認し、 Trident オブジェクトを取得して、すべてのデータが利用可能であることを確認します。
6. 必要なバックエンドをすべて更新して、新しいデスティネーション SVM 名を反映させるために、「 ./tridentctl update backend <backend-name> -f <backend-json -file> -n <namespace> 」コマンドを実行します。

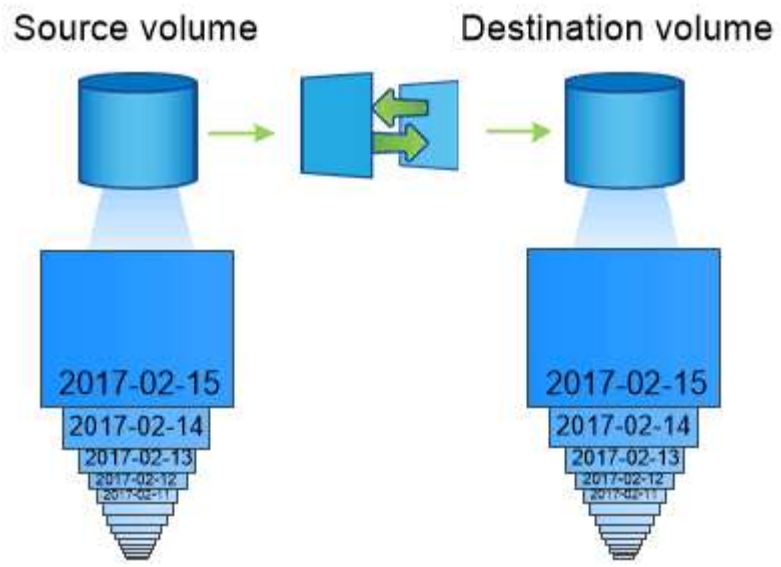
> アプリケーション永続ボリュームの場合、デスティネーション SVM がアクティブ化されると、 Trident によってプロビジョニングされたすべてのボリュームがデータの提供を開始します。前述の手順に従って Kubernetes クラスタをデスティネーション側でセットアップしたら、すべての導入ポッドとポッドが開始され、コンテナ化されたアプリケーションは問題なく実行されます。

SnapMirror ボリュームのレプリケーション

ONTAP SnapMirror ボリュームレプリケーションはディザスタリカバリ機能です。この機能を使用すると、ボリュームレベルでプライマリストレージからデスティネーションストレージにフェイルオーバーできます。SnapMirror は、 Snapshot を同期することで、セカンダリストレージ上のプライマリストレージのボリュームレプリカまたはミラーを作成します。

ONTAP の SnapMirror ボリュームレプリケーションのセットアップ手順の概要を次に示します。

- ボリュームが配置されているクラスタとボリュームからデータを提供する SVM 間のピアリングを設定します。
- 関係の動作を制御する SnapMirror ポリシーを作成し、その関係の設定属性を指定します。
- を使用して、デスティネーションボリュームとソースボリューム間の SnapMirror 関係を作成します "d9934e78a9254dde4a227181c30fa2d2" をクリックし、適切な SnapMirror ポリシーを割り当てます。
- SnapMirror 関係の作成後、ソースボリュームからデスティネーションボリュームへのベースライン転送が完了するように、関係を初期化します。



Trident の **SnapMirror** ボリュームディザスタリカバリワークフロー

Astra Trident で単一のマスター Kubernetes クラスタをリカバリする手順を次に示します。

1. 災害が発生した場合は、スケジュールされたすべての SnapMirror 転送を停止し、実行中のすべての SnapMirror 転送を中止します。デスティネーションボリュームが読み取り / 書き込み可能になるように、デスティネーションボリュームとソースボリュームの間のレプリケーション関係を解除します。
2. デスティネーション SVM から、 Kubernetes の「 etcd」 データファイルおよび証明書が格納されているボリュームをホストにマウントします。このホストはマスターノードとしてセットアップされます。
3. Kubernetes クラスタに関連する必要な証明書をすべて、「 /etc/Kubernetes /pki 」の下にコピーし、「 /var/lib/etcd」 の下に「 etcd`m ember-」 ファイルをコピーします。
4. --ignore-preflight -errors=DirAvailable --var-lib-etcd` フラグを指定して kubeadm init コマンドを実行し、Kubernetes クラスタを作成します。ホスト名はソースの Kubernetes クラスタと同じにする必要があります。
5. 「 kubectl get CRD 」コマンドを実行して、すべての Trident カスタムリソースが起動しているかどうかを確認し、すべてのデータが利用可能であることを確認します。
6. 前のバックエンドをクリーンアップし、 Trident に新しいバックエンドを作成します。デスティネーション SVM の新しい管理 LIF とデータ LIF 、新しい SVM 名、パスワードを指定します。

アプリケーション永続ボリュームのディザスタリカバリワークフロー

次の手順は、災害発生時に SnapMirror デスティネーションボリュームをコンテナ化されたワークロードで使用できるようにする方法を示しています。

1. スケジュールされたすべての SnapMirror 転送を中止し、実行中のすべての SnapMirror 転送を中止します。デスティネーションボリュームが読み取り / 書き込み可能になるように、デスティネーションボリュームとソースボリュームの間のレプリケーション関係を解除します。ソース SVM のボリュームにバインドされた PVC を使用していた環境をクリーンアップします。
2. 前述の手順に従ってデスティネーション側で Kubernetes クラスタをセットアップしたら、 Kubernetes クラスタから導入環境、 PVC 、 PV をクリーンアップします。

3. Trident で新しい管理 LIF とデータ LIF 、デスティネーション SVM の新しい SVM 名とパスワードを指定して、新しいバックエンドを作成します。
4. Trident のインポート機能を使用して、必要なボリュームを、新しい PVC にバインドされた PV としてインポートします。
5. 新しく作成した PVC を使用してアプリケーション展開を再展開します。

Element Snapshot を使用してデータをリカバリします

ボリュームの Snapshot スケジュールを設定し、必要な間隔で Snapshot が作成されていることを確認して、Element ボリューム上のデータをバックアップします。Snapshot スケジュールは、 Element UI または API を使用して設定します。現時点では 'olidfire-san' ドライバを使用して ' スナップショットスケジュールをボリュームに設定することはできません

データが破損した場合は、特定の Snapshot を選択し、 Element UI または API を使用してボリュームを手動で Snapshot にロールバックできます。その Snapshot の作成後にボリュームに対して行われた変更はすべて元に戻ります。

セキュリティ

ここに記載された推奨事項に従って、 Astra Trident のインストールを確実にセキュリティで保護してください。

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（ trident` ）にインストールしてください。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 ネームスペースへのアクセスを管理者のみに許可し、 tridentctl アプリケーションへのアクセスを許可する必要があります。

ONTAP SAN バックエンドで CHAP 認証を使用します

Astra Trident は、 ONTAP SAN ワークロード向けの CHAP ベースの認証をサポートしています（「 ONTAP-SAN' 」ドライバと「 ONTAP-SAN-エコノミー 」ドライバを使用）。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 をセットアップし 'tridentctl を使用して CHAP ユーザー名とシークレットを管理できます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、それぞれの PV の「 tridentvolume 」 CR オブジェクトに格納されます。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。

参照

Astra Trident ポート

Trident が通信するポートの詳細をご確認ください。

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
8000	Trident REST サーバ
17546	Trident デミ作用 / レディネスプローブポートは、 Trident デミ作用ポッドで使用されます

liveness / Readiness プローブポートは '--probe-port' フラグを使用してインストール時に変更できますこのポートがワーカーノード上の別のプロセス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 の REST API を操作するには簡単です。REST エンドポイントは必要に応じて直接使用できます。

これは、 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。

セキュリティを強化するために、 Astra Trident の「 RE ST API 」は、ポッド内で実行される場合のデフォルトで localhost に制限されています。この動作を変更するには、ポッド構成で Astra Trident の「 -address 」引数を設定する必要があります。

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/< オブジェクトタイプ >/< オブジェクト名 >: 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、を参照してください "ce87eb03803d5633c163541464e9e7f2"。

- 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`

`D eleet<trident-address>/trident/v1/<` オブジェクトタイプ `>/<` オブジェクト名 `>` ：名前付きリソースを削除します。

> バックエンドまたはストレージクラスに関連付けられているボリュームは削除されず、削除されません。詳細については、を参照してください "ce87eb03803d5633c163541464e9e7f2"。

これらの API の呼び出し方法の例については 'debug(d`) フラグを渡してください詳細については、を参照してください "ce87eb03803d5633c163541464e9e7f2"。

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>` ：ロギングレベルを設定します（ debug 、 info 、 warn 、 error 、 fatal ）。デフォルトは info です。

Kubernetes

- `-k8s_pod `: Kubernetes サポートを有効にするには ' このオプションまたは -k8s_api_server を使用しますこれを設定すると、 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 -k8s_pod を使用して Kubernetes サポートを有効にしますTrident を指定すると、セキュアでないアドレスとポートを使用して Kubernetes API サーバに接続されます。これにより、 Trident をポッドの外部に導入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、「 -k8s_pod 」オプションを使用してポッドに Trident を導入します。
- `-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 プラグインを登録するときに使用するドライバ名。デフォルトは 'NetApp' です
- `-ddriver_port <port-number>`: このポートでは 'UNIX ドメインソケットではなくリッスンします
- `-config <file>`: 必須。このパスをバックエンド構成ファイルに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。ポッドの IP アドレスから REST インターフェイスにアクセスできるようにするには、「 -address 」を使用します。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）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 インタフェー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。このサービスを使用すると、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. ユーザーは、「 PersistentVolumeClaim 」を作成して、特定のサイズの新しい「 PersistentVolume 」を、管理者が以前に設定した Kubernetes の「 torageClass 」から要求します。

2. Kubernetes の「 torageClass 」は、 Trident をプロビジョニングツールとして識別し、要求されたクラスのボリュームのプロビジョニング方法を Trident に指示するパラメータを含んでいます。
3. Trident は、対応する「 Backends 」と「 toragePools 」を識別する同じ名前の「 S torageClass 」を参照します。この名前は、このクラスのボリュームのプロビジョニングに使用できます。
4. Trident は、対応するバックエンドにストレージをプロビジョニングし、 2 つのオブジェクトを作成します。 Kubernetes では、「 PersistentVolume 」とは、ボリュームを検索、マウント、処理する方法を Kubernetes に伝える「 PersistentVolume 」と、「 PersistentVolume 」と実際のストレージの関係を保持する Trident 内のボリュームです。
5. Kubernetes は 'PersistentVolumeClaim を新しい 'PersistentVolume' にバインドしますPersistentVolume が実行される任意のホストに PersistentVolume をマウントする 'PersistentVolumeClaim を含むポッド。
6. ユーザーは、 Trident を指す「 VolumeSnapshotClass 」を使用して、既存の PVC の「 VolumeSnapshot 」を作成します。
7. Trident が PVC に関連付けられているボリュームを特定し、バックエンドにボリュームの Snapshot を作成します。また ' スナップショットの識別方法を Kubernetes に指示する 'VolumeSnapshotContent' も作成します
8. ユーザーは 'VolumeSnapshot' をソースとして使用して 'PersistentVolumeClaim を作成できます
9. Trident は必要なスナップショットを識別し、「 PersistentVolume 」と「 Volume 」の作成に関連する一連のステップを実行します。

Kubernetes オブジェクトの詳細については、を参照することを強く推奨します "永続ボリューム" Kubernetes のドキュメントのセクション。

Kubernetes `PersistentVolumeClaim` オブジェクト

Kubernetes の「 PersistentVolumeClaim 」オブジェクトは、 Kubernetes クラスタユーザが作成したストレージの要求です。

Trident では、標準仕様に加えて、バックエンド構成で設定したデフォルト設定を上書きする場合に、ボリューム固有の次のアノテーションを指定できます。

アノテーション	ボリュームオプション	サポートされているドライバ
trident.netapp.io/fileSystem	ファイルシステム	ONTAP-SAN 、 solidfire-san-SAN 、 eseries-iscsi 、 ONTAP-SAN-エコノミー の 2 つのシステムがあります
trident.netapp.io/cloneFromPVC	cloneSourceVolume の実行中です	ontap - NAS 、 ontap - san 、 solidfire-san-files 、 gcvs 、 ONTAP - SAN - 経済性
trident.netapp.io/splitOnClone	splitOnClone	ONTAP - NAS 、 ONTAP - SAN
trident.netapp.io/protocol	プロトコル	任意
trident.netapp.io/exportPolicy	エクスポートポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/snapshotPolicy	Snapshot ポリシー	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAP-SAN

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/snapshotReserve	Snapshot リザーブ	ONTAP-NAS 、 ONTAP-NAS-flexgroup 、 ONTAP-SAN 、 GCP-cvs
trident.netapp.io/snapshotDirectory	snapshotDirectory の略	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup
trident.netapp.io/blockSize	ブロックサイズ	solidfire - SAN

作成された PV に「削除」再利用ポリシーがある場合、 Trident は PV が解放されると（つまり、ユーザーが PVC を削除したときに）、 PV と元のボリュームの両方を削除します。削除操作が失敗した場合、 Trident は PV をマークします。そのような状態で操作が成功するか、 PV が手動で削除されるまで、定期的に再試行します。PV が「 + Retain + 」ポリシーを使用している場合、 Trident はそのポリシーを無視し、管理者が Kubernetes とバックエンドからクリーンアップすると想定します。これにより、ボリュームを削除する前にバックアップまたは検査を行うことができます。PV を削除しても、原因 Trident で元のボリュームが削除されないことに注意してください。REST API (tridentctl') を使用して削除してください。

Trident では CSI 仕様を使用したボリュームスナップショットの作成がサポートされています。ボリュームスナップショットを作成し、それをデータソースとして使用して既存の PVC のクローンを作成できます。これにより、 PVS のポイントインタイムコピーを Kubernetes にスナップショットの形で公開できます。作成した Snapshot を使用して新しい PVS を作成できます。「 +On-Demand Volume Snapshots + 」を見て、これがどのように機能するかを確認してください。

Trident には ' クローン作成用の cloneFromPVC' および 'plitOnClone' 注釈も用意されていますCSI 実装（ Kubernetes 1.13 以前）を使用しなくても、または Kubernetes リリースがベータ版のボリュームスナップショット（ Kubernetes 1.16 以前）をサポートしていない場合は、これらのアノテーションを使用して PVC のクローンを作成できます。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。

> CSI Trident では 'cloneFromPVC' および 'plitOnClone' 注釈を使用できますまた ' 従来の CSI 以外のフロントエンドも使用できます

例 : ユーザがすでに「 m ysql 」という PVC を持っている場合、ユーザは「 trident.netapp.io/cloneFromPVC: mysql 」などの注釈を使用して「 m ysqlclone 」という新しい PVC を作成できます。このアノテーションセットを使用すると、 Trident はボリュームをゼロからプロビジョニングするのではなく、 MySQL PVC に対応するボリュームのクローンを作成します。

次の点を考慮してください。

- アイドルボリュームのクローンを作成することを推奨します。
- PVC とそのクローンは、同じ Kubernetes ネームスペースに存在し、同じストレージクラスを持つ必要があります。
- また 'ONTAP-NAS' および 'ONTAP-SAN' ドライバを使用すると 'pvc 注釈 trident.netapp.io/splitOnClone` を trident.netapp.io/cloneFromPVC` と組み合わせて設定することが望ましい場合がありますTrident は 'trident.netapp.io/splitOnClone` を true に設定した場合 ' クローン・ボリュームを親ボリュームからスプリットするため ' 一部のストレージ効率を失うことなく ' クローン・ボリュームのライフサイクルを親ボリュームから完全に分離しますtrident.netapp.io/splitOnClone` を設定したり 'false に設定したりしないと ' 親ボリュームとクローンボリューム間の依存関係を作成する代わりに ' バックエンドでのスペース消費が削

減されますこれにより ' クローンを最初に削除しない限り ' 親ボリュームを削除できなくなりますクローンをスプリットするシナリオでは、空のデータベースボリュームをクローニングする方法が効果的です。このシナリオでは、ボリュームとそのクローンで使用するデータベースボリュームのサイズが大きく異なっており、 ONTAP ではストレージ効率化のメリットはありません。

「 sample-input' 」ディレクトリには、 Trident で使用する PVC 定義の例が含まれています。Trident ボリュームに関連するパラメータと設定の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 の 'PersistentVolume' オブジェクトは 'Kubernetes クラスタで利用できるようになったストレージの一部ですポッドに依存しないライフサイクルがあります。

> Trident は 'PersistentVolume' オブジェクトを作成し ' プロビジョニングするボリュームに基づいて自動的に Kubernetes クラスタに登録します自分で管理することは想定されていません。

Trident をベースとする「 torageClass 」を参照する PVC を作成すると、 Trident は対応するストレージクラスを使用して新しいボリュームをプロビジョニングし、そのボリュームに新しい PV を登録します。プロビジョニングされたボリュームと対応する PV の構成では、 Trident は次のルールに従います。

- Trident は、 Kubernetes に PV 名を生成し、ストレージのプロビジョニングに使用する内部名を生成します。どちらの場合も、名前がスコープ内で一意であることが保証されます。
- ボリュームのサイズは、 PVC で要求されたサイズにできるだけ近いサイズに一致しますが、プラットフォームによっては、最も近い割り当て可能な数量に切り上げられる場合があります。

Kubernetes `StorageClass` オブジェクト

Kubernetes の「 torageClass 」オブジェクトは、「 PersistentVolumeClaims 」内の名前によって指定され、一連のプロパティを持つストレージをプロビジョニングします。ストレージクラス自体が、使用するプロビジョニングツールを特定し、プロビジョニングツールが理解できる一連のプロパティを定義します。

管理者が作成および管理する必要がある 2 つの基本オブジェクトのうちの 1 つです。もう 1 つは Trident バックエンドオブジェクトです。

Trident を使用する Kubernetes の「 torageClass 」オブジェクトは次のようになります。

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters:
  <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

これらのパラメータは Trident 固有で、クラスのボリュームのプロビジョニング方法を Trident に指示します。

ストレージクラスのパラメータは次のとおりです。

属性	を入力します	必須	説明
属性（ Attributes ）	[string] 文字列をマップします	いいえ	後述の「属性」セクションを参照してください
ストレージプール	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
AdditionalStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング
excludeStoragePools	[string] StringList をマップします	いいえ	内のストレージプールのリストへのバックエンド名のマッピング

ストレージ属性とその有効な値は、ストレージプールの選択属性と Kubernetes 属性に分類できます。

ストレージプールの選択の属性

これらのパラメータは、特定のタイプのボリュームのプロビジョニングに使用する Trident で管理されているストレージプールを決定します。

属性	を入力します	値	提供	リクエスト	でサポートされます
メディア ^1	文字列	HDD 、ハイブリッド、 SSD	プールにはこのタイプのメディアが含まれています。ハイブリッドは両方を意味します	メディアタイプが指定されました	ONTAPNAS 、 ONTAPNAS エコノミー、 ONTAP-NAS-flexgroup 、 ONTAPSAN 、 solidfire-san-SAN 、 solidfire-san-SAN のいずれかに対応しています
プロビジョニングタイプ	文字列	シン、シック	プールはこのプロビジョニング方法をサポートします	プロビジョニング方法が指定されました	シック： All ONTAP & eseries-iscsi ； thin ： all ONTAP & solidfire-san-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
backendType	文字列	ONTAPNAS 、ONTAPNAS エコノミー、ONTAP-NAS-flexgroup 、ONTAPSAN 、solidfire-san-SAN 、 solidfire-san-SAN 、 E シリーズ - iSCSI 、 GCP-cvs 、azure-NetApp-files 、 ONTAP-SAN-Eエコノミー	プールはこのタイプのバックエンドに属しています	バックエンドが指定されて	すべてのドライバ
Snapshot	ブール値	true false	プールは、Snapshot を含むボリュームをサポートします	Snapshot が有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
クローン	ブール値	true false	プールはボリュームのクローニングをサポートします	クローンが有効なボリューム	ONTAP-NAS, ONTAP-SAN, solidfire-san-, gcvs
暗号化	ブール値	true false	プールでは暗号化されたボリュームをサポート	暗号化が有効なボリューム	ONTAP-NAS 、ONTAP-NAS-エコノミー 、ONTAP-NAS-FlexArray グループ、 ONTAP-SAN
IOPS	整数	正の整数	プールは、この範囲内で IOPS を保証する機能を備えています	ボリュームで IOPS が保証されました	solidfire - SAN

^1 ^ ： ONTAP Select システムではサポートされていません

ほとんどの場合、要求された値はプロビジョニングに直接影響します。たとえば、シックプロビジョニングを要求した場合、シックプロビジョニングボリュームが使用されます。ただし、 Element ストレージプールでは、提供されている IOPS の最小値と最大値を使用して、要求された値ではなく QoS 値を設定します。この場合、要求された値はストレージプールの選択のみに使用されます。

理想的には ' 属性だけを使用して ' 特定のクラスのニーズを満たすために必要なストレージの特性をモデル化できますTrident は ' 指定した属性の *ALL* に一致するストレージ・プールを自動的に検出して選択します

「 attributes 」を使用してクラスに適切なプールを自動的に選択できない場合は、「 toragePools 」および「 additionalStoragePools 」パラメータを使用してプールをさらに改良したり、特定のプールセットを選択したりできます。

'toragePools' パラメータを使用すると ' 指定した属性に一致するプールのセットをさらに制限できますつまり 'attributes' パラメータと 'toragePools' パラメータで指定されたプールの交点をプロビジョニングに使用します どちらか一方のパラメータを単独で使用することも、両方を同時に使用することも

「 additionalStoragePools 」パラメータを使用すると、「 attributes 」パラメータと「 toragePools 」パラメータで選択されたプールに関係なく、 Trident がプロビジョニングに使用するプールのセットを拡張できます。

excludeStoragePools' パラメータを使用して、 Trident がプロビジョニングに使用するプールのセットをフィルタリングできます。このパラメータを使用すると、一致するプールがすべて削除されます。

'toragePools' パラメータと 'additionalStoragePools' パラメータでは ' 各エントリは '<backend>:<storagePoolList>' の形式で指定したバックエンドのストレージプールのカンマ区切りリストです たとえば、「 additionalStoragePools 」の値は「 ontapnas_192.168.1.100 ： aggr1 、 aggr2 ； solidfire_192.168.1.101 ： bronze 」のようになります。これらのリストでは、バックエンド値とリスト値の両方に正規表現値を使用できます。tridentctl get backend を使用してバックエンドとそのプールのリストを取得できます

Kubernetes の属性

これらの属性は、動的プロビジョニングの際に Trident が選択するストレージプール / バックエンドには影響しません。代わりに、 Kubernetes Persistent Volume でサポートされるパラメータを提供するだけです。ワーカーノードはファイルシステムの作成操作を担当し、 xfsprogs などのファイルシステムユーティリティを必要とする場合があります。

属性	を入力します	値	説明	関連するドライバ	**Kubernetes** のバージョン
FSstype （英語）	文字列	ext4 、 ext3 、 xfs など	ブロックボリュームのファイルシステムのタイプ	solidfire-san-group 、 ontap/nas 、 ontap -nas-エコノミー 、 ontap -nas-flexgroup 、 ontap -san 、 ONTAP - SAN 経済、 eseries-iscsi	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

属性	を入力します	値	説明	関連するドライバ	**Kubernetes** のバージョン
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	1.17 以上

- SAN LUN に必要なファイル・システム・タイプを制御するには 'fsType' パラメータを使用しますまた 'Kubernetes は ' ストレージ・クラスに fsType が存在することも使用して ' ファイル・システムが存在することを示しますボリューム所有権は 'fsType' が設定されている場合に限り 'pod の fsGroup' セキュリティコンテキストを使用して制御でき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" 「 fsGroup 」コンテキスト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。次の場合にのみ 'fsGroup' 値が適用されます
  - 「 fsType 」はストレージクラスで設定されます。
  - PVC アクセスモードは RWO です。

  NFS ストレージドライバの場合、 NFS エクスポートにはファイルシステムがすでに存在します。fsGroup を使用するには ' ストレージ・クラスで fsType を指定する必要があります この値は 'NFS' に設定することも ' ヌル以外の任意の値に設定することもできます
- を参照してください "ボリュームを展開します" ボリューム拡張の詳細については、を参照してください。
- Trident インストーラバンドルには、「 `sample -input/storageclass-*.yaml 」で Trident で使用するストレージクラス定義の例がいくつか用意されています。Kubernetes ストレージクラスを削除すると、対応する Trident ストレージクラスも削除されます。

Kubernetes `VolumeSnapshotClass` オブジェクト

Kubernetes 'VolumeSnapshotClass' オブジェクトは 'S torageClasses' に似ていますこの Snapshot コピーは、複数のストレージクラスの定義に役立ちます。また、ボリューム Snapshot によって参照され、Snapshot を必要な Snapshot クラスに関連付けます。各ボリューム Snapshot は、単一のボリューム Snapshot クラスに関連付けられます。

スナップショットを作成するには 'VolumeSnapshotClass' を管理者が定義する必要がありますボリューム Snapshot クラスは、次の定義で作成されます。

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

「 driver 」は、「 csi-snapclass 」クラスのボリュームスナップショットの要求が Trident によって処理される Kubernetes を指定します。「要素ポリシー」は、スナップショットを削除する必要がある場合に実行され

るアクションを指定します。「削除ポリシー」が「削除」に設定されている場合、 Snapshot を削除すると、ボリューム Snapshot オブジェクトおよびストレージクラスタ上の基盤となる Snapshot は削除されます。または、「 Retain 」に設定すると、「 VolumeSnapshotContent 」と物理スナップショットが保持されます。

Kubernetes `VolumeSnapshot` オブジェクト

Kubernetes の VolumeSnapshot オブジェクトは ' ボリュームのスナップショットを作成する要求ですPVC がボリュームに対するユーザからの要求を表すのと同様に、ボリュームスナップショットは、ユーザが既存の PVC のスナップショットを作成する要求です。

ボリュームスナップショット要求が受信されると、 Trident はバックエンドでのボリュームのスナップショット作成を自動的に管理し、ユニークな「 VolumeSnapshotContent 」オブジェクトを作成することによってスナップショットを公開します。既存の PVC からスナップショットを作成し、新しい PVC を作成するときにスナップショットを DataSource として使用できます。

> VolumeSnapshot のライフサイクルはソース PVC とは無関係です。ソース PVC が削除されても、スナップショットは維持されます。スナップショットが関連付けられている PVC を削除すると、 Trident はその PVC のバッキングボリュームを **Deleting** 状態でマークしますが、完全には削除しません。関連付けられている Snapshot がすべて削除されると、ボリュームは削除されます。

Kubernetes `VolumeSnapshotContent` オブジェクト

Kubernetes の「 VolumeSnapshotContent 」オブジェクトは、すでにプロビジョニングされているボリュームから取得されたスナップショットを表します。これは「 PersistentVolume 」と似ており、ストレージ・クラスタ上でプロビジョニングされた Snapshot を表します。「 PersistentVolumeClaim 」および「 PersistentVolume 」オブジェクトと同様に、スナップショットが作成されると、「 VolumeContent Snapshot」 オブジェクトは「 VolumeSnapshot 」オブジェクトへの 1 対 1 のマッピングを保持します。これは、スナップショットの作成を要求しました。

> Trident は「 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自動的に Kubernetes クラスタに登録します。自分で管理することは想定されていません。

「 VolumeSnapshotContent 」オブジェクトには、スナップショットを一意に識別する詳細（「 napshotHandle 」など）が含まれています。この「 napshotHandle 」は、 PV の名前と「 VolumeSnapshotContent 」オブジェクトの名前を組み合わせた一意のものです。

Trident では、スナップショット要求を受信すると、バックエンドにスナップショットが作成されます。スナップショットが作成されると、 Trident は「 VolumeSnapshotContent 」オブジェクトを構成し、そのスナップショットを Kubernetes API に公開します。

Kubernetes `CustomResourceDefinition` オブジェクト

Kubernetes カスタムリソースは、管理者が定義した Kubernetes API 内のエンドポイントであり、類似するオブジェクトのグループ化に使用されます。Kubernetes では、オブジェクトのコレクションを格納するためのカスタムリソースの作成をサポートしています。これらのリソース定義を取得するには 'kubectl get CRDs' を実行します

カスタムリソース定義（ CRD ）と関連するオブジェクトメタデータは、 Kubernetes によってメタデータス

トアに格納されます。これにより、 Trident の独立したストアが不要になります。

19.07 リリース以降、 Trident は多数の「 CustomResourceDefinition 」オブジェクトを使用して、 Trident バックエンド、 Trident ストレージクラス、 Trident ボリュームなどの Trident オブジェクトの ID を保持します。これらのオブジェクトは Trident によって管理されます。また、 CSI のボリュームスナップショットフレームワークには、ボリュームスナップショットの定義に必要ないくつかの SSD が導入されています。

CRD は Kubernetes の構成要素です。上記で定義したリソースのオブジェクトは Trident によって作成されます。簡単な例として 'tridentctl' を使用してバックエンドを作成すると ' 対応する tridentBackendsCRD オブジェクトが Kubernetes によって消費されるように作成されます

Trident の CRD については、次の点に注意してください。

- Trident をインストールすると、一連の CRD が作成され、他のリソースタイプと同様に使用できるようになります。
- Trident の以前のバージョンからアップグレードする場合（ステートを維持するために「 etcd」 を使用したもの）、 Trident インストーラは「 etcd' キーバリューデータストア」からデータを移行し、対応する CRD オブジェクトを作成します。
- tridentctl uninstall コマンドを使用して Trident をアンインストールすると、 Trident ポッドは削除されますが、作成された CRD はクリーンアップされません。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 は 'Kubernetes のプロビジョニング・フィールドに csi.trident.netapp.io`/`netapp.io/trident` を指定する 'S torageClass' オブジェクトに一致するストレージ・クラスを作成しますストレージクラス名は、そのストレージクラスが表す Kubernetes の「 torageClass 」オブジェクトの名前と一致します。

> Kubernetes では、 Trident をプロビジョニングツールとして使用する Kubernetes 「 torageClass 」が登録されると、これらのオブジェクトが自動的に作成されます。

ストレージクラスは、ボリュームの一連の要件で構成されます。Trident は、これらの要件と各ストレージプール内の属性を照合し、一致する場合は、そのストレージプールが、そのストレージクラスを使用するボリュームのプロビジョニングの有効なターゲットになります。

REST API を使用して、ストレージクラスを直接定義するストレージクラス設定を作成できます。ただし、Kubernetes の導入では、新しい Kubernetes の「 torageClass 」オブジェクトを登録するときに、これらのオブジェクトが作成されることを期待しています。

Trident バックエンドオブジェクト

バックエンドとは、 Trident がボリュームをプロビジョニングする際にストレージプロバイダを表します。 1 つの Trident インスタンスであらゆる数のバックエンドを管理できます。

> これは、自分で作成および管理する 2 つのオブジェクトタイプのうちの 1 つです。もう 1 つは、 Kubernetes の「 torageClass 」オブジェクトです。

これらのオブジェクトの作成方法の詳細については、バックエンド構成を参照してください。

Trident `StoragePool` オブジェクト

ストレージプールは、各バックエンドでのプロビジョニングに使用できる個別の場所を表します。ONTAP の場合、これらは SVM 内のアグリゲートに対応します。NetApp HCI / SolidFire では、管理者が指定した QoS 帯域に対応します。Cloud Volumes Service の場合、これらはクラウドプロバイダのリージョンに対応します。各ストレージプールには、パフォーマンス特性とデータ保護特性を定義するストレージ属性があります。

他のオブジェクトとは異なり、ストレージプールの候補は常に自動的に検出されて管理されます。

Trident `Volume` オブジェクト

ボリュームは、 NFS 共有や iSCSI LUN などのバックエンドエンドエンドエンドポイントで構成される、プロビジョニングの基本単位です。Kubernetes では ' これらは 'PersistentVolumes' に直接対応しますボリュームを作成するときは、そのボリュームにストレージクラスが含まれていることを確認します。このクラスによって、ボリュームをプロビジョニングできる場所とサイズが決まります。

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

関連付けられた Snapshot がある PV を削除すると、対応する Trident ボリュームが * Deleting * 状態に更新されます。Trident ボリュームを削除するには、ボリュームの Snapshot を削除する必要があります。

ボリューム構成は、プロビジョニングされたボリュームに必要なプロパティを定義します。

属性	を入力します	必須	説明
バージョン	文字列	いいえ	Trident API のバージョン（「 1 」）
名前	文字列	はい。	作成するボリュームの名前
ストレージクラス	文字列	はい。	ボリュームのプロビジョニング時に使用するストレージクラス
サイズ	文字列	はい。	プロビジョニングするボリュームのサイズ（バイト単位）
プロトコル	文字列	いいえ	使用するプロトコルの種類：「 file 」または「 block 」
インターン名	文字列	いいえ	Trident が生成した、ストレージシステム上のオブジェクトの名前
cloneSourceVolume の実行中です	文字列	いいえ	ONTAP （ NAS 、 SAN ） & SolidFire - * ：クローン元のボリュームの名前

属性	を入力します	必須	説明
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 は ' ボリュームの作成時に internalName を生成しますこの構成は 2 つのステップで構成されます。最初に、ストレージプレフィックス（デフォルトの「 trident 」またはバックエンド構成のプレフィックス）をボリューム名の前に付加し、「 <prefix> - <volume-name> 」という形式の名前を付けます。その後、名前の完全消去が行われ、バックエンドで許可されていない文字が置き換えられます。ONTAP バックエンドでは、ハイフンをアンダースコアで置き換えます（つまり、内部名は「 <prefix>_<volume-name>` 」になります）。Element バックエンドの場合、アンダースコアはハイフンに置き換えられます。

ボリューム設定を使用して、 REST API を使用してボリュームを直接プロビジョニングできますが、Kubernetes 環境では、ほとんどのユーザが標準の Kubernetes の「 PersistentVolumeClaim 」メソッドを使用することを想定しています。Trident は、プロビジョニングプロセスの一環として、このボリュームオブジェクトを自動的に作成します。

Trident `Snapshot` オブジェクト

Snapshot はボリュームのポイントインタイムコピーで、新しいボリュームのプロビジョニングやリストア状態に使用できます。Kubernetes では ' これらは 'VolumeSnapshotContent' オブジェクトに直接対応します各 Snapshot には、 Snapshot のデータのソースであるボリュームが関連付けられます。

個々の「スナップショット」オブジェクトには、以下のプロパティが含まれています。

属性	を入力します	必須	説明
バージョン	文字列	はい。	Trident API のバージョン（「 1 」）
名前	文字列	はい。	Trident Snapshot オブジェクトの名前

属性	を入力します	必須	説明
インターン名	文字列	はい。	ストレージシステム上の Trident Snapshot オブジェクトの名前
ボリューム名	文字列	はい。	Snapshot を作成する永続的ボリュームの名前
ボリュームの内部名	文字列	はい。	ストレージシステムに関連付けられている Trident ボリュームオブジェクトの名前

Kubernetes では、これらのオブジェクトが自動的に管理されます。Trident がプロビジョニングしたものを表示できます。

Kubernetes の「 VolumeSnapshot 」オブジェクト要求が作成されると、 Trident は元のストレージシステム上にスナップショットオブジェクトを作成することによって動作します。このスナップショットオブジェクトの「 internalName 」は、プレフィックス「 snapshot-` と「 VolumeSnapshot 」オブジェクトの「 UID 」を組み合わせることによって生成されます（例：「 snapshot-e8d8d8a0ca-9826-11e9-9807-525400f3f660 」）。「 volumeName 」と「 volumeInternalName 」には、バッキングボリュームの詳細を取得して値を設定します。

tridentctl コマンドとオプション

。 "Trident インストーラバンドル" Astra Trident へのシンプルなアクセスを提供するコマンドラインユーティリティ「 tridentctl 」が含まれています。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。

使用方法については 'tridentctl --help' を実行してください

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 にリソースを追加します。
- 「削除」 : Astra Trident から 1 つ以上のリソースを削除します。
- 「 GET 」： Astra Trident から 1 つ以上のリソースを取得します。
- 「 help 」：任意のコマンドに関するヘルプ。
- `imag`: Astra Trident が必要とするコンテナイメージのテーブルを印刷しなさい。
- import` ：既存のリソースを Astra Trident にインポートします。

- install` ： Astra Trident をインストールします。
- logs` ： Astra Trident からログを印刷します。
- 'send': Astra Trident からリソースを送信します
- uninstall: Astra Trident をアンインストールします。
- `update` ： Astra Trident のリソースを変更します。
- アップグレード： Astra Trident のリソースをアップグレードします。
- 've rsion ： Astra Trident のバージョンを印刷します。

フラグ：

- `-d, --debug`: デバッグ出力。
- `-h, --help` ： tridentctl のヘルプ。
- `n,--namespace string`: Astra Trident 配備の名前空間。
- `-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 インタフェースのアドレス / ポート。

> Trident REST インターフェイスは、 127.0.0.1 （ IPv4 の場合）または [::1] （ IPv6 の場合）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`

create コマンドを実行して 'Astra Trident にリソースを追加できます

```
Usage:
  tridentctl create [option]
```

利用可能なオプション : backend`: Astra Trident にバックエンドを追加します。

`delete`

「削除」コマンドを実行すると、 Astra Trident から 1 つ以上のリソースを削除できます。

```
Usage:
  tridentctl delete [option]
```

使用可能なオプション：

- backend`: Astra Trident から 1 つ以上のストレージバックエンドを削除します。

- 'node`: Astra Trident から 1 つ以上の CSI ノードを削除します。
- 'Snapshot` ： Astra Trident から 1 つ以上のボリュームスナップショットを削除します。
- 'torageclas' ： Astra Trident から 1 つ以上のストレージクラスを削除します
- 'volume` ： Astra Trident から 1 つ以上のストレージボリュームを削除します

get

get コマンドを実行すると 'Astra Trident から 1 つ以上のリソースを取得できます

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

使用可能なオプション：

- backend`: Astra Trident から 1 つ以上のストレージバックエンドを取得します。
- 'Snapshot` ： Astra Trident から 1 つ以上のスナップショットを取得します。
- 「 torageclass 」： Astra Trident から 1 つ以上のストレージクラスを取得します。
- 'volume`: Astra Trident から 1 つ以上のボリュームを取得します

images

「 images 」フラグを実行すると、 Astra Trident が必要とするコンテナイメージのテーブルを印刷できます。

```
Usage:
  tridentctl images [flags]
```

flags:*`-h, --help`: イメージのヘルプ。*`-v ,--k8s-version string`: Kubernetes クラスタのセマンティックバージョン。

import volume

import volume コマンドを実行して、既存のボリュームを Astra Trident にインポートできます。

```
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。

- ``-- 管理なし `: 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

"install" フラグを実行して、 Astra Trident をインストールできます。

```
Usage:
  tridentctl install [flags]
```

フラグ：

- `--autosupport-image string` ： AutoSupport Telemetry のコンテナイメージ（デフォルトは「 NetApp/trident autosupport ： 20.07.0 」）。
- `-- autosupport - proxy string` : AutoSupport Telemetry を送信するためのプロキシのアドレス / ポート。
- `--csI`: CSI Trident をインストールします (Kubernetes 1.13 のみをオーバーライドしますが、機能ゲートが必要です) 。
- `--enable-node-prep : 必要なパッケージをノードにインストールしようとします
- ``--generate-custom-yaml `: インストールしないで 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オーバーライドします (デフォルトは 1 分 30s) 。
- `-- image-registry string : 内部イメージレジストリのアドレス / ポート。
- `--k8s-timeout duration`: すべての Kubernetes 操作のタイムアウト (デフォルトは 3m0s)
- `-- kubbelet-dir string`: kubelet の内部状態のホストの場所 (デフォルトは /var/lib/kubbelet") 。
- `--log-format string`: Astra Trident のログ形式 (text,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 が使用する従来の PVC の名前は、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でください (デフォルトは true) 。
- `-- silent` : インストール中のほとんどの出力を無効にしなさい。
- `--trident-image string`: インストールする Astra Trident 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 のコミュニケーションに IPv6 を使用します。

logs

"logs" フラグを実行して、 Astra Trident からログを印刷することができます。

```
Usage:
  tridentctl logs [flags]
```

フラグ：

- `-a, --archive: 特に指定がない限り ' すべてのログを含むサポート・アーカイブを作成します
- `-h, --help`: ログのヘルプ。
- `-l, --log string`: アストラトライデントログを表示します。trident | auto | trident-operator | all （デフォルトは「 auto 」）のいずれかです。
- `-- ノード文字列 : ノードポッドログの収集元となる Kubernetes ノード名
- `-p, --previous `: 以前のコンテナインスタンスのログが存在する場合は、そのログを取得します。
- `-- sidecars` : sidecar コンテナのログを取得します

`send`

'end' コマンドを実行して 'Astra Trident からリソースを送信できます

```
Usage:
  tridentctl send [option]
```

利用可能なオプション： AutoSupport ： AutoSupport アーカイブをネットアップに送信します。

uninstall

uninstall フラグを実行して 'Astra Trident をアンインストールできます

```
Usage:
  tridentctl uninstall [flags]
```

flags:*`-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`

「 update 」コマンドを実行して、 Astra Trident のリソースを変更できます。

```
Usage:
  tridentctl update [option]
```

利用可能なオプション : backend`: Astra Trident のバックエンドを更新します。

upgrade

'upgrade' コマンドを実行して 'Astra Trident のリソースをアップグレードできます

```
Usage:
tridentctl upgrade [option]
```

使用可能なオプション : volume 'NFS/iSCSI から CSI に 1 つ以上の永続ボリュームをアップグレードします

「バージョン」

「 rsion 」フラグを実行して、「 tridentctl 」のバージョンと Trident サービスを実行して印刷できます。

```
Usage:
  tridentctl version [flags]
```

flags:*`--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。

以前のバージョンのドキュメント

Astra Trident 22.01 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参照できます。

- "Astra Trident 21.10"
- "Astra Trident 21.07"

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 © 2024 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。